KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering

Lifted Inference and Learning in Statistical Relational Models

Guy Van den Broeck

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

January 2013

Lifted Inference and Learning in Statistical Relational Models

Guy VAN DEN BROECK

Supervisory Committee:
Prof. dr. ir. Joos Vandewalle, chair
Prof. dr. Luc De Raedt, supervisor
Prof. dr. ir. Hendrik Blockeel
Prof. dr. Jesse Davis
Prof. dr. ir. Herman Bruyninckx
Prof. dr. Adnan Darwiche
(University of California, Los Angeles, USA)
Prof. dr. Stuart Russell
(University of California, Berkeley, USA)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

January 2013

D/2013/10.705/5
ISBN 978-90-8649-590-0

Abstract

Statistical relational models combine aspects of first-order logic and probabilistic graphical models, enabling them to model complex logical and probabilistic interactions between large numbers of objects. This level of expressivity comes at the cost of increased complexity of inference, motivating a new line of research in *lifted probabilistic inference*. By exploiting symmetries of the relational structure in the model, and reasoning about groups of objects as a whole, lifted algorithms dramatically improve the run time of inference and learning.

The thesis has five main contributions. First, we propose a new method for logical inference, called *first-order knowledge compilation*. We show that by compiling relational models into a new circuit language, hard inference problems become tractable to solve. Furthermore, we present an algorithm that compiles relational models into our circuit language. Second, we show how to use first-order knowledge compilation for statistical relational models, leading to a new state-of-the-art *lifted probabilistic inference algorithm*. Third, we develop a *formal framework* for exact lifted inference, including a definition in terms of its complexity w.r.t. the number of objects in the world. From this follows a first completeness result, showing that the two-variable class of statistical relational models always supports lifted inference. Fourth, we present an algorithm for

approximate lifted inference by performing exact lifted inference in a relaxed, approximate model. Statistical relational models are receiving a lot of attention today because of their expressive power for learning. Fifth, we propose to harness the full power of relational representations for that task, by using *lifted parameter learning*.

The techniques presented in this thesis are evaluated empirically on statistical relational models of thousands of interacting objects and millions of random variables.

Beknopte Samenvatting

Statistische relationele modellen combineren aspecten van eerste-order logica met probabilistische grafische modellen. Dat laat hen toe om complexe logische en probabilistische interacties voor te stellen tussen een groot aantal objecten. Een nadeel van zulke hoge expressiviteit is de aanzienlijke toename van de complexiteit van inferentie. Dit was de motivatie voor een nieuw onderzoeksonderwerp rond *eerste-orde inferentie*. Door symmetrieën van de relationele structuur van de modellen uit te buiten, en door te redeneren over groepen van objecten in zijn geheel, hebben eerste-orde inferentie- en leeralgoritmes een sterk versnelde rekentijd.

Dit proefschrift heeft vijf belangrijke bijdragen. Ten eerste stellen we een nieuwe methode voor om aan logische inferentie te doen, met de naam *eerste-order kenniscompilatie*. We tonen aan dat bepaalde moeilijke inferentietaken oplosbaar worden door relationele modellen te compileren naar een nieuwe circuittaal. Bovendien beschrijven we een algoritme dat relationele modellen kan compileren in de circuittaal die we voorstellen. Ten tweede tonen we aan hoe eerste-order kenniscompilatie gebruikt kan worden voor statistische relationele modellen, met een nieuw geavanceerd *eerste-order probabilistisch inferentialgoritme* tot gevolg. Ten derde ontwikkelt dit proefschrift een formeel

raamwerk voor exacte eerste-order inferentie, met onder andere een definitie van eerste-order inferentie in termen van de complexiteit van inferentie m.b.t. tot het aantal objecten in de wereld. Hieruit volgt een eerste volledigheidsresultaat, dat aantoont dat statistische relationele modellen met twee logische variabelen altijd eerste-order inferentie toelaten. Ten vierde presenteren we een nieuw algoritme voor *benaderende eerste-order inferentie*, door exacte inferentie in een vereenvoudigd, benaderend model. Er is tegenwoordig veel belangstelling voor het leren van statistische relationele modellen, dankzij hun expressiviteit. Ten vijfde stellen we voor om hiervoor de volledige kracht van de relationele voorstelling te benutten, door gebruik te maken van een *eerste-order leeralgoritme* voor de parameters van het model.

De technieken beschreven in dit proefschrift worden empirisch geëvalueerd op statistische relationele modellen met duizenden interagerende objecten en miljoenen toevalsvariabelen.

Acknowledgements

This PhD has been a fun, exciting and gratifying experience, and at the same time challenging and humbling. It gave me the chance to work with some amazing people, whom I would like to thank.

I am very grateful to my supervisor Luc De Raedt. Luc always encouraged me to "go for gold" and continuously raised the bar. In many ways, he was exactly the kind of advisor I needed. He does not micromanage, but gives people the freedom to explore and develop their own ideas. Yet, at the right times, he knows exactly which direction to guide you in. In August 2010, I was explaining him Prof. Darwiche's work on knowledge compilation. He asked me, "Can you lift this?", and that was the end of our meeting. In my second year, he selflessly challenged me to write a paper all by myself. I was not happy being given that task and it was scary, but in retrospect, I am very grateful he did. I was certainly not the easiest and most obedient person to work for him. Fortunately, Luc has an astounding patience with students who think they know it all.

I would like to thank Prof. Vandewalle for chairing the jury and Profs. Blockeel, Davis, Bruyninckx, Darwiche and Russell for their insightful questions and comments when discussing earlier versions of this text. I am honored to have Profs. Darwiche and Russell as my external jury members. Their work was

an important inspiration for this thesis. It is comforting to know that they are tackling some very fundamental problems in their research; that they maintain a standard of rigor and quality that will make their work stand the test of time. Jesse is much more than a member of my jury. I want to thank him for all the collaboration, his advice and friendship. Go Packers!

I could not have done this work without the help of my coauthors. I hope I do justice to their work as I present it here. I want to thank Wannes M. in particular, for being a LaTeX guru, for helping me develop and maintain the software implementation of the algorithms in this thesis, and simply for being a good friend. I also want to thank my 'real boss', the Research Foundation-Flanders (FWO-Vlaanderen), for their financial support.

Research is a team sport. Therefore, I am happy that so many people took the time to share their ideas with me. I very much enjoyed discussions at the whiteboard (or at the bar) with Ingo, Kurt D., Daan, Joris, Mathias N., Arthur, Kristian, Udi and many others. Jan V. H., Maurice, Joris and Jonas were kind enough to give feedback on this text. Especially Jan has a great future as a journal editor.

I had the pleasure of sharing an office with Laura, Kurt D., Ingo, Martijn, Robby, Sam, Elena, Khaled, Davide, Daniele and Francesco. I enjoyed many lunches with current and past Alma-goers, including Angelika, Anton, Ben, Behrouz, Bogdan, Daan, Davide, Eduardo, Jan V. H., Jonas, Joris, Kurt D. G., Leander, Martin, Mathias, Matthijs, McElory, Siegfried, Thanh, Tias and Vladimir. I greatly value the people that helped me at the beginning of my PhD. This is when you are most insecure, trying to navigate a new environment and wondering whether you will ever be able to write a good paper. Therefore, I want to especially thank Laura and Kurt D. for their support then.

I am grateful to be given the chance to move to Los Angeles for five months. The UCLA-crew made it that much more enjoyable. Many thanks to Arthur, Elias, Khaled, Shahin, Suming and Tiansheng. In particular Arthur was an excellent host, a true genius and a good friend. I look forward to seeing them again.

Finally, I could not have made it here without my friends and family. Let me just mention Seppe, Wannes, Leentje, Lynn, Lore, Sophie, Stijn, Aline, Ruth, Klaartje, Leen, Michiel, Kenzo and Cedric for being there in the most difficult times. Some of them even had to live with me. I want to express my gratitude to Dirk for cultivating my interest in science from a very early age, explaining to me at around age 11 how our microwave worked, and where those bubbles

came from when my mom was boiling water. It is all just molecules bouncing around!

Finally there are the people I want to make most proud. My loving mother is incredibly strong and truly exceptional. She created the perfect environment for me to succeed. I am lucky to have Irma in my life. Thank you for all your jokes, your support and your love.

Bedankt!

Guy Van den Broeck
Leuven, January 2013

Contents

List of Symbols

T	True
F	False
$\wedge$	Conjunction, AND
$\vee$	Disjunction, OR
$\neg$	Negation, NOT
$\Rightarrow$	Material implication
$\equiv$	Logical equivalence
x	Random variable
q	Query
e	Evidence
ω, db	Possible world, interpretation, database
Ω, DB	Data set, set of interpretations, set of databases
$f(X)$	Potential

$\psi(\omega)$	Feature
Ψ	Set of features
$[V/v]$	Substitution, constraint set solution
θ	Parameter-variable, substitution, constraint set solution
Θ	Set of substitutions
ϕ, ψ, χ	Logical formula
γ	Clause
a	Atom
a_g	Ground atom
l	Literal
L	Set of literals
$\mathcal{MOD}_\phi$	Set of models of ϕ
p	Predicate
X, Y, Z	Logical variable
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	Set of logical variables
a, b, c	Constant symbol
d	Set of constant symbols
t	Term
D, F	Domain variable
$\mathbf{D}, \mathbf{F}$	Set of domain variables
V	Logical or domain variable
$\mathbf{V}$	Set of logical or domain variables
cs	Constraint set
Δ	Logical theory
$\mathrm{gr}(\phi)$	Grounding of ϕ

Σ, C	Circuit
$\mathbf{L}_1 \leq \mathbf{L}_2$	$\mathbf{L}_1$ is at least as succinct as $\mathbf{L}_2$
$\mathbf{L}_1 \not\leq \mathbf{L}_2$	$\mathbf{L}_1$ is less succinct than $\mathbf{L}_2$
$\mathbf{L}_1 < \mathbf{L}_2$	$\mathbf{L}_1$ is strictly more succinct than $\mathbf{L}_2$
$\phi \perp\!\!\!\perp \psi$	ϕ is independent of ψ
π	Permutation
$\mathrm{bvars}(\phi)$	Set of bound logical variables in ϕ
$\mathrm{atom_c}(\phi)$	Set of constrained atoms in ϕ
$\mathcal{P}$	Partition
E	Partition element
w	MLN weight
weight	Literal weight
$\mathrm{w_T}$	Positive predicate weight function
$\mathrm{w_F}$	Negative predicate weight function
PI	Class of probabilistic inference problems
$\mathcal{S}$	Class of sentences
$\mathcal{Q}$	Class of queries
$\mathcal{E}$	Class of evidence
$\mathcal{A}$	Single ground atoms
$\mathcal{T}$	Terms of ground literals
$\mathcal{T}_{0,1}$	Terms of ground literals with arity 0 or 1
KLD	Kullback–Leibler divergence
B_n	nth Bell number

List of Algorithms

1

Introduction

1.1 Artificial Intelligence

Driven by a growing number of digital devices with a growing amount of computing power, Artificial Intelligence (AI) is becoming increasingly important. It already plays a major role in the technology we use every day. There are many definitions of what exactly AI is. Its most ambitious definition is that AI tries to build *machines that think like humans*. This definition raises many philosophical questions and is an inspiration to many artists and aspiring researchers. However, it defines AI as a challenge so big it seems impossible to realize it. A more practical definition says that AI builds *machines that perform tasks that seemingly require intelligence*. This definition of AI lowers the bar, requiring machines to *behave* intelligently.

What constitutes intelligent behavior is again debatable, and a moving target. When the goal is to imitate human behavior (Turing, 1950), intelligent machines should have limited capabilities. Alternatively, the goal is to achieve rational behavior. After all, very few people are grandmasters in chess. In practice, AI is often about building machines that perform a specific task that people thought could never be performed by a machine. Once this is achieved, expectations

shift and the field moves on to solve the next challenge. This continuous drive to increase the range of AI applications has resulted in many AI systems being used today. AI system are sometimes called *rational* or *intelligent agents* (Russell and Norvig, 2010), to emphasize their embodiment in the real world and distinguish them from other software systems. Intelligent agents perceive their environment, adapt themselves based on sensory input, act in the environment and thereby modify it.

Examples of tasks dealt with by AI systems include planning, scheduling, theorem proving, understanding natural language, game playing, medical diagnosis, drug discovery, fraud detection and robotics. Solving these tasks often involves solving problems such as knowledge representation, automated reasoning, speech recognition and computer vision. Some recent successes of AI include Watson (Ferrucci et al., 2010), who beat the best humans at the question-answering game of Jeopardy!, self-driving cars, that share the road with human drivers, and the Deep Blue system (Campbell, Hoane, and Hsu, 2002), which beat the human world champion at chess.

1.2 Machine Learning and Automated Reasoning

This dissertation is situated in the subfields of AI called *machine learning* (ML) and *automated reasoning*.

The goal of machine learning is to build AI systems that *improve their performance* on a task with experience. This allows us to extend AI beyond human knowledge and what human experts can program in a computer. Machine learning systems typically learn concepts from examples or learn skills, either by observing an expert, or by interacting with the environment. For example, a machine learning system can predict which web page you want to visit based on your search query, filter your spam emails, or assist doctors with diagnosing patients based on their medical records.

Many machine learning algorithms learn *models* of the world. These are representations of how the concept being learned is a function of the system's inputs. Once a model has been learned, it is primarily used for one of two purposes. First, a rational agent can predict its environment with the model and use this information to decide which actions to take. Second, the model can assist humans in understanding the environment. In some cases, humans can

interpret and understand the learned model. Alternatively, the learned model can be used to provide predictions to a user. In medical diagnosis, for instance, a machine learning algorithm can learn a model to predict how successful each treatment will be.

In either case, using a learned model requires making predictions, that is, analyzing how modifying the system's inputs will change its outputs, and deducing what exactly are the consequences implied by the model. This problem is investigated in another subfield of AI, called *automated reasoning*. For learning a model in the first place, algorithms often depend on predictions provided by an automated reasoning algorithm. The key contributions of this thesis are in this field.

1.3 Logic and Probability

Logic and *probability* are two cornerstones of AI. Logic as a field predates AI by thousands of years and is the standard formalism for knowledge representation. It is particularly suitable for automated reasoning and has therefore been an essential tool since the early days of AI. Probability gained importance in AI later on, when it was realized that the environment that rational agents are embedded in is inherently uncertain and that machine learning models should take this uncertainty into account.

Within logic, one can distinguish further between *propositional* and *first-order* logic. Propositional logic expresses knowledge about a single set of properties of the world, not associated with objects in the world. These properties are represented by propositional variables. For example, propositional logic can express that when it is raining (variable rain) and the sun is shining (variable sun), then you can see a rainbow (variable rainbow). In contrast, first-order logic expresses knowledge about objects in the world, of which there can be many. This knowledge is expressed in terms of a set of properties of each object and the relations that hold between objects. For example, first-order logic can express that if an object X is a human (represented by human(X)), then X is mortal (represented by mortal(X)). This is a statement about a large, possibly infinite number of objects X in the world. Hence, first-order logic is a more expressive formalism, more suitable for expressing structured knowledge.

Many subfields of AI have moved from logical to probabilistic and from

propositional to first-order or relational approaches. As an example, consider the field of planning. Classical planning problems are expressed in a first-order plan language such as STRIPS (Fikes and Nilsson, 1972). Extending these with probabilistic concepts lead to the field of probabilistic planning (Kushmerick, Hanks, and Weld, 1995), with its probabilistic first-order planning language PPDDL (Younes et al., 2005). From another starting point, factored Markov decision processes (Bellman, 1957; Puterman, 1994), which correspond to propositional probabilistic planning problems, were extended with the expressivity of first-order logic (Poole, 1997; Boutilier et al., 2000) to arrive at the same destination.

	Logical	**Probabilistic**
Propositional	Symbolic ML, Decision tree induction, Rule learning	Statistical ML
First-Order	Inductive logic programming, Relational learning	*Statistical relational learning, Probabilistic logic learning*

(a) Machine learning

Propositional	Propositional logic, Rules, Decision trees	Probabilistic graphical models, Bayesian and Markov networks
First-Order	First-order logic, Logic programs Relational databases	Probabilistic logics, *Statistical relational models Probabilistic logic programs*

(b) Knowledge representation and machine learning models

Propositional	Resolution, SAT solving, Knowledge compilation	Graphical model inference, Knowledge compilation
First-Order	Resolution, Theorem proving, *First-order knowledge compilation*	*Lifted probabilistic inference, First-order knowledge compilation*

(c) Automated reasoning

Table 1.1: A comparison of fields and techniques along the axis logical vs. probabilistic and propositional vs. first-order approaches.

Machine learning has followed a similar course, as shown in Table 1.1a. At different points in time, the field focused on learning either *propositional logical* models (e.g., rules and decision trees) or *propositional probabilistic* models (e.g., probabilistic graphical models). Inductive logic programming (Muggleton and De Raedt, 1994) and relational learning (De Raedt, 2008) are subfields of machine learning that introduced the problem of learning *first-order logical* models. This dissertation is situated in the subfields of statistical relational learning (Getoor and Taskar, 2007) and probabilistic logic learning (De Raedt et al., 2008).

These fields are both concerned with learning *first-order probabilistic* models, but reached this objective separately. Statistical relational learning extends statistical machine learning methods with concepts from relational databases, making them first-order, whereas probabilistic logic learning adds probabilistic primitives to first-order methods from inductive logic programming.

Machine learning and knowledge representation *models* can be classified into logical and probabilistic models as well. Each subfield of machine learning in Table 1.1a learns models from a corresponding knowledge representation formalism in Table 1.1b. We can again observe the shift from propositional to first-order and from logical to probabilistic models. Statistical relational models extend probabilistic graphical models with first-order logical relations and probabilistic logic programs extend logic programs with probabilities. Both are high-level representation languages for probabilistic models of structured data.

In turn, each knowledge representation formalism has an associated subfield of automated reasoning, as shown in Table 1.1c. The combination of first-order and statistical models for machine learning recently gave rise to a new automated reasoning problem: first-order probabilistic inference. This area of automated reasoning is called *lifted probabilistic inference* (Poole, 2003). It is different from propositional reasoning because its algorithms reason about groups of objects and exploit the symmetries in first-order models. It is different from logical reasoning because its algorithms deal with the uncertainty found in probabilistic models.

Most of the areas mentioned in Table 1.1 are discussed in more detail throughout this dissertation. The key contribution of this dissertation is a new approach to lifted probabilistic inference, called first-order knowledge compilation. This approach, and other areas we make contributions to, are shown in italics.

1.4 Motivation and Problem Statement

This thesis considers the problem of automated reasoning about first-order probabilistic information. Here, first order refers to the fact that the uncertainty information is associated with properties of objects in the world, or with the relations between objects. This type of information is different from propositional probabilistic information, where uncertainty is associated with variables not pertaining to any specific object.

We will now give examples of the types of problems, models and inference tasks that are being investigated in this thesis.

Example 1.1. Assume we are investigating a rare genetic defect, which presents itself in one in every billion people. Here we have probabilistic information ("one in every billion") which is a single statement that applies to multiple entities in the world (all people). It can be captured in a statistical relational model that intuitively states the following.

> *For all people X, the probability that X has the defect is* 10^{-9}.

The probability that somebody in the world has this defect is high. It presents in at least one in seven billion people with a probability higher than 99.9%. Answering the question what is the probability that more than five people have the disease is more difficult. Yet, using basic statistics and combinatorics, one can find that it is around 70%. These are examples of inference tasks in the statistical relational model.

Another example task which we would want an intelligent agent to solve is the following instance of the *prosecutor's fallacy* (Thompson and Schumann, 1987).

Example 1.2. The DNA found at a crime scene is compared to a database of 20000 people's DNA samples and matches one. The accused argues that one in 10000 tests is a false positive. This problem can intuitively be modeled in a statistical relational model as follows.

> *For all people X, if X is guilty, his DNA matches.*
> *For all people X, if X is not guilty, his DNA matches with probability* 0.0001.

It has been shown that many people would convict someone based on this statistical evidence, even though the probability of getting at least one false positive match among 20000 people is $1 - (1 - 1/10000)^{20000} \approx 86\%$. This is an example of a concrete inference task.

The above examples illustrate that compact statistical relational models can make general statements about a large number of entities. However, they do not yet show the full potential of relational representations. Statistical relational models can be much more sophisticated, also taking into account relations

between objects. They have, for example, been used to predict the topics of web pages based on the hyperlinks between them (the linked relation), to model the success of sending advertisements to people based on their social network (the friends relation) and to model the regulatory system of cells, detailing how proteins interact (e.g., the inhibits relation).

The fields of statistical relational learning and probabilistic logic learning have given us the necessary tools to express these problems, and much more complex ones, in a formal language. However, while supporting model specifications at an abstract, first-order logic level, inference is typically performed at the level of concrete ground instances of the models, i.e., at the propositional level. We lack the basic tools that allow a machine to automatically and efficiently reason about this type of information and come to intelligent conclusions. *Lifted inference* algorithms attempt to alleviate this situation. Several of these algorithms have been proposed recently, but still have limited capabilities.

Lifted inference algorithms can scale much better than propositional algorithms. In this thesis, we will see certain models for which lifted algorithms can deal with worlds containing *thousands of objects*, modeling *millions of random variables*. For the same models, propositional inference algorithms can often reason about no more than a few objects in the world.

Our long-term objective is to build the tools that allow a rational agent to learn statistical relational models of the world, dealing with a large number of objects, and that let it efficiently reason with these models, predict future observations and estimate the consequences of its actions. To achieve this long-term objective, this dissertation has three goals. The *first goal* is to develop lifted inference algorithms, both exact and approximate, that can deal with a breadth of statistical relational models. The *second goal* is to theoretically analyze which broad classes of statistical relational models and inference tasks permit lifted inference. The *third goal* is to look at the implication of these new automated reasoning algorithms on the problem of learning statistical relational models.

1.5 Thesis Contributions

We identify *five main contributions* of our work, one in the area of knowledge compilation, and four in the area of statistical relational learning. We will now briefly list these contributions.

1. First-order knowledge compilation into negation normal form

Our first main contribution is a new automated reasoning problem, namely *first-order knowledge compilation* into *first-order negation normal form* (FO-NNF) circuits. This contribution comprises

- a definition of the syntax and semantics of the FO-NNF *circuit language*,
- a definition of the syntax of *tractable subsets* of the FO-NNF language,
- an analysis of the *properties* of these circuits, in terms of their completeness, succinctness and support for queries and transformations, and
- an algorithm to *compile* logical theories into tractable subsets of FO-NNF.

This work upgrades propositional knowledge compilation and NNF circuits to the first-order setting. FO-NNF circuits represent theories in first-order logic. They are used to efficiently answer queries about a knowledge base, such as consistency checking and model counting. These contributions are in the area of automated logical reasoning and knowledge representation.

2. Lifted probabilistic inference by weighted first-order model counting and first-order knowledge compilation

Our second main contribution is a new exact lifted probabilistic inference algorithm that reduces probabilistic inference to a *weighted first-order model counting* problem which can be solved by *first-order knowledge compilation*.

This technique is able to perform lifted inference for a large class of statistical relational models and is arguably the state of the art in exact lifted inference. It is a first-order generalization of the weighted model counting and knowledge compilation approach to propositional probabilistic inference. Many advantages of propositional knowledge compilation carry over to this approach. Because it lifts a *logical inference* algorithm, as opposed to a probabilistic one, it naturally exploits local structure such as context-specific independencies and determinism in lifted inference.

3. A formal framework for lifted probabilistic inference as a well-defined problem

Our third contribution is a *formal framework* for lifted probabilistic inference, which includes

- a first formal *definition* of lifted inference, called *domain-lifted* inference,
- the notion of *completeness* of a lifted inference algorithm,
- the notion of *liftability* of a class of inference tasks, and
- several *theoretical results*.

Domain-lifted inference defines lifted inference in terms of its complexity w.r.t. the number of objects in the world. The notions of completeness and liftability allow us to identify classes of problems that lifted inference is guaranteed to work on, and classes for which no efficient lifted inference algorithm can exist.

This *formal framework* is one of the main contributions of this thesis. We use it to prove the following two claims for several popular statistical relational languages.

Claim. The problem of computing single marginal probabilities in quantifier-free models with *up to two logical variables per formula* is amenable to lifted inference.

Claim. The problem of computing *conditional probabilities* is not liftable by any inference algorithm, unless the evidence consists solely of *unary atoms* and *propositions*.

The former claim is a first completeness and liftability result for a non-trivial class of lifted inference tasks. The latter is a precise characterization of which conditional probability queries can efficiently be answered by lifted inference algorithms.

4. A lifted relax, compensate & recover framework for approximate lifted inference

Our fourth main contribution is a new *approximate lifted probabilistic inference* algorithm, which generalizes the relax, compensate & recover (RCR) framework to the first-order setting.

This algorithm performs approximate lifted inference by performing exact lifted inference in a simplified first-order model. The RCR framework provides a *unifying semantics* for existing approximate lifted inference algorithms, such as lifted belief propagation and lifted mini-buckets, in terms of relaxing equivalences in the model and compensating for the relaxations. In addition, it introduces a *new set of lifted algorithms* in its spectrum of approximations, including a family of lifted joingraph propagation and generalized belief propagation approximations.

5. The application of exact lifted inference to lifted learning

Our fifth contribution is the application of exact lifted inference techniques to the problem of *lifted learning* of the *parameters* of a statistical relational model.

There are two benefits to this approach

- Lifted learning learns models that optimize the *exact likelihood* of the parameters where this was previously intractable.
- Lifted learning has a complexity that is *polynomial* in the size of each training database, whereas existing algorithms that optimize the exact likelihood of the parameters have an exponential complexity.

We show for the first time that exact lifted inference techniques can be applied to learning models of *real-world* data sets.

1.6 Structure of the Thesis

The line of work presented in this thesis is a first-order generalization of the propositional work on logical reasoning, probabilistic inference and learning. We review these *propositional foundations* in **Chapter 2**.

The next three chapters are concerned with exact lifted inference. **Chapter 3** introduces the fundamental data structure used in this dissertation, namely FO-NNF *circuits*, and describes their properties in terms of which queries and transformations can efficiently be performed on them. **Chapter 4** presents an algorithm for *compiling* a first-order knowledge base into FO-NNF circuits that permit tractable inference. **Chapter 5** starts with a review of statistical relational models and inference algorithms. It then wraps up the work on exact inference by giving a formal definition of lifted probabilistic inference and showing how to use compilation into FO-NNF for lifted inference in *probabilistic* models. Chapters 3, 4 and 5 mainly draw upon the following publications, but are extended with more recent results.

> G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011a). "Lifted probabilistic inference by first-order knowledge compilation". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. Menlo Park, California, pp. 2178–2185
>
> G. Van den Broeck and J. Davis (2012). "Conditioning in first-order knowledge compilation and lifted probabilistic inference". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence,* Palo Alto, California, USA

Chapter 6 puts the problem of exact lifted inference on more formal grounds. Using the definition of domain-lifted inference, it defines the notions of completeness and liftability and proves several theoretical results. This chapter is based on the following publications:

> G. Van den Broeck (2011b). "On the completeness of first-order knowledge compilation for lifted probabilistic inference". In: *Advances in Neural Information Processing Systems 24 (NIPS),* pp. 1386–1394
>
> M. Jaeger and G. Van den Broeck (2012). "Liftability of probabilistic inference: Upper and lower bounds". In: *Proceedings of the 2nd International Workshop on Statistical Relational AI,*

Chapter 7 is concerned with Lifted Relax, Compensate & Recover (RCR), which is a new approximate lifted inference algorithm. The chapter first explains the propositional RCR algorithm by applying it to grounded statistical relational

models. It then lifts each step of this algorithm to the first-order case. The work in this chapter was previously published as

> G. Van den Broeck, A. Choi, and A. Darwiche (2012). "Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference". In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*

Chapter 8 presents a lifted generative parameter learning algorithm. First, it describes a general algorithm that employs a black-box lifted probabilistic inference algorithm for learning. Second, it proposes a specific lifted learning algorithm that uses first-order knowledge compilation to efficiently optimize the likelihood of the model. The work in this chapter is under review as

> G. Van den Broeck, W. Meert, and J. Davis (2012). *Lifted parameter learning for Markov logic*. (submitted)

Scattered throughout the chapters are short **intermezzos** that describe some of the work done in the context of this thesis that was not on the topic of lifted inference. These intermezzos talk about *decision-theoretic probabilistic logic programs*, *algebraic model counting* and *inference in probabilistic logic programs by weighted model counting and max-SAT*, which were published as

> G. Van den Broeck, I. Thon, M. van Otterlo, and L. De Raedt (2010). "DTProbLog: A decision-theoretic probabilistic Prolog". In: *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence,* Menlo Park, California, pp. 1217–1222
>
> A. Kimmig, G. Van den Broeck, and L. De Raedt (2011). "An algebraic Prolog for reasoning about possible worlds". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence,* pp. 209–214
>
> A. Kimmig, G. Van den Broeck, and L. De Raedt (Nov. 2012a). "Algebraic Model Counting". In: arXiv:1211.4475.
>
> D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt (2011a). "Inference in probabilistic logic programs using weighted CNF's". In: *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI),* pp. 211–220
>
> D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt (2012b). "Inference

and learning in probabilistic logic programs using weighted Boolean formulas". In: (submitted)

Other work that is not covered in these chapters and intermezzos is listed at the end of the thesis, in the **list of publications**.

A **concluding chapter** *summarizes* the thesis, discusses its implications and provides an outlook on *future research*.

The algorithms presented in this thesis are available as open source software at `http://dtai.cs.kuleuven.be/wfomc/`.

2

Propositional Foundations

This chapter lays out the foundations of the work presented in this dissertation. Its common theme can be described as "*probabilistic inference by logical reasoning*". This chapter reviews the literature on that topic. We describe most concepts informally, with examples, and refer to the literature for formal definitions.

We start by reviewing standard notions of *propositional logic* in **Section 2.1**. **Section 2.2** presents a general framework for logical inference, called *knowledge compilation*. We then switch our attention to the representation of probabilistic models in **Section 2.3**, which deals with *probabilistic graphical models* such as Bayesian networks and factor graphs. The logical approach to probabilistic inference has been applied to graphical models. We review this work in **Section 2.4**. In essence, it reduces probabilistic inference to a (weighted) *model counting* problem. In particular, we will discuss the use of knowledge compilation for solving this type of inference tasks. Whereas the previous techniques perform exact inference, **Section 2.5** discusses techniques for *approximate probabilistic inference* in graphical models. Finally, **Section 2.6** deals with the problem of learning the parameters of a probabilistic graphical model from data.

Each of the following chapters in this dissertation lifts one or more sections from this chapter to the first-order case. The mapping is as follows.

- Propositional logic and logical inference by knowledge compilation in Sections 2.1 and 2.2 are lifted to first-order logic and first-order knowledge compilation in Chapters 3 and 4.
- Probabilistic graphical models in Section 2.3 and exact probabilistic inference (by knowledge compilation) in Section 2.4 are lifted to statistical relational models and lifted probabilistic inference (by first-order knowledge compilation) in Chapters 5 and 6.
- Approximate probabilistic inference, specifically the relax, compensate and recover algorithm in Section 2.5 is generalized to perform lifted inference in statistical relational models in Chapter 7.
- The generative parameter learning task for probabilistic graphical models in Section 2.6 is lifted to the same task for statistical relational models in Chapter 8, which also presents a lifted weight learning algorithm to solve that task.

2.1 Propositional Logic

Propositional or Boolean logic is a formal language for modeling and reasoning about the truth of propositions, or *sentences*. Propositions are statements about the world, which can be true (T) or false (F). These sentences are constructed from propositional *variables* and logical *connectives* such as negation (NOT, $\neg$), conjunction (AND, $\wedge$), disjunction (OR, $\vee$) and implication ($\Rightarrow$). A *literal* is a propositional variable or its negation. A *theory* or *knowledge base* is a conjunction of sentences. An *interpretation* is a truth assignment to all propositional variables. An interpretation *satisfies* a sentence Σ when it evaluates to true for that truth assignment to the variables. The interpretations that satisfy a sentence are the *models* of the sentence.

Example 2.1. The sentence $\text{sun} \wedge \text{rain} \Rightarrow \text{rainbow}$ encodes that when there is sun (variable sun is true) and it is raining (variable rain is true), there is a rainbow (variable rainbow is true). Every interpretation that assigns true to rainbow is a model of this sentence. The interpretation $\text{sun} = \text{T}$, $\text{rain} = \text{T}$ and $\text{rainbow} = \text{F}$ is not a model.

There are many logical inference tasks one considers in a knowledge base. The best-known problem is *consistency*, or *satisfiability* (SAT) checking, which

checks whether the knowledge base has a model. Its dual problem is *validity* checking, which checks whether every interpretation is a model. Both of these are special cases of the *model counting* task, which counts the number of models of the knowledge base. The *maximum satisfiability* problem (MAX-SAT) is an optimization problem concerned with finding the largest number of clauses (disjunctions of literals) in a knowledge base that can be satisfied by an interpretation.

Applications of propositional logic appear in many fields of computer science. In artificial intelligence, planning problems can be solved by answering a Boolean satisfiability problem (Kautz, Selman, et al., 1992; Rintanen, 2009). Inference engines for a variety of different knowledge representation formalisms reduce inference tasks to the satisfiability problem. Examples are inference in logic and answer set programs (Lin and Zhao, 2004) and modal and description logics (Sebastiani and Tacchella, 2009). The application we will focus on in Section 2.4.1 is the use of model counting for probabilistic inference. Propositional logic is furthermore used in such fields as hardware design (Meinel and Theobald, 1998) and verification (Biere, 2009), software verification (Kroening, 2009), fault diagnosis (De Kleer, Mackworth, and Reiter, 1992; Darwiche, 2000), product configuration (Felfernig et al., 2004) and scheduling (Crawford and Baker, 1994).

2.2 Logical Inference by Knowledge Compilation

Knowledge compilation is a powerful technique for reasoning about propositional knowledge bases (Selman, Kautz, et al., 1991; Marquis, 1995; Cadoli and Donini, 1997; Darwiche and Marquis, 2002). Knowledge compilation transforms or compiles a logical theory into a circuit representation so that specific inference tasks can be performed in time polynomial in the size of the circuit. Once a circuit is compiled, it can be reused to efficiently answer a large number of queries. While the compilation step may be computationally expensive, it is a one-time cost that can be amortized over all subsequent queries.

Circuit reuse is one of the main computational advantages of compilation. Transformations modify the compiled circuit, enabling it to answer additional queries. Applying an efficient transformation is much cheaper than modifying the original theory and recompiling. Conditioning, which is such a transforma-

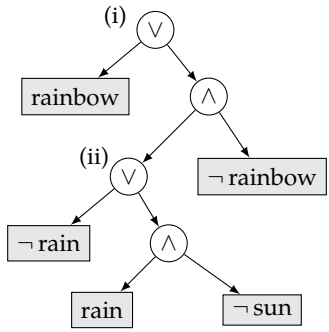


Figure 2.1: Propositional d-DNNF circuit for the sentence sun ∧ rain ⇒ rainbow

tion, is the topic of Section 2.2.2. It updates a knowledge base to incorporate information about the truth values of a set of literals in the theory.

A circuit language is a set of circuits that have a shared set of properties. Darwiche and Marquis (2002) give an overview of propositional circuit languages and their properties in a *knowledge compilation map*. All circuit languages in the knowledge compilation map are subsets of the *negation normal form* (NNF) language. A NNF circuit is a directed, acyclic graph where the leaves are labeled with either a literal (e.g., x or ¬ x) or a truth value. The inner nodes represent formulae in propositional logic, either conjunctions or disjunctions.

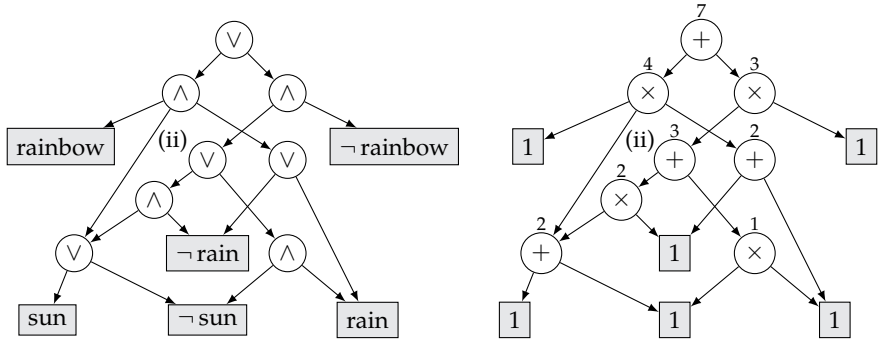
A language of particular interest is deterministic decomposable negation normal form (d-DNNF) circuits (Darwiche, 2001b). A d-DNNF circuit restricts a NNF circuit such that all the conjunctions are decomposable and all the disjunctions are deterministic. In a *decomposable conjunction*, each pair of conjuncts (child nodes) must be independent (i.e., they cannot share any variables). In a *deterministic disjunction*, only one disjunct (child node) can be true at the same time (i.e., their conjunction is unsatisfiable). Other notable circuit languages in knowledge compilation are ordered binary decision diagrams (OBDD) (Bryant, 1986) and sentential decision diagrams (SDD) (Darwiche, 2011). These are special cases of d-DNNF circuits that support more tractable queries and transformations at the cost of a potentially larger circuit size.

Example 2.2. Figure 2.1 shows a d-DNNF circuit. The node labeled (i) represents the theory sun ∧ rain ⇒ rainbow. It can be read in two ways. Top-down, it represents a sentence in propositional logic. Bottom-up it represents a circuit where the literal leafs are the input wires and the root node is an output wire.

Knowledge compilation has been used to solve inference tasks in many different fields, such as hardware design (Meinel and Theobald, 1998), fault diagnosis (Elliott and Williams, 2006), conformant planning (Palacios et al., 2005) and databases (Dalvi, Schnaitter, and Suciu, 2010). One application of knowledge compilation we will explore further in Section 2.4.2 is probabilistic inference (e.g., Chavira, Darwiche, and Jaeger (2006); Chavira and Darwiche (2008); Fierens et al. (2011a)).

2.2.1 Model Counting

Different circuit languages support different polytime queries. As an example of such a query, we will now describe how the model counting task can efficiently be solved in a smooth d-DNNF (sd-DNNF) circuit. Smooth circuits have the property that any pair of disjuncts mentions the same set of propositional variables.



(a) sd-DNNF circuit (b) Model counting arithmetic circuit

Figure 2.2: Circuits for the sentence sun ∧ rain ⇒ rainbow

Example 2.3. The d-DNNF circuit of Figure 2.1 is not smooth. The left child of the node labeled (i) mentions the set of variables {rainbow} whereas the right child mentions {rainbow, rain, sun}. Figure 2.2a depicts an sd-DNNF circuit for the same sentence.

An sd-DNNF circuit can be transformed into an arithmetic circuit that computes the model count of the sentence (Darwiche, 2001b). This is done by replacing

literals by the constant 1, conjunctions by multiplications and disjunctions by additions. Evaluating the arithmetic circuit outputs the model count.

Example 2.4. Figure 2.2b depicts the arithmetic circuit for the sd-DNNF of Figure 2.2a. The model count of each subcircuit is shown above its root node. The model count of the entire circuit, and therefore of sentence sun $\wedge$ rain $\Rightarrow$ rainbow, is 7. We can verify this by checking that there are three variables and therefore $2^3 = 8$ interpretations, all of which are models, except for the one that assigns true to rain and sun but false to rainbow.

2.2.2 Conditioning

Conditioning is one of the most basic transformations that can be applied to a circuit. All circuits in the propositional knowledge compilation map (Darwiche and Marquis, 2002), including OBDDs, d-DNNFs and sd-DNNFs, support polynomial time conditioning (in the size of the circuit). Intuitively, conditioning updates a knowledge base to reflect information about the truth values of specific propositions. More formally, conditioning a logical theory Σ on a term γ is denoted as $\Sigma|\gamma$, where $\gamma = l_1 \wedge l_2 \wedge \cdots \wedge l_n$ and each l_i is a literal. For each positive (negative) l_i, conditioning replaces all positive (negative) occurrences of l_i in Σ with *true* and all negative (positive) occurrences by *false*.

Example 2.5. Conditioning sun $\wedge$ rain $\Rightarrow$ rainbow on $\neg$ rainbow results in the theory sun $\wedge$ rain $\Rightarrow$ F, or equivalently $\neg$ sun $\vee \neg$ rain.

There are two ways of obtaining a circuit for the sentence $\Sigma|\gamma$. The first way is to condition the sentence Σ on term γ and then compile the result. The second way is to compile a circuit for Σ and condition it on γ. When we want to pose queries about many different $\Sigma|\gamma_i$ sentences, conditioned on different terms, the latter approach can be much more efficient. Since compilation is computationally demanding, conditioning the circuit, as opposed to the sentence Σ, permits reuse of the previously compiled circuit for inference over all $\Sigma|\gamma_i$. This circuit reuse is the motivation for the knowledge compilation approach.

Conditioning a NNF circuit on γ is achieved by replacing all terminal nodes that occur as variables in γ by T or F terminals. Darwiche and Marquis (2002) show that every language in the knowledge compilation map can efficiently be conditioned and that conditioning any circuit preserves its properties. That is, the conditioned circuit is in exactly the same circuit family as the original (i.e., unconditioned) circuit.

Example 2.6. Conditioning the circuits of Figures 2.1 and 2.2a on $\neg$ rainbow replaces the terminal rainbow with F and the terminal $\neg$ rainbow with T. After some simplifications, this is equivalent to the circuit rooted at the nodes labeled (ii). The arithmetic circuit rooted in node (ii) of Figure 2.2b computes the model count of $((\text{sun} \wedge \text{rain} \Rightarrow \text{rainbow})|\neg\,\text{rainbow})$, which is 3. These are all four interpretations of variables sun and rain, except for the one where both are true.

2.3 Probabilistic Graphical Models

Classical logic can only express concepts that are certainly true or certainly false. For many AI tasks, this is a severe limitation. This observation has led to the rise of many alternative knowledge representation formalisms. Logicians extended classical logic to formalize more commonsense knowledge. Examples are non-monotonic logics such as default logic (Reiter, 1980), autoepistemic logic (Moore, 1985) and logic programming (Clark, 1978; Gelfond and Lifschitz, 1988). Another approach has incorporated probability theory into knowledge representation and reasoning. Examples are probabilistic logics (Nilsson, 1986) and probabilistic graphical models (Pearl, 1988). The latter have become increasingly important in AI, in fields such as machine learning, computer vision and robotics.

A *probabilistic graphical model* is a concise representation of a probability distribution. It consists of a graph which represents the dependency structure of the distribution and an associated set of functions that quantitatively specify it. The next two sections briefly review two popular types of graphical models: Bayesian networks and factor graphs. For a more detailed treatment, see Koller and Friedman (2009) and Darwiche (2009).

2.3.1 Bayesian Networks

A *Bayesian network* is a directed acyclic graph that models a probability distribution as a product of conditional probabilities. The nodes in the graph represent propositional random variables. The Bayesian network defines a conditional probability distribution for every node in the graph, given its

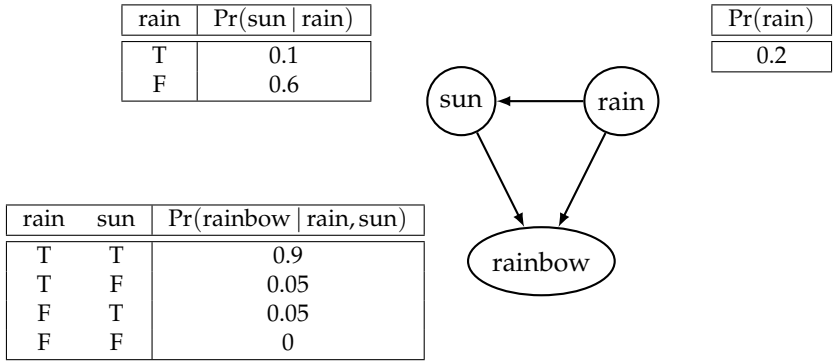


Figure 2.3: Bayesian Network

parents:

$$\Pr(x_1, \ldots, x_n) = \prod_i \Pr(x_i | \text{ parents}(x_i)). \tag{2.1}$$

These conditional probabilities are usually represented in a table.

Example 2.7. Figure 2.3 shows a Bayesian network of three variables: sun, rain and rainbow. The modeled probability distribution is factorized as

$$\Pr(\text{sun}, \text{rain}, \text{rainbow}) = \Pr(\text{rainbow} \mid \text{rain}, \text{sun}) \Pr(\text{sun} \mid \text{rain}) \Pr(\text{rain}),$$

where each factor on the right-hand side is given by a conditional probability table of the Bayesian network. Note that when the probability of a variable being true is p in the tables, the probability of the variable being false is implicitly $1 - p$. Table 2.1 shows the joint probability distribution encoded by this network.

An advantage of using a Bayesian network representation is that it concisely models a distribution over a large number of variables. The joint probability distribution of n variables is represented by a single table with 2^n rows (and $2^n - 1$ parameters). A row in such a table is called a *possible world*, representing a unique assignment of truth values to the variables. When the same distribution is modeled with a Bayesian network where each node has up to p parents, the number of rows in all conditional probability tables is less than $n2^p$, which can be a substantial reduction in size.

rain	sun	rainbow	Pr(rain, sun, rainbow)	
T	T	T	$0.2 \cdot 0.1 \cdot 0.9$	0.018
T	T	F	$0.2 \cdot 0.1 \cdot (1 - 0.9)$	0.002
T	F	T	$0.2 \cdot (1 - 0.1) \cdot 0.05$	0.009
T	F	F	$0.2 \cdot (1 - 0.1) \cdot (1 - 0.05)$	0.171
F	T	T	$(1 - 0.2) \cdot 0.6 \cdot 0.05$	0.024
F	T	F	$(1 - 0.2) \cdot 0.6 \cdot (1 - 0.05)$	0.456
F	F	T	$(1 - 0.2) \cdot (1 - 0.6) \cdot 0$	0
F	F	F	$(1 - 0.2) \cdot (1 - 0.6) \cdot (1 - 0)$	0.32

Table 2.1: Joint Probability Distribution Encoded by Figure 2.3

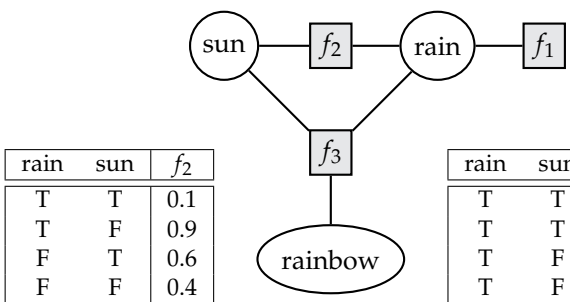
There are several inference tasks one can consider in a Bayesian network. It is a natural question to ask for *marginal posterior probabilities*. These are the probabilities of single variables given observations or evidence (a truth value assignment to some variables). Another task is finding the *most probable explanation* (MPE) of some evidence. It involves finding the most likely possible world where the evidence is true. A related task is finding the (partial) *maximum a posteriori* (MAP) hypothesis, which is the most likely assignment to a subset of the variables, given some evidence. We will discuss inference algorithms for answering these queries in Section 2.4.

2.3.2 Markov Random Fields and Factor Graphs

Bayesian networks are *directed* models that describe a generative process. This facilitates their use in modeling and knowledge representation. Given the right graph structure, each parameter in a Bayesian network has a clear semantics. However, when learning a graphical model from data, or when designing inference algorithms, it is often convenient to work with *undirected* models, such as Markov random fields or factor graphs.

A *Markov random field* or *Markov network* is an undirected probabilistic graphical model that represents a joint probability distribution over a set of random variables $x_1, \ldots x_n$. Each clique of variables X_i in the graph has a potential function, $f_k(X_i)$, associated with it. The distribution over possible worlds ω represented by a Markov network is:

$$\Pr(\omega) = \frac{1}{Z} \prod_i f_i(\omega_i) \qquad \text{with } Z = \sum_\omega \prod_i f_i(\omega_i) \tag{2.2}$$



rain	f_1
T	0.2
F	0.8

rain	sun	f_2
T	T	0.1
T	F	0.9
F	T	0.6
F	F	0.4

rain	sun	rainbow	f_3
T	T	T	0.9
T	T	F	0.1
T	F	T	0.05
T	F	F	0.95
F	T	T	0.05
F	T	F	0.95
F	F	T	0
F	F	F	1

Figure 2.4: Factor Graph

where ω_i is the state of the ith clique (i.e., the state of the variables that appear in that clique), and Z is a normalization constant.

A *factor graph* (Kschischang, Frey, and Loeliger, 2001) explicitly represents these potentials as factor nodes in the graph structure. It is a bipartite graph of variable and factor nodes with edges connecting a variable and a factor. An edge indicates that the variable is in the set X_i of the connected potential f_i. Any Bayesian network can be converted into a factor graph by replacing every conditional probability $\Pr(x_i|\,\text{parents}(x_i))$ by a factor $f(\{x_i\} \cup \text{parents}(x_i))$.

Example 2.8. Figure 2.4 depicts a factor graph that represents the exact same distribution as the Bayesian network of Figure 2.3. Each conditional probability table is represented by one of the factors. The joint probability distribution encoded by this network is identical to the one in Table 2.1. For this undirected model, the partition function Z is 1. Note that this is not the case in general, but a remnant from translating conditional probabilities into factors. For example, multiplying one of the factor tables by a constant still constitutes a valid factor graph, but causes Z to not equal 1.

Factor graphs are often represented as *log-linear models*, where potentials are replaced by an exponentiated weighted sum of features $\psi_i(X_i)$ of the state:

$$\Pr(\omega) = \frac{1}{Z} \exp\left(\sum_i w_i \psi_i(\omega_i)\right). \tag{2.3}$$

A feature may be any real-valued function of the state.

The same inference tasks that exist for Bayesian networks also exist for undirected models. Inference algorithms are often conveniently formulated on factor graphs, as we will see in the next section.

2.4 Exact Probabilistic Inference

From an automated reasoning perspective, probabilistic graphical models are interesting because they support efficient inference algorithms. For exact inference, notable examples are *variable elimination* (Zhang and Poole, 1994; Dechter, 1996), the *junction tree* algorithm (Pearl, 1988), *recursive conditioning* (Darwiche, 2001c) and *weighted model counting* (Sang, Beame, and Kautz, 2005; Chavira and Darwiche, 2008), which is the subject of this section.

2.4.1 Probabilistic Inference by Weighted Model Counting

Weighted model counting (WMC) is a logical inference task with close connections to probabilistic inference. It takes as input a sentence ϕ in propositional logic and a function weight : $L \to \mathbb{R}_{\geq 0}$ that associates a non-negative weight with every literal (set L) in the sentence. The weight of an interpretation ω (set of literals assigning a truth value to each variable) is $\prod_{l \in \omega} \text{weight}(l)$. These interpretation weights are aggregated into the weighted model count of a sentence as follows.

Definition 2.1 (WMC)**.** Let $\mathcal{MOD}_\phi$ be the set of models of ϕ. The weighted model count is then defined as

$$\text{WMC}(\phi, \text{weight}) = \sum_{\omega \in \mathcal{MOD}_\phi} \prod_{l \in \omega} \text{weight}(l). \tag{2.4}$$

Example 2.9. For the sentence $\text{sun} \wedge \text{rain} \Rightarrow \text{rainbow}$ and weight function $\text{weight}(\text{rain}) = 2$, $\text{weight}(\neg\,\text{rain}) = 7$, $\text{weight}(\text{sun}) = 1$, $\text{weight}(\neg\,\text{sun}) = 5$, $\text{weight}(\text{rainbow}) = 0.1$ and $\text{weight}(\neg\,\text{rainbow}) = 10$, Table 2.2 lists the weight of each interpretation. Note that the second interpretation does not satisfy the sentence and is therefore not included in the weighted model count, which totals 525.4.

rain	sun	rainbow	Weight	
T	T	T	$2 \cdot 1 \cdot 0.1$	0.2
T	T	F	~~$2 \cdot 1 \cdot 10$~~	~~20~~
T	F	T	$2 \cdot 5 \cdot 0.1$	1
T	F	F	$2 \cdot 5 \cdot 10$	100
F	T	T	$7 \cdot 1 \cdot 0.1$	0.7
F	T	F	$7 \cdot 1 \cdot 10$	70
F	F	T	$7 \cdot 5 \cdot 0.1$	3.5
F	F	F	$7 \cdot 5 \cdot 10$	350
			$+$	525.4

Table 2.2: Weighted Interpretations and Model Count

Computing the Partition Function by Weighted Model Counting

We will now show that we can *reduce* the problem of computing the partition function Z of a factor graph (Equation 2.2) to a weighted model counting problem (Sang, Beame, and Kautz, 2005; Chavira and Darwiche, 2008). The reduction creates a propositional theory with two sets of logical variables. The first set is simply the set of random variables $x_1, \ldots, x_n$ in the factor graph. The second set consists of variables $\theta_1, \ldots, \theta_k$, denoting the parameters in the potential tables. Each formula in the generated theory corresponds to one or more rows from a potential table. For example, the row $x_1 = \mathrm{T}, \ldots, x_m = \mathrm{F}$ with parameter w_i is represented by the formula $\theta_i \Leftrightarrow x_1 \wedge \cdots \wedge \neg x_m$. It encodes that this configuration of the random variables increases or decreases the probability of possible worlds by a factor w_i. The associated weight function assigns these w_i parameters to the positive θ_i literals and a weight of 1 to all other literals.

The weight of a possible world of the factor graph is the product of certain parameters. These parameters have a one-to-one correspondence to the positive θ literals in some model of the logical theory. This means that the weight of a possible world equals the weight of a model of the logical theory. Furthermore, there is a one-to-one mapping between models and possible worlds of the factor graph. Therefore, the weighted model count of this theory is equal to the partition function of the factor graph.

We can further optimize the reduction to a weighted model counting problem by exploiting various types of *local structure*, such as determinism, parameter equality and context-specific independencies (Boutilier et al., 1996; Chavira and Darwiche, 2005).

- *Determinism* means that some potentials are 0, so that certain possible worlds have zero probability. We can exclude those interpretations from the model count using hard logical constraints.
- *Parameter equality* means that multiple rows in a table have the same potential value. We only need a single θ variable to represent all those rows, by adding a formula $\theta \Leftrightarrow \phi$ such that the models of ϕ correspond to the rows.
- *Context-specific independencies* arise when the truth values of the θ-variables belonging to a factor become independent of certain variables in the factor, given an assignment to other variables. This can occur when the formula ϕ above represents a large number of rows in the potential table but has a concise logical representation.

Example 2.10. Consider the factor graph of Figure 2.4. Its weighted model counting formulation is obtained by converting each factor table individually. The table for factor f_3 is represented by the theory

$$\theta_1 \Leftrightarrow \text{rain} \wedge \text{sun} \wedge \text{rainbow}$$

$$\theta_2 \Leftrightarrow \text{rain} \wedge \text{sun} \wedge \neg\, \text{rainbow}$$

$$\theta_3 \Leftrightarrow (\text{rain} \wedge \neg\, \text{sun} \wedge \text{rainbow}) \vee (\neg\, \text{rain} \wedge \text{sun} \wedge \text{rainbow})$$

$$\theta_4 \Leftrightarrow (\text{rain} \wedge \neg\, \text{sun} \wedge \neg\, \text{rainbow}) \vee (\neg\, \text{rain} \wedge \text{sun} \wedge \neg\, \text{rainbow})$$

$$\neg(\neg\, \text{rain} \wedge \neg\, \text{sun} \wedge \text{rainbow})$$

and a weight function with $\text{weight}(\theta_1) = 0.9$, $\text{weight}(\theta_2) = 0.1$, $\text{weight}(\theta_3) = 0.05$, $\text{weight}(\theta_4) = 0.95$, and all other literal weights equal to 1. The formulas for θ_3 and θ_4 are examples of local structure being exploited. The last formula encodes a deterministic dependency. The last row of the potential table is not represented in the theory because it has potential value 1 and therefore does not change the probability of a possible world.

Computing Queries by Weighted Model Counting

Given a probabilistic model M, the main inference task is now to compute the *marginal probability* of some variable x_j by summing out the remaining x

variables from the probability distribution. From Equation 2.2, this is

$$\Pr(x_j) = \sum_{\omega \models x_j} \Pr(\omega) = \frac{1}{Z} \sum_{\omega \models x_j} \prod_i f_i(\omega_i).$$

We already know how to compute the partition function denominator of this equation using weighted model counting. The numerator corresponds precisely to the partition function of a modified factor graph, where possible worlds that satisfy $\neg x_j$ get zero probability. Assume that the weighted model counting reduction of M consists of the sentence ϕ with its weight function. We can then compute marginal probabilities as $\Pr(x_j) = \text{WMC}(\phi \wedge x_j, \text{weight}) / \text{WMC}(\phi, \text{weight})$.

This can be generalized to the computation of the *conditional probability* of some query q given the evidence e. Intuitively, calculating $\Pr(q|e)$ requires computing the weight of the possible worlds where e and q are both true and dividing it by the weight of the possible worlds where e is true. Formally, this can be written as follows:

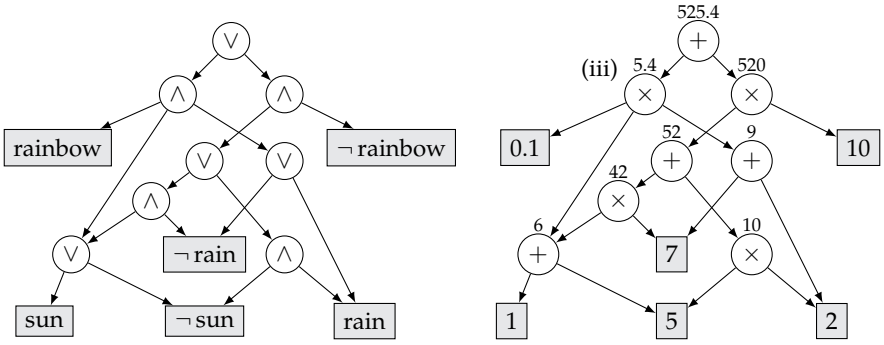
$$\Pr(q|e) = \frac{\Pr(q \wedge e)}{\Pr(e)} = \frac{\text{WMC}(\phi \wedge q \wedge e, \text{weight})}{\text{WMC}(\phi \wedge e, \text{weight})}. \tag{2.5}$$

There exist many algorithms to solve the weighted model counting task, based on DPLL search (Sang, Beame, and Kautz, 2005), local search (Wei and Selman, 2005), sampling (Gogate and Dechter, 2011) and knowledge compilation (Chavira and Darwiche, 2008). The next section describes this knowledge compilation approach in more detail.

2.4.2 Probabilistic Inference by Knowledge Compilation

We will first show how to perform weighted model counting by knowledge compilation and then discuss its application to probabilistic inference.

Computing the weighted model count of an `sd-DNNF` circuit is similar to the computation of the model count outlined in Section 2.2.1. It involves transforming the `sd-DNNF` circuit into an arithmetic circuit by replacing conjunctions by multiplications and disjunctions by additions. The difference is in the transformation of literals, which for the weighted model counting task are replaced by their weights according to the weight function.



(a) sd-DNNF circuit (b) Weighted model counting circuit

Figure 2.5: Circuits for the sentence sun $\wedge$ rain $\Rightarrow$ rainbow

Example 2.11. Figure 2.5a again shows an sd-DNNF circuit for the sentence sun $\wedge$ rain $\Rightarrow$ rainbow. The weighted model count of this sentence for the weight function of Example 2.9 is computed by the arithmetic circuit in Figure 2.5b. It confirms the result of Table 2.2, that the weighted model count is 525.4.

Now that we can efficiently perform weighted model counting, we can compute the partition function Z using the reduction of Section 2.4.1 and compute marginal and conditional probabilities using Equation 2.5.

Example 2.12. Consider the probability distribution represented by the weighted model counting problem with sentence $\phi = (\text{sun} \wedge \text{rain} \Rightarrow \text{rainbow})$ and the weight function of Example 2.9. The query $\Pr(\text{rainbow})$ can be computed by the ratio $\text{WMC}(\phi \wedge \text{rainbow}, \text{weight}) / \text{WMC}(\phi, \text{weight})$. The denominator is computed by the circuit in Figure 2.5b to be 525.4. The numerator is computed to be 5.4 by the node labeled (iii) of the same circuit. Therefore, we have that $\Pr(\text{rainbow}) = 5.4/525.4 \approx 0.0103$.

The motivation for using a knowledge compilation approach is circuit reuse: by compiling a circuit once, we can answer a large number of queries efficiently. This also applies to probabilistic inference, as we can see from rewriting

Equation 2.5 as follows.

$$\begin{aligned}\Pr(q|e) &= \frac{\text{WMC}(\phi \wedge q \wedge e, \text{weight})}{\text{WMC}(\phi \wedge e, \text{weight})} \\ &= \frac{\text{WMC}((\phi|(q \wedge e)), \text{weight}) \cdot \prod_{l \in q} \text{weight}(l) \cdot \prod_{l \in e} \text{weight}(l)}{\text{WMC}((\phi|e), \text{weight}) \cdot \prod_{l \in e} \text{weight}(l)} \\ &= \frac{\text{WMC}((\phi|(q \wedge e)), \text{weight}) \cdot \prod_{l \in q} \text{weight}(l)}{\text{WMC}((\phi|e), \text{weight})}\end{aligned}$$

Note that now the '|' symbol denotes logical conditioning, as explained in Section 2.2.2. The equation shows that to perform probabilistic inference, it suffices to compile a *single circuit*, for ϕ (the partition function computation), and reuse it to compute the conditional probability of any query term (i.e., conjunction of literals) given any evidence term. With additional optimizations, this single circuit allows computing the probability of evidence $\Pr(e)$ and *all posterior marginals* $\Pr(x_i|e)$ in only two passes over the circuit, and time and memory linear in its size (Darwiche, 2003).

2.5 Approximate Probabilistic Inference

Exact inference algorithms for general probabilistic graphical models have a upper complexity bound that is exponential in the *treewidth* (Robertson and Seymour, 1983; Robertson and Seymour, 1986) of the underlying graph structure. Treewidth is a measure of the connectivity of the network, indicating how tree-like the network is. Unfortunately, this often makes exact inference intractable. The knowledge compilation approach outlined in the previous section has the same complexity: compiling a logic theory into `sd-DNNF` is exponential in the treewidth of the logical theory (Darwiche, 2001b).

To overcome this problem, approximate inference algorithms are used in practice. Notable examples are *loopy* or *iterative belief propagation* (Pearl, 1988), *variational* (Jordan et al., 1999), and *sampling* methods. We will briefly present the intuition behind two related approximate inference algorithms. The first is belief propagation – a seminal algorithm. The second is a state-of-the-art generalization of belief propagation, called *relax, compensate and recover*. What makes this algorithm so appealing is that it allows us to use the algorithms for

Intermezzo 1: Algebraic Model Counting

Knowledge compilation is used to solve many more tasks, other than computing probabilities and weighted model counts. For example, replacing summation by maximization in a model counting circuit computes the probability of the MPE state. Kimmig, Van den Broeck, and De Raedt (2012a) explore this observation and introduce a new inference task called *algebraic model counting* (AMC), which generalizes model counting to a semiring structure.

Definition 2.2. A *commutative semiring* is a structure $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, where *addition* $\oplus$ and *multiplication* $\otimes$ are associative and commutative binary operations over the set $\mathcal{A}$, $\otimes$ distributes over $\oplus$, $e^{\oplus} \in \mathcal{A}$ is the neutral element of $\oplus$, $e^{\otimes} \in \mathcal{A}$ that of $\otimes$, and for all $a \in \mathcal{A}$, $e^{\oplus} \otimes a = a \otimes e^{\oplus} = e^{\oplus}$.

Definition 2.3. Given (i) a *propositional logic theory* ϕ over a set of variables V, (ii) a *commutative semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, and (iii) a *labeling function* $\alpha : L \to \mathcal{A}$, mapping literals L of the variables in V to elements of the semiring set $\mathcal{A}$, *algebraic model counting* computes $\mathrm{AMC}(\phi) = \bigoplus_{\omega \in \mathcal{MOD}_\phi} \bigotimes_{l \in \omega} \alpha(l)$.

AMC supports various types of labels, including numbers, sets, tuples, polynomials and formulae. It generalizes many different tasks from a variety of different fields. For example, deciding consistency of ϕ is captured by the semiring $(\{\mathrm{T}, \mathrm{F}\}, \vee, \wedge, \mathrm{F}, \mathrm{T})$ with $\alpha(v) = \alpha(\neg v) = \mathrm{T}$. Counting its models corresponds to the semiring $(\mathbb{N}, +, \times, 0, 1)$ with $\alpha(x) = \alpha(\neg x) = 1$ and the WMC task (Equation 2.4) to the semiring $(\mathbb{R}_{\geq 0}, +, \times, 0, 1)$ with $\alpha(v) = \mathrm{weight}(v)$. The MPE probability task is obtained by setting $\oplus$ to max in the WMC semiring. Other AMC tasks compute the MPE state, gradients and expected costs. AMC generalizes inference in probabilistic graphical models (Bacchus, Dalmao, and Pitassi, 2009) and algebraic Prolog (Eisner, Goldlust, and Smith, 2005; Kimmig, Van den Broeck, and De Raedt, 2011), sensitivity analysis, soft constraint satisfaction (Meseguer, Rossi, and Schiex, 2006), and network and database analysis (Baras and Theodorakopoulos, 2010). As AMC is defined in terms of the models of a sentence ϕ, it can be performed by knowledge compilation. Any AMC task can be evaluated on a `sd-DNNF` representation of ϕ, which is more succinct than a direct representation of the set of models. Furthermore, certain properties of AMC tasks allow for using more succinct circuit types. For example, when $\oplus$ is *idempotent* (i.e., $a \oplus a = a$), it can be evaluated on a (non-deterministic) `s-DNNF` representation. When addition is *neutral* (i.e., $\alpha(v) \oplus \alpha(\neg v) = e^{\otimes}$), non-smooth circuits can be used. Other properties even allow for evaluation on non-decomposable circuits.

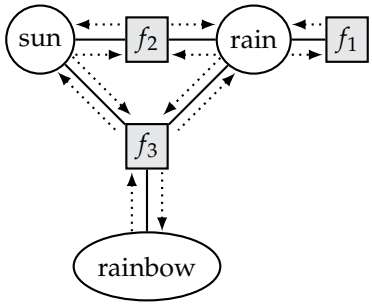


Figure 2.6: Iterative belief propagation on a factor graph. Dotted arrows represent messages with belief updates.

exact inference presented earlier to also perform approximate inference, even in graphical models with high treewidth.

2.5.1 Iterative Belief Propagation

Loopy or *iterative belief propagation* (IBP) (Pearl, 1988) is a message passing algorithm where nodes in the factor graph exchange messages along the edges of the graph. Intuitively, these messages quantify the influence that each node has on its neighbors' probability. Each node keeps an estimate of its marginal probability, called a *belief*. These beliefs are updated based on incoming messages. Messages are sent out iteratively from each node, based on its belief, until the beliefs and messages have converged.

We refer to Koller and Friedman (2009) and Darwiche (2009) for a more formal description. The details of how belief propagation messages are computed for a specific type of log-linear models are also explained in Chapter 7. Belief propagation is a special case of the algorithm presented in that chapter.

When the factor graph is a tree, belief propagation is an exact inference algorithm. Otherwise, it often finds good approximations, if the iterative algorithm converges (Murphy, Weiss, and Jordan, 1999).

Example 2.13. Figure 2.6 visualizes the messages sent by IBP on the factor graph of Figure 2.4. Because this network has a loop, the marginals estimated by the beliefs of IBP will only be approximate.

2.5.2 Relax, Compensate & Recover

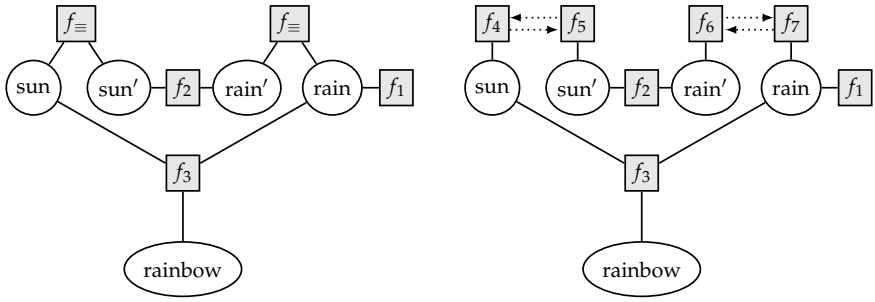
Relax, Compensate & Recover (RCR) (Choi and Darwiche, 2011) is an algorithm that performs approximate probabilistic inference by doing exact inference in a simplified, approximate model. This simplified model is changed while running the algorithm, in order to improve its approximation. RCR performs the following three steps.

1. Every edge in the factor graph connects a random variable x with a factor f. *Cloning* removes an edge from the network and adds a clone x' of x. It then connects x' to f and adds a factor $f_{\equiv}$ between x and x', encoding an equivalence constraint. Cloning does not change the modeled distribution, it only adds random variables to it. *Relaxation* removes an equivalence constraint $f_{\equiv}$ from the network, making the graph structure more sparse and reducing its treewidth. A common setup is to clone variables and relax equivalences until the relaxed model is a tree, which is amenable to exact inference.

2. To correct for the removed equivalences, *compensation* connects one new unary factor with x and one with x'. The variables x and x' were equivalent in the exact probability distribution. Intuitively speaking, compensation tries to make these variables equivalent again, for some weaker notion of equivalence. It does this by setting the added unary potentials in an appropriate way, to compensate for the ignored dependencies. Intuitively, one approach is to enforce that $\Pr(x) = \Pr(x')$. When $\Pr(x = \mathrm{T}) < \Pr(x' = \mathrm{T})$, this is compensated for in the unary potentials f_x and $f_{x'}$ by increasing the potential $f_x(\mathrm{T})$ and decreasing $f_{x'}(\mathrm{T})$. Setting these potentials is done by a message passing algorithm similar to belief propagation. It performs exact inference in the relaxed model to compute the right messages.

3. *Recovery* adds back relaxed equivalences $f_{\equiv}$ to the model in order to incrementally improve approximation quality. Each recovery step makes inference in the relaxed model harder, also making the compensation step more difficult. If exact inference in the original model is tractable, all equivalences will eventually be recovered, and RCR returns exact results.

Example 2.14. Figure 2.7 shows two steps of the RCR algorithm applied to the factor graph of Figure 2.4. Figure 2.7a depicts the cloning of two variables sun and rain into sun$'$ and rain$'$. The added factors $f_{\equiv}$ encode



(a) Cloning (b) Relaxation and Compensation

Figure 2.7: Relax and Compensate on a Factor Graph

that clones are equivalent to the original variables in every possible world. Hence, Figure 2.7a models the same distribution over the original variables as Figures 2.4 and 2.3. Figure 2.7b shows the relaxation step ($f_{\equiv}$ factors are removed) and compensation step (factors f_4 to f_7 are added). The relaxation step turns the loopy graph into two trees, in which exact inference is more efficient. The dotted lines represent messages being sent by the iterative compensation algorithm, to synchronize the beliefs of the original variables and their clones.

Knowledge compilation inference provides additional advantages in the context of RCR. The messages sent by the compensation algorithm are a function of all marginal probabilities in the relaxed network. As discussed in Section 2.4.2, we can use a single compiled `sd-DNNF` circuit for this task. Furthermore, in different iterations of the compensation algorithm, the structure of the factor graph does not change, only the parameters of the compensating factors. In weighted model counting terms, this means that only the weight function changes, not the sentence for which we compute the weighted model count. Hence, we can reuse a single compiled `sd-DNNF` circuit in each iteration of the compensation algorithm, until an equivalence is recovered and the structure of the relaxed model changes.

The RCR framework leads to a spectrum of approximations. When all equivalences are relaxed, the compensation phase sends and receives the same messages as IBP and obtains the same approximate marginals (Choi and Darwiche, 2006). When recovering more edges, the approximations obtained will correspond to *generalized belief propagation* approximations (Yedidia, Freeman, and Weiss, 2003), in particular *iterative joingraph propagation* (Dechter,

Kask, and Mateescu, 2002). We will discuss more connections to related work in Chapter 7.

2.6 Learning Probabilistic Graphical Models

There are many different settings for learning a probabilistic graphical model. These are distinguished by the following properties. When the graph structure is given and we want to learn the parameters in the potential or conditional probability tables, the task is called *parameter learning*. If the graph structure is also unknown, it is called *structure learning*. One distinguishes between learning from *complete data*, when the training examples assign a value to every variable, and learning from *incomplete data*, when certain values are missing. Finally, there can be different criteria for evaluating learned models. In a *generative* learning setting, the performance of a learned model is evaluated based on how it explains the data as a whole. In *discriminative* learning, the goal is to best explain certain variables in the data, given an assignment to the other variables. The *maximum likelihood* setting prefers the model that maximizes the probability of the data, whereas *Bayesian learning* incorporates prior knowledge about preferred parameters and structures into the evaluation.

There are too many combinations of settings to cover here. We will focus on one task: *generative maximum-likelihood parameter learning from complete data* and refer to Koller and Friedman (2009) and Darwiche (2009) for the other tasks. This setting is arguably the most fundamental one, as it is a building block in many algorithms for the other tasks. For example, structure learning algorithms typically search for appropriate graph structures by evaluating their quality with a parameter learning algorithm (Cooper and Herskovits, 1992; Della Pietra, Della Pietra, and Lafferty, 1997). The expectation maximization algorithm (Dempster, Laird, and Rubin, 1977) allows us to learn from incomplete data by iteratively learning from complete data.

Generative Maximum-Likelihood Parameter Learning

A natural objective function for our learning task is to learn maximum-likelihood weights, that is, to maximize the (log-)probability of the training

data (Equation 2.2).

$$\log \Pr(\Omega) = \sum_{i=1}^{N} \log \Pr(\Omega_i), \tag{2.6}$$

where N is the number of examples, Ω_i is the state of the ith example and $\Pr(\Omega_i)$ is given by Equation 2.1 for Bayesian networks or Equation 2.2 for undirected models.

For Bayesian networks, optimizing Equation 2.6 can be done in *closed form*. The most likely parameter representing $\Pr(x_i = v | \text{parents}(x_i) = w)$ is simply the ratio of how many times $x_i = v$ and $\text{parents}(x_i) = w$ appear in the data.

For undirected networks, in general, optimizing Equation 2.6 cannot be done in closed form. Instead, weight learning is addressed via *convex optimization*. Each iteration of the optimization involves running inference over the current model to compute the likelihood and its gradient. However, even just evaluating the probability of the data is (often) intractable, since it involves computing the partition function Z.

Consequently, people often optimize an approximate objective function such as *pseudolikelihood* (Besag, 1975; Koller and Friedman, 2009), which is currently the default generative weight learning approach for undirected models. The pseudo-log-likelihood for the data Ω is more efficient to compute than the likelihood. It is defined as

$$\log \Pr^{\bullet}(\Omega) = \sum_{j=1}^{V} \sum_{i=1}^{N} \log \Pr(x_j = \Omega_{i,j} | MB_{\Omega_i}(x_j)), \tag{2.7}$$

where V is the number of variables, $\Omega_{i,j}$ is the value of the jth variable of the ith example, $MB_{\Omega_i}(x_j)$ is the state of x_j's Markov blanket (i.e., the neighbors of node x_j in the undirected graph) in the ith example. This objective can also be optimized via convex optimization.

2.7 Conclusions

We have reviewed the literature on propositional logical and probabilistic graphical models. These form the representational foundations of the work presented in the following chapters, on first-order logic and statistical relational

models. For both logical and probabilistic models, we have discussed inference tasks and inference algorithms, focusing on knowledge compilation for exact inference and relax, compensate and recover for approximate inference. Finally, we looked at the task of learning probabilistic graphical models and presented a generative weight learning algorithm. These will form the basis of the lifted inference and learning algorithms presented in this dissertation.

3

First-Order Circuits

First-order logic (FOL), also called predicate logic, is a language used for knowledge representation and automated reasoning. It is different from propositional logic in its use of logical variables and quantifiers. With these, it is possible to make general statements about multiple objects in the world. It causes first-order logic to be more expressive than propositional logic.

This chapter introduces a new problem in the area of knowledge compilation: *first-order knowledge compilation* to *negation normal form* (NNF). Knowledge compilation is a technique that compiles theories in propositional logic into circuits. These circuits can then be used to efficiently answer certain types of queries, such as deciding satisfiability of the theory. For example, compilation to a d-DNNF circuit allows us to count the number of models of a theory efficiently. What we present here is a first-order generalization of knowledge compilation. We lift the NNF circuit language to the first-order setting by defining the declarative syntax and semantics of these circuits. The contributions of this chapter are in the field of automated *logical* reasoning. These circuits will form a basic building block for the algorithms for automated probabilistic reasoning developed in later chapters.

First-order logic is a very general formalism, not perfectly suitable for our purposes. Therefore, in **Section 3.1**, we define the variation on first-order logic

that we will use, called first-order logic with domain constraints. For this logic we then define a first-order NNF circuit language in **Section 3.2**. In **Section 3.3**, we define subsets of first-order NNF circuits that permit tractable inference for certain types of hard queries. The properties of these subsets are described in **Section 3.4**. Finally, **Section 3.5** discusses related work, after which we conclude.

An early version of first-order negation normal form was presented as

> G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011a). "Lifted probabilistic inference by first-order knowledge compilation". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. Menlo Park, California, pp. 2178–2185

The explanation in this chapter is very different. It is a more general, declarative treatment of the subject, with many additional unpublished contributions.[1] Our discussion of the conditioning transformation on NNF circuits was published as

> G. Van den Broeck and J. Davis (2012). "Conditioning in first-order knowledge compilation and lifted probabilistic inference". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence,* Palo Alto, California, USA

3.1 First-Order Logic with Domain Constraints

In order to define our first-order NNF language, we want more explicit control over the set of constants that quantifiers range over in FOL. To this end, we introduce a variant of first-order logic called *first-order logic with domain constraints* (FOL-DC). We first motivate the need for FOL-DC, after which we define its syntax and semantics. We review standard syntax and semantics of (function-free) FOL where needed. For a more thorough treatment of FOL, see Brachman and Levesque (2004).

[1] These are in part the product of discussions with Adnan Darwiche and Luc De Raedt.

3.1.1 Motivation

Classical FOL has a domain of discourse that can be infinite, meaning that the quantifiers $\forall$ and $\exists$ range over an infinite set of objects. FOL-DC allows us to elegantly restrict quantifiers to a finite domain. The motivation for using finite domains and introducing FOL-DC is as follows.

First, having a finite domain of discourse opens up interesting new *inference tasks*, such as (weighted) model counting, which is used extensively in this thesis (see Chapter 5). Some of the popular probabilistic logical languages have the same restriction (e.g., Friedman et al., 1999; Richardson and Domingos, 2006) to obtain a well-defined probability distribution over ground atoms. Other inference tasks are already well defined for FOL, but are easier in FOL-DC. For example, checking validity or satisfiability of a sentence is undecidable in full FOL but becomes decidable in FOL-DC.[2]

Second, it can be desirable to interpret first-order logic theories in terms of the propositional theories they represent, when all formulas are grounded. Such a corresponding propositional theory always exists, but is only finite when the quantifiers range over finite domains. It is sometimes convenient to think of FOL-DC as a rich and concise template language for finite propositional theories.

Third, by keeping a close connection to propositional logic, it is possible to capitalize on the algorithmic advances that happened in the last decades in the *automated propositional reasoning* world. For example, SAT solving is a great success story of computer science (Järvisalo et al., 2012). Similar progress has been made for the model counting problem (Gomes, Sabharwal, and Selman, 2009). Also in the probabilistic reasoning world, many systems reduce first-order problems to propositional problems for answering queries (e.g., Friedman et al., 1999; Kersting and De Raedt, 2001b; Richardson and Domingos, 2006; Fierens et al., 2012b). This is known as *propositionalization* or *knowledge-based model construction* (Wellman, Breese, and Goldman, 1992) and is discussed in more detail in Section 5.2.

These are motivations for using finite domains. However, as will become clear in what follows, FOL-DC is *more expressive*. It can represent domains with logical variables that refer to sets and (in)equality constraints between constants. The motivation for having this level of expressivity is the need to reason about

[2]This is a consequence of using finite Herbrand logic (see Section 3.1.3). It is not a unique property of FOL-DC or finite Herbrand logic. Many subclasses of FOL were shown to be decidable. See Börger, Grädel, and Gurevich (2001) for an overview.

abstract, unknown domains, which can have arbitrary size, and compile logical circuits that apply to any finite domain of discourse. This way, we will be able to efficiently reason about different domains of discourse, involving multiple distinct subdomains, and reuse compiled NNF circuits for answering multiple queries.

3.1.2 Syntax

The syntax of FOL-DC states which expressions are well-formed and correctly structured. This section describes the syntax, consisting of an alphabet and a grammar.

Alphabet

The *alphabet* of FOL-DC contains the alphabet of (function-free) FOL:

- Logical symbols
  - Logical connectives $\wedge$, $\vee$, $\neg$, etc.
  - Quantifiers $\forall$ and $\exists$
  - Equality symbol $=$
  - True T and false F
  - A set of *logical variables* $\mathbf{X} = \{X, Y, \dots\}$
  - (Parentheses and punctuation)
- Non-logical symbols (or signature)
  - A set of predicate symbols p/n of arity n, including propositional variables, which have arity 0.
  - A set of constant symbols $\{a, b, c, \dots\}$

We will assume that there is a natural order on the set of constants in the alphabet. The FOL-DC alphabet further extends this with:

- Logical symbols
  - Less-than symbol $<$

  - Set membership symbol $\in$ and set inclusion symbol $\subseteq$
  - A set of *domain variables* $\mathbf{D} = \{\mathsf{D}, \mathsf{F}, \dots\}$.
  - Union symbol $\cup$, intersection symbol $\cap$ and set difference symbol $\setminus$
- Non-logical symbols (or signature)
  - A set of "*set of constants*" symbols $\{\{a, b, c\}, \{a, d\}, \emptyset, \dots\}$

Grammar

A *logical term* is either a logical variable or a constant. A *domain term* is either a domain variable, a set of constants or a combination of two other domain terms t_d and t'_d in the form $t_d \cup t'_d$, $t_d \cap t'_d$ or $t_d \setminus t'_d$. Similar to how logical terms refer to objects in the domain of discourse, domain terms refer to sets of these objects.

We similarly distinguish between two types of atomic formulas in FOL-DC: logical and domain atoms.

Definition 3.1 (Logical Atom)**.** A *logical atom* $\mathrm{p}(t_1, \dots, t_n)$ is constructed by applying a predicate p/n (which excludes $=$, $<$, $\in$ and $\subseteq$) to a n-tuple of logical term arguments t_i (i.e, logical constants or logical variables from $\mathbf{X}$).

Definition 3.2 (Domain Atom)**.** *Domain atoms* are of the form

- $t_l = t'_l$ or $t_l < t'_l$ between logical terms t_l and t'_l, or
- $t_d = t'_d$, $t_d \subseteq t'_d$ between domain terms t_d and t'_d, or
- $t_l \in t_d$ between a logical term t_l and a domain term t_d.

Example 3.1. Examples of domain atoms are $X = Y$, $X = a$, $a = b$, $X < Y$, $X < a$, $a < b$, $\mathsf{D} = \mathsf{F}$, $\mathsf{D} = \{a, b, \dots\}$, $\{a, b, \dots\} = \{a, d, \dots\}$, $\mathsf{D} \subseteq \mathsf{F}$, $\mathsf{D} \subseteq \{a, b, \dots\}$, $\{a, b, \dots\} \subseteq \mathsf{D}$ and $\{a, b, \dots\} \subseteq \{c, d, \dots\}$.

Definition 3.3 (Domain Constraint)**.** A *domain constraint* is a domain atom or its negation.[3]

[3]The negation of $=, \in, \subseteq$ is denoted by the symbols $\neq, \notin, \not\subseteq$.

Definition 3.4 (Constraint Set)**.** A *constraint set* is a conjunction of domain constraints.[4]

Example 3.2. The constraint set $X \in \mathsf{Bird} \wedge X \neq \mathit{kiwi}$ constrains the logical variable X to be some bird (member of the domain variable Bird), but not a kiwi (the constant *kiwi*).

Definition 3.5 (Well-Formed Formula)**.** Well-formed *formulas* are inductively defined as follows:

- A logical atom is a formula.
- If ϕ, ψ are formulas, then the negation $\neg\phi$, *extensional conjunction* $\phi \wedge \psi$ and *extensional disjunction* $\phi \vee \psi$ are formulas.
- If ϕ is a formula, $\mathbf{V}$ a (potentially empty) set of (logical or domain) variables and *cs* a constraint set that contains at least one domain atom of the form $V = t$, $V \in t$ or $V \subseteq t$ for for every $V \in \mathbf{V}$, then the following are formulas:
  - *intensional conjunction* $\forall \mathbf{V}, \mathit{cs} : \phi$, and
  - *intensional disjunction* $\exists \mathbf{V}, \mathit{cs} : \phi$

An intensional operator $((\forall, \exists)\mathbf{V}, \mathit{cs} : \phi)$ *quantifies* the variables in $\mathbf{V}$. Note that in the definition of a formula, domain terms and the symbols $=$, $<$, $\in$ and $\subset$ may only occur in the constraint set of an intensional operator. Logical atoms cannot appear in the constraint set of an intensional operator. Having this clear separation in the syntax between atoms that define a finite domain and classical logical atoms is one of the motivations for using FOL-DC.

The *syntax* of classical (function-free) FOL is similar to the syntax of FOL-DC, but disallows domain constraints to be associated with its quantifiers. Although we extended some notions from FOL for our purposes, any legal expression in classical FOL can easily be transformed into an equivalent legal expression in FOL-DC. For a given domain of discourse $\mathcal{D}$, the FOL formula $\forall X \phi$ maps to the FOL-DC formula $\forall X, X \in \mathcal{D} : \phi$. Similarly, existential quantification is mapped to intensional disjunction. Other formulas in FOL are already legal expressions in FOL-DC.

[4] For the sake of simplicity, we limit the constraint language to conjunctions of domain constraints. Most of the notions about FOL-DC can also be defined in terms of more expressive constraint languages. The purpose of a more expressive constraint language could be that certain constraint sets could be expressed much more concisely.

Syntactic Terminology

A *literal* l is a logical atom a or its negation $\neg a$. A *clause* is a disjunction of literals and a theory in *conjunctive normal form* (CNF) is a conjunction of clauses. A *term* is a conjunction of literals and a DNF is a disjunction of terms. A *theory* or *knowledge base* is a conjunction of formulas. A variable is *bound* if it is quantified by an enclosing intensional disjunction or conjunction. A *free variable* is a variable that is not bound. A formula without free variables is called a *sentence*. A formula is *ground* if it does not contain any variables.

Definition 3.6 (Substitution)**.** A *substitution* $[X/t_l, \ldots, \mathsf{D}/t_d]$ is a mapping from logical variables to logical terms and from domain variables to domain terms.

Applying a substitution is a syntactic transformation that replaces logical variables by logical terms and domain variables by domain terms. The result of applying a substitution θ to a formula ϕ is denoted by $\phi\theta$.

Definition 3.7 (Closing Substitution)**.** A substitution θ is a *closing substitution* for formula ϕ iff $\phi\theta$ is a sentence.

Example 3.3. The FOL-DC formula

$$\forall X, X \in \mathsf{People} : \mathrm{belgian}(X) \Rightarrow \mathrm{likes}(X, \mathit{chocolate}) \tag{3.1}$$

contains one logical variable (X), one domain variable (People), one constant (*chocolate*), two atoms (belgian(X) and likes$(X, \mathit{chocolate})$) and two predicates (belgian /1 and likes /2). The variable X is bound by the quantifier $\forall X, X \in$ People of the intensional conjunction. Both atoms are non-ground. Applying the substitution $[X/\mathit{tintin}]$ to belgian(X) results in belgian(tintin), which is a ground atom.

Example 3.4. Consider the following formula ϕ in FOL-DC:

$$\forall X, X \in \mathsf{Bird} \wedge X \neq \mathit{kiwi} : \mathrm{flies}(X). \tag{3.2}$$

The domain variable Bird in ϕ is a free variable. Therefore, the formula is not a sentence. Applying the closing substitution $\theta = [\mathsf{Bird}/\{\mathit{pigeon}, \mathit{kiwi}, \mathit{eagle}\}]$ creates the sentence $\phi\theta$, where all variables are bound.

3.1.3 Semantics

The *semantics* of classical FOL assigns a truth value to each sentence, given a *domain of discourse* and an interpretation of the constants and predicates in the alphabet (Brachman and Levesque, 2004). It gives meaning to each sentence in FOL. The domain of discourse is the set of objects that quantifiers range over. Constants and logical variables refer to these objects. In this dissertation, we always assume that the domain of discourse is itself a set of constants and that each constant refers to itself. This logic is called Finite Herbrand Logic (Hinrichs and Genesereth, 2006). A Herbrand *interpretation* is a set of ground atoms where 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 Δ, written as $I \models \Delta$, if it satisfies all the formulas in Δ. Satisfaction of a formula is defined in the usual way (Brachman and Levesque, 2004). An interpretation that satisfies the theory is a *model* for the theory.

Example 3.5. Consider the FOL sentence $\forall X$ belgian$(X) \Rightarrow$ likes$(X, \mathit{chocolate})$. Informally speaking, its intended meaning is that all Belgians like chocolate. Assuming a domain of discourse consisting of the 3 constants *chocolate*, *tintin* and *rubens*, the sentence has a Herbrand base of size 12 (3 atoms for belgian /1 and 9 for likes /2). Therefore, there are 2^{12} interpretations of this sentence, only some of which are models. An example of a model is $\{\text{belgian}(\mathit{tintin}), \text{likes}(\mathit{tintin}, \mathit{chocolate}), \text{likes}(\mathit{rubens}, \mathit{tintin})\}$, but also the empty set $\emptyset$ or the Herbrand base. An example of an interpretation that is not a model is $\{\text{belgian}(\mathit{tintin}), \text{likes}(\mathit{rubens}, \mathit{tintin})\}$.

This section presents two alternative semantics for FOL-DC, one by reduction to first-order logic and one by reduction to propositional logic, and a semantics for constraint sets.

First-Order Reduction Semantics

The semantics of FOL-DC are formally defined by first *reducing* sentences in FOL-DC to sentences in classical FOL with equality and set theory axioms. The set theory axioms give an interpretation to the set-theoretic symbols $\cup$, $\cap$, $\setminus$, $\in$ and $\subseteq$.[5] Furthermore, the natural order on the set of constants in the alphabet

[5]The best-known set-theory axioms are the *ZFC axioms* (Kunen, 1980), for Zermelo-Fraenkel set theory with the axiom of Choice. There is a problem with using ZFC axioms here: they

gives an interpretation to the less-than symbol $<$. After this reduction we can reuse the semantics of FOL, which are well known.

The reduction is performed by a syntactic rewrite to remove those logical symbols that are not in the alphabet of FOL, such that the result is a legal expression in FOL. Intensional conjunctions and disjunctions in FOL-DC are rewritten as follows:

$$\forall \mathbf{V}, cs : \phi \quad \rightarrow \quad \forall\, \mathbf{V} : cs \Rightarrow \phi \tag{3.3}$$

$$\exists \mathbf{V}, cs : \phi \quad \rightarrow \quad \exists\, \mathbf{V} : cs \wedge \phi \tag{3.4}$$

Other types of formulas are already syntactically well-formed formulas in FOL.

Example 3.6. Formula 3.2 from Example 3.4 reduces to the following FOL theory:

$$\forall\, X : X \in \mathsf{Bird} \wedge X \neq \mathit{kiwi} \Rightarrow \mathrm{flies}(X). \tag{3.5}$$

Interpretations of the FOL theory are mapped back to interpretations of the FOL-DC theory by removing domain atoms (retaining only logical atoms).

This defines the semantics of sentences, and allows us to talk about their equivalence.

Definition 3.8 (Sentence Equivalence)**.** Sentences α and β are *equivalent* ($\equiv$) when they have the same truth value in every interpretation.

The semantics of formulas are defined w.r.t. a variable-assignment, mapping free variables to objects from the domain of discourse. We will adopt the following notion of equivalence for formulas.

Definition 3.9 (Formula Equivalence)**.** Formulas α and β are equivalent ($\equiv$) when $\alpha\theta \equiv \beta\theta$ for any closing substitution θ.

Constraint Set Semantics

Intuitively, a constraint set has a semantics of its own: it represents a constraint satisfaction problem. A substitution $\theta = [V_1/k_1, \ldots, V_n/k_n]$ is a solution to a

assume that all objects in the domain of discourse are sets. This is clearly not the case for logical constants. We require an axiomatization that supports so-called *urelements* (elements of sets which are not themselves sets). The New Foundations (NF) axioms with urelements (NFU) (Jensen, 1968; Mendelson, 1997) provides such an axiomatization. It is too extensive to show here.

constraint set cs if and only if all domain constraints in $cs\theta$ are ground and satisfied. When a solution to a constraint set exists, we say that the constraint set is satisfiable.

Definition 3.10 (solutions)**.** The set of assignments (substitutions) to variables $\mathbf{V}$ for which the constraint set cs is satisfiable is denoted by solutions$(cs, \mathbf{V})$.

Note that the constraint set can contain variables other than $\mathbf{V}$.

Example 3.7. The constraint set $cs = (X \in \{pigeon, kiwi, eagle\} \wedge X \neq kiwi)$ from Example 3.4 represents a constraint satisfaction problem that has two solutions. The first is $\theta = [X/pigeon]$, because all the constraints in

$$cs\,\theta = (pigeon \in \{pigeon, kiwi, eagle\} \wedge pigeon \neq kiwi) \tag{3.6}$$

are ground and satisfied. The second solution is $[X/eagle]$.

Grounding Semantics

Sentences in FOL-DC have an alternative, more intuitive meaning in terms of their *groundings*.

Definition 3.11 (Grounding)**.** The *grounding* gr of T is T, of F is F, and of a ground logical atom is the atom itself. The grounding of a complex sentence is recursively defined as follows: $\text{gr}(\neg\phi) \equiv \neg\,\text{gr}(\phi)$, $\text{gr}(\phi \wedge \psi) \equiv \text{gr}(\phi) \wedge \text{gr}(\psi)$ and $\text{gr}(\phi \vee \psi) \equiv \text{gr}(\phi) \vee \text{gr}(\psi)$. For intensional disjunction and conjunction,

$$\text{gr}\,(\forall V, cs : \phi) \equiv \text{gr}(\phi\theta_1) \wedge \cdots \wedge \text{gr}(\phi\theta_n) \tag{3.7}$$

$$\text{gr}\,(\exists V, cs : \phi) \equiv \text{gr}(\phi\theta_1) \vee \cdots \vee \text{gr}(\phi\theta_n), \tag{3.8}$$

with $\{\theta_1, \ldots, \theta_n\} = \text{solutions}(cs, V)$.

Each intensional conjunction (disjunction) represents an extensional conjunction (disjunction) over the solutions to its constraint set. The grounding of a formula with free variables is undefined (because the grounding of a non-ground atom is undefined). In a sentence, however, every logical variable is quantified by some enclosing intensional operator, all logical variables are substituted by some constant in Equations 3.7 or 3.8 and gr is well defined. The output of gr does not contain any logical variables or domain atoms and can be

regarded as a theory in propositional logic. Having this propositional grounding semantics was one of the motivations for using FOL-DC (see Section 3.1.1).

Example 3.8. The theory $\forall X, X \in \{pigeon, kiwi, eagle\} \wedge X \neq kiwi : \text{flies}(X)$ represents the ground theory $\text{flies}(pigeon) \wedge \text{flies}(eagle)$.

Other Alternatives

FOL-DC can still be regarded in several other ways. One alternative treats domain variables D as unary predicates $\mathsf{D}(X)$ in *second-order logic*. Intensional disjunction with a domain variable then corresponds to existential second-order quantification. Constraint sets can be defined as a logical sentence, without the need for set theory axioms. For example, the intersection $\mathsf{D} \cap \mathsf{F}$ can be represented by $\mathsf{D}(X) \wedge \mathsf{F}(X)$. Another first-order alternative is to view top-level domain variables (People, Bird) as types in a *typed* first-order logic.

Our grounding semantics reduces a sentence in FOL-DC to a propositional theory without any domain atoms. There is an alternative grounding semantics, which is obtained by first applying the reduction to first-order logic, and grounding w.r.t. the Herbrand universe afterwards. Now, our propositional theory does contain domain atoms. When they are removed from each model, we obtain again the same set of models that define the semantics of FOL-DC.

3.2 First-Order Negation Normal Form Circuits

Traditionally, knowledge compilation focused on propositional logic. Section 2.2 explained one popular approach, namely knowledge compilation to negation normal form circuits (Darwiche, 1999; Darwiche, 2001a), which represent theories in propositional logic. In this section, we introduce a first-order negation normal form circuit language that represents theories in FOL-DC.

3.2.1 Syntax and Semantics

Definition 3.12 (FO-NNF[6])**.** A formula in first-order negation normal form (FO-NNF) is a rooted directed acyclic graph, where

[6]The first-order negation normal form circuits of Van den Broeck et al. (2011a) are not identical to the ones defined here. There are the following differences: (i) we do not allow for inclusion-

- *leafs* are labeled with a (potentially non-ground) literal, T or F,
- *inner nodes* with *one child* are labeled with $\forall \mathbf{V}, cs$ or $\exists \mathbf{V}, cs$, where cs is a constraint set in FOL-DC and $\mathbf{V}$ is a set of logical or domain variables, and
- *inner nodes* with arbitrarily *many children* are labeled with $\wedge$ or $\vee$.

First-order NNF circuits represent formulas[7] in FOL-DC. Each inner node represents a FOL-DC operator. Edges going out of a node point to the circuits that represent the operands of that node.

Example 3.9. Figure 3.1 depicts three FO-NNF circuits, one for Formula 3.2 from Example 3.4 and two for Formula 3.1 from Example 3.3. Note that FO-NNF circuits are not unique. Grey rectangular leaf nodes represent literals and domain constraints. White round inner nodes represent intensional and extensional operators. These diagrams can be read either top-down or bottom-up. Top-down, they read as formulas in FOL-DC, with quantifiers and operators that apply to their operands. Bottom-up, they can be regarded as circuits, where the leaf nodes are inputs, the inner nodes are logical gates and the root node is the output. In the constraint sets, the domain variables Bird and People are free variables. Hence, these circuits represent formulas that are not sentences.

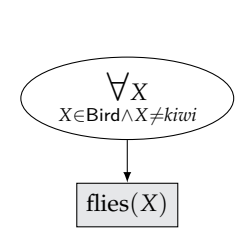
The fact that Figure 3.1c represents Formula 3.1 is not obvious. It should be read as follows. There exists a set of people D with the property that all people in this set are Belgian ($\forall X, X \in \mathsf{D} : \text{belgian}(X)$), and everyone else is not ($\forall X, X \notin \mathsf{D} \wedge X \in \mathsf{People} : \neg\,\text{belgian}(X)$). Hence, this part of the circuit encodes that D represents all Belgians. For precisely this set of people, the circuit requires that they like chocolate ($\forall X, X \in \mathsf{D} : \text{likes}(X, \textit{chocolate})$).

Similar to the grounding semantics of sentences in FOL-DC, each FO-NNF circuit that represents a sentence can easily be transformed into a propositional NNF circuit.

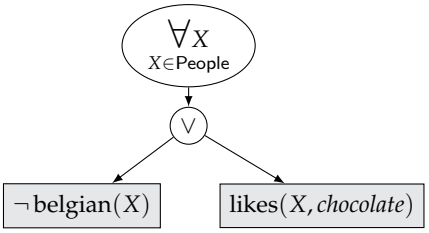
Proposition 3.1. *Each* FO-NNF *circuit that is a sentence can be grounded into a corresponding* NNF *circuit.*

exclusion labels on inner nodes, (ii) we use a different name and syntax for intensional conjunctions and disjunctions, which are called set conjunction $\bigwedge$ and set disjunction $\bigvee$ in Van den Broeck et al. (2011a), (iii) we allow for intensional operators to quantify a set of variables instead of one, (iv) we extend the expressivity of constraint sets with the symbols $<$, $\cup$, $\cap$ and $\setminus$, and (v) we allow for free variables. Compared to the circuits of Van den Broeck (2011b), we do not allow for domain recursion nodes.

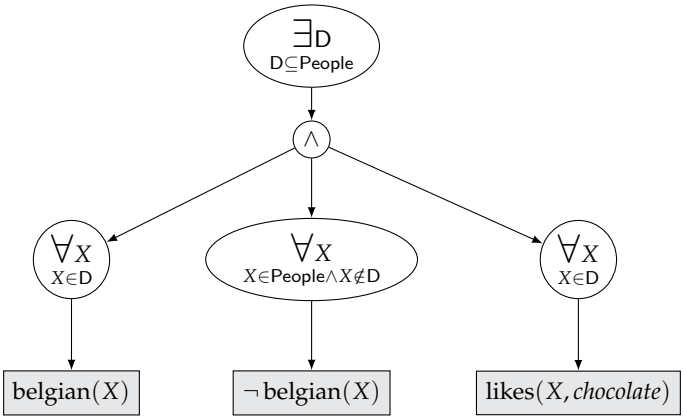
[7] The original knowledge compilation map (Darwiche and Marquis, 2002), which deals with propositional NNF circuits, talks about sentences, not formulas. However, FO-NNF circuits can also represent formulas that have free variables, hence not sentences.



(a) Circuit for Formula 3.2



(b) Circuit for Formula 3.1



(c) Alternative circuit for Formula 3.1

Figure 3.1: Examples of FO-NNF circuits.

Grounding involves replacing intensional conjunctions and disjunctions by extensional ones, as in Equations 3.7 and 3.8. This transformation makes every inner node extensional, as in propositional circuits, and substitutes logical variables by constants, in the end grounding all leaf nodes. These leafs can then be regarded as variables in propositional logic.

3.2.2 Properties

Darwiche and Marquis (2002) analyse the properties of propositional NNF circuits according to several criteria: *completeness*, *inclusion*, *succinctness*, and tractability of *queries* and *transformations*. This section gives a first fragmentary characterization of FO-NNF circuits along these dimensions. For ease of notation,

the abbreviations for circuit languages, such as NNF and FO-NNF, will be used to represent the sets of circuits that are a member of this language.

Completeness

Definition 3.13 (Completeness of a Language)**.** A language $\mathbf{L}_1$ is complete for a language $\mathbf{L}_2$ iff for every formula $\alpha \in \mathbf{L}_2$ there exists an equivalent formula $\beta \in \mathbf{L}_1$.

In propositional knowledge compilation, a language is complete iff it is complete for propositional sentences. For our analysis, it is important to distinguish between two additional languages:

- Formula, for formulas in FOL-DC, and
- Sentence $\subset$ Formula, for sentences in FOL-DC.

Both can be first-order, but propositional formulas can only be sentences. We now have the following language completeness results.

Proposition 3.2. NNF *is a complete language for* Sentence *but not for* Formula*.*

Proof. NNF is a complete language for ground sentences (Darwiche and Marquis, 2002). Because any Sentence has an equivalent ground sentence through the grounding function gr (see Definition 3.11), NNF is also a complete language for Sentence. NNF is not a complete language for Formula, because it lacks the syntax to represent free variables. □

Proposition 3.3. FO-NNF *is a complete language for* Formula *(and* Sentence*).*

Proof. FO-NNF is complete for Formula, because any formula can be turned into a FO-NNF by pushing negations inside of quantifiers and connectives. □

Inclusion and Succinctness

Definition 3.14 (Inclusion)**.** Language $\mathbf{L}_1$ is included in language $\mathbf{L}_2$ iff $\mathbf{L}_1 \subseteq \mathbf{L}_2$.

The FO-NNF language is a strict superset of the NNF language, so FO-NNF includes NNF. A NNF is a FO-NNF where the leaf literals have no logical variable arguments and the inner nodes are restricted to extensional conjunctions and disjunctions.

Darwiche and Marquis (2002) investigate the relative *succinctness* of NNF sentences, where the size of a circuit Σ is measured by its number of edges and is denoted by $|\Sigma|$.

Definition 3.15 (Succinctness)**.** Let $\mathbf{L}_1$ and $\mathbf{L}_2$ be two subsets of FO-NNF. $\mathbf{L}_1$ is at least as succinct as $\mathbf{L}_2$, denoted $\mathbf{L}_1 \leq \mathbf{L}_2$, iff there exists a polynomial p such that for every formula $\alpha \in \mathbf{L}_2$ that has an equivalent representation in $\mathbf{L}_1$, there exists an equivalent formula $\beta \in \mathbf{L}_1$ with $|\beta| \leq p(|\alpha|)$.

Compared to the definition of succinctness by Darwiche and Marquis (2002), we only consider formulas α and β in the comparison for which the languages $\mathbf{L}_1$ and $\mathbf{L}_2$ are both complete.

Through inclusion (NNF $\subset$ FO-NNF), we have the following:

Proposition 3.4. FO-NNF $\leq$ NNF

The negative results from Darwiche and Marquis (2002) carry over, stating that certain subsets of NNF are less succinct.

Proposition 3.5. d-NNF $\not\leq$ FO-NNF, s-NNF $\not\leq$ FO-NNF *and* DNNF $\not\leq$ FO-NNF.

Proof. There is some $\alpha \in$ NNF and therefore $\alpha \in$ FO-NNF for which no equivalent polysize β in d-NNF, s-NNF or DNNF exists. □

The positive results from Darwiche and Marquis (2002), proving that $\mathbf{L} \leq$ NNF for certain languages $\mathbf{L}$, do not imply that $\mathbf{L} \leq$ FO-NNF, because there might exist a formula $\alpha \in$ FO-NNF $\setminus$ NNF without a corresponding $\beta \in \mathbf{L}$, as we show next.

Proposition 3.6. NNF $\not\leq$ FO-NNF

Proof. Consider the sentence

$$\phi_n \equiv \forall X_1, \ldots X_n, X_1 \in \{a, b\} \wedge \cdots \wedge X_n \in \{a, b\} : \mathrm{p}(X_1, \ldots, X_n). \tag{3.9}$$

The size of the FO-NNF circuit for ϕ_n is linear in n. Any representation of ϕ_n as a NNF circuit needs to represent each ground p/n atom as a leaf, of which there are 2^n, and has therefore a size exponential in n. □

Propositions 3.4 and 3.6 together imply that FO-NNF is strictly more succinct than NNF, denoted by FO-NNF $<$ NNF.

Queries and Transformations

Darwiche and Marquis (2002) investigate which *queries* and *transformations* on NNF circuits can be answered in polynomial time. See Tables 3.1 and 3.2 for a list of queries and transformations, and their abbreviations. We refer to Darwiche and Marquis for a formal definition of these problems. This dissertation focuses on the following queries:

- *Consistency checking* (**CO**,**SAT**): checking whether the sentence has at least one model,
- *Validity checking* (**VA**): checking whether every interpretation is a model of the theory,
- *Clausal entailment checking* (**CE**): checking whether the sentence entails a given clause,
- *Implicant checking* (**IM**): checking whether a given term (conjunction) entails the sentence,
- *Equivalence checking* (**EQ**): checking whether two sentences are equivalent,
- *Sentential entailment checking* (**SE**): checking whether one sentence entails the other,
- *Model counting* (**CT**,**#SAT**): counting the number of models (satisfying assignments) of the theory,
- *Model enumeration* (**ME**): enumerating all models of the theory,

and on a subset of transformations:

- *conditioning* (**CD**): setting ground atoms in the theory to true or false,
- *forgetting* (**FO**): removing a set of ground atoms A from a theory ϕ, creating a formula ψ that does not mention the atoms in A, such that for any α not mentioning atoms from A, $\psi \models \alpha$ precisely when $\phi \models \alpha$.

Notation	Query
CO	polytime consistency checking
VA	polytime validity checking
CE	polytime clausal entailment checking
IM	polytime implicant checking
EQ	polytime equivalence checking
SE	polytime sentential entailment checking
CT	polytime model counting
ME	polytime model enumeration

Table 3.1: Notation for queries.

Notation	Transformation
CD	polytime conditioning
FO	polytime forgetting

Table 3.2: Notation for transformations.

For a propositional circuit language to support these queries and transformations, their complexity needs to be *polynomial time in the size of the circuits* and in the size of other inputs (clauses, terms, etc.). Complexity for the **ME** task is also required to be polynomial in the number of models of the sentence. For first-order circuits, we will call a procedure polytime if in addition, it is polynomial in the size of the constraint sets that appear in intensional circuit nodes. This includes the size of the sets of constants that appear in constraints.

Definition 3.16. A procedure operating on a FO-NNF is *polytime* if it is polynomial in the number of edges in the circuit and the size of the node labels.

Because NNF $\subset$ FO-NNF, all the negative results for queries on NNF from Darwiche and Marquis (2002) carry over to FO-NNF circuits: if a polytime query is impossible for some NNF, it cannot be possible for all FO-NNF.

Theorem 3.7 (Queries on FO-NNF)**.** *There is no polytime algorithm for* **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **CT** *or* **ME** *in* FO-NNF*, unless the polynomial hierarchy PH collapses.*

Table 3.3 summarizes these results.

The same reasoning does not apply to transformations. Just because a polytime transformation is not possible from NNF to NNF does not mean it is impossible

Notation	NNF	FO-NNF
CO	○	○
VA	○	○
CE	○	○
IM	○	○
EQ	○	○
SE	○	○
CT	○	○
ME	○	○

Table 3.3: Queries supported by FO-NNF. The symbol '○' means the query is not supported (contingent on complexity assumptions).

from NNF to FO-NNF, which is more expressive. However, we can prove that the single negative result for transforming NNF also holds for FO-NNF.

Theorem 3.8 (Transformations on FO-NNF)**.** *There is no polytime algorithm for* **FO** *in a* FO-NNF*.*

Proof. The proof is analogous to the proof for **FO** in NNF of Darwiche and Marquis (2002). If FO-NNF supports polytime **FO**, it can be used to decide consistency of any CNF in polynomial time. A CNF is already in FO-NNF. Forgetting all its ground atoms evaluates to true iff the CNF is consistent. □

3.3 Subsets of the FO-NNF Language

Even though the FO-NNF language is a theoretically interesting extension of the NNF language, Theorem 3.7 showed that neither language has properties that make it a tractable language: no type of query can be answered efficiently in a NNF or FO-NNF circuit. As with propositional NNF circuits, by putting additional constraints on the nodes of the circuit, one can obtain new languages which do support interesting queries and transformations. This section defines properties of FO-NNF nodes and a new set of circuit languages with these properties.

3.3.1 Constraints on FO-NNF Nodes

This section defines four properties of FO-NNF nodes: *decomposability*, *determinism*, *smoothness* and *automorphism*. The first three properties can also be found in propositional NNF circuits. We generalize these concepts to FO-NNF circuits. The automorphism constraint is new and specific to FO-NNF. The constraints will be defined on sentences. The constraints on formulas are defined as follows.

Definition 3.17. A formula ϕ is decomposable, deterministic, smooth or automorphic iff $\phi\theta$ is respectively decomposable, deterministic, smooth or automorphic for all closing substitutions θ.

Decomposability

Definition 3.18 (Independence)**.** Two sentences ϕ and ψ are *independent*, denoted by $\phi \perp\!\!\!\perp \psi$, when $\mathrm{gr}(\phi)$ and $\mathrm{gr}(\psi)$ do not share any atoms.

Definition 3.19 (Extensional Decomposability)**.** An extensional conjunction $\phi_1 \wedge \cdots \wedge \phi_n$ is decomposable iff for all $i \neq j$, $\phi_i \perp\!\!\!\perp \phi_j$.

Definition 3.20 (Intensional Decomposability)**.** An intensional conjunction $\forall \mathbf{V}, cs \ : \ \phi$ is decomposable iff $\phi\theta_1 \wedge \cdots \wedge \phi\theta_n$ is decomposable, where $\{\theta_1, \ldots, \theta_n\} = \mathrm{solutions}(cs, \mathbf{V})$.

Here, $\mathrm{solutions}(cs, \mathbf{V})$ is the set of *all* solutions of cs for variables $\mathbf{V}$.

Example 3.10. The FO-NNF in Figure 3.1b has a root node that is decomposable, because for any two choices of substitution of X, for example *tintin* and *rubens*, the child theories are independent. In the one case, it mentions atoms $\mathrm{belgian}(tintin)$ and $\mathrm{likes}(tintin, chocolate)$. In the other case, it mentions $\mathrm{belgian}(rubens)$ and $\mathrm{likes}(rubens, chocolate)$.

Determinism

Definition 3.21 (Extensional Determinism)**.** An extensional disjunction $\phi_1 \vee \cdots \vee \phi_n$ is deterministic iff for all $i \neq j$, $\phi_i \wedge \phi_j$ is unsatisfiable.

Definition 3.22 (Intensional Determinism)**.** An intensional disjunction $\exists \mathbf{V}, cs$: ϕ is deterministic iff $\phi\theta_1 \vee \cdots \vee \phi\theta_n$ is deterministic, where $\{\theta_1, \ldots, \theta_n\} = \mathrm{solutions}(cs, \mathbf{V})$.

Example 3.11. The FO-NNF in Figure 3.1b has an extensional disjunction node that is not deterministic. Its operands, belgian(X) and likes$(X, \textit{chocolate})$, can both be satisfied in a model.

Example 3.12. The FO-NNF in Figure 3.1c has an intensional disjunction node that is deterministic. Any pair of distinct solutions to D must have one constant c that is contained in the one set but not in the other. The disjuncts for these solutions cannot both be satisfied, because one requires that belgian(c) is true and the other that belgian(c) is false.

Smoothness

Definition 3.23 (Extensional Smoothness)**.** An extensional disjunction $\phi_1 \vee \cdots \vee \phi_n$ is smooth iff for all i and j, groundings $\text{gr}(\phi_i)$ and $\text{gr}(\phi_j)$ contain the same atoms.

Definition 3.24 (Intensional Smoothness)**.** An intensional disjunction $\exists \mathbf{V}, cs : \phi$ is smooth iff $\phi\theta_1 \vee \cdots \vee \phi\theta_n$ is smooth, where $\{\theta_1, \ldots, \theta_n\} = \text{solutions}(cs, \mathbf{V})$.

Example 3.13. The FO-NNF in Figure 3.1b has an extensional disjunction node that is not smooth. Its operands, belgian(X) and likes$(X, \textit{chocolate})$, mention different sets of atoms.

Automorphism

To define automorphism, we first need to define what it means for two sentences to be equivalent up to a permutation of constants.

Definition 3.25. Let ϕ and ψ be sentences and let $\mathcal{MOD}_\phi$ and $\mathcal{MOD}_\psi$ be their sets of models. Let C be the set of constants in the signature of ϕ and ψ. Sentences ϕ and ψ are *equivalent up to a permutation of constants* iff there exists a bijection (one-to-one mapping) $\pi : C \to C$ which maps $\mathcal{MOD}_\phi$ to $\mathcal{MOD}_\psi$.

Definition 3.26 (Automorphism)**.** An intensional conjunction or disjunction $(\forall, \exists)\mathbf{V}, cs : \phi$ is *automorphic* if either

- $\mathbf{V}$ is a set of *logical variables* and for all solutions $\theta_1, \theta_2 \in \text{solutions}(cs, \mathbf{V})$, the sentences $\phi\theta_1$ and $\phi\theta_2$ are equivalent up to a permutation of constants, or

- $\mathbf{V} = \{\mathsf{D}\}$ is a *domain variable* D and for all solutions $[\mathsf{D}/d_1], [\mathsf{D}/d_2] \in \text{solutions}(cs, \mathsf{D})$ for which $|d_1| = |d_2|$, the sentences $\phi[\mathsf{D}/d_1]$ and $\phi[\mathsf{D}/d_2]$ are equivalent up to a permutation of constants.

Intuitively, automorphism formalizes the property that the names of individual constants are unimportant, similar to Skolem constants. In the case of logical variables, automorphism means that we can reason about all the operands represented by the intensional conjunction or disjunction by looking at a single operand and generalizing its properties (model set, model count, satisfiability, etc.) to the other operands through a bijection of the constants. In the case of a domain variable, it means we can reason about all solutions of the same size by looking at one representative set of constants. There may be 2^n solutions for a domain variable that all have size n.

Example 3.14. The FO-NNF in Figure 3.1b has a root intensional conjunction node that is automorphic, because for any two choices of substitution for X, for example *tintin* and *rubens*, the models of the child theory can be mapped to each other by the mapping $\{tintin \mapsto rubens, rubens \mapsto tintin\}$.

Example 3.15. The FO-NNF in Figure 3.1c has a root intensional disjunction node that is automorphic. For any two substitutions of D by sets of equal size, for example $[\mathsf{D}/\{alice, bob\}]$ and $[\mathsf{D}/\{charlie, dave\}]$, the models of the child circuit are equivalent up to a permutation of constants, for example $\{alice \mapsto charlie, charlie \mapsto alice, bob \mapsto dave, dave \mapsto bob\}$.

3.3.2 Languages

By requiring the above constraints to hold for all nodes in a FO-NNF circuit, we obtain new first-order languages.

Definition 3.27 (FO-DNNF)**.** A FO-DNNF is an FO-NNF where all (intensional and extensional) conjunctions are *decomposable*.

Definition 3.28 (FO-s-NNF)**.** A FO-s-NNF is an FO-NNF where all (intensional and extensional) disjunctions are *smooth*.

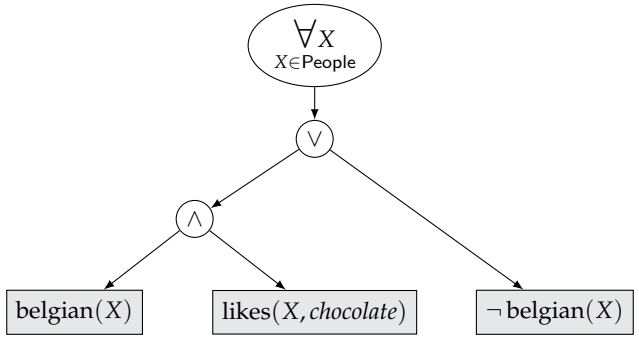
Definition 3.29 (FO-d-NNF)**.** A FO-d-NNF is an FO-NNF where all (intensional and extensional) disjunctions are *deterministic*.

Definition 3.30 (FO-a-NNF)**.** A FO-a-NNF is an FO-NNF where all intensional conjunctions and disjunctions are *automorphic*.

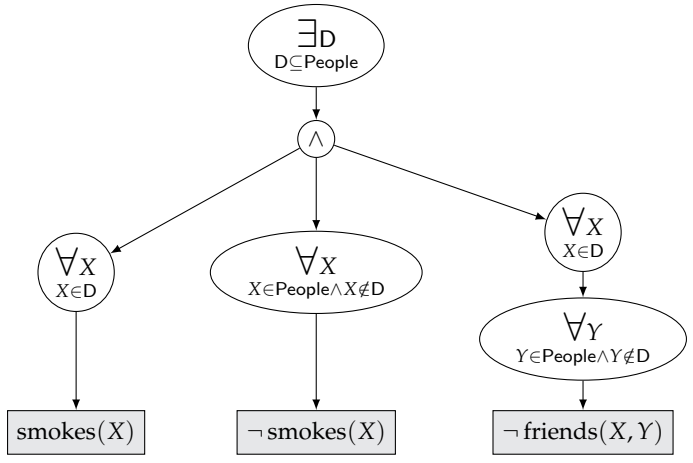
The FO-da-DNNF Language

The FO-da-DNNF language, or *first-order deterministic automorphic decomposable negation normal form*, is the first language of practical interest. Chapter 4 presents an algorithm that compiles theories in CNF into FO-da-DNNF.

Definition 3.31 (FO-da-DNNF)**.** FO-da-DNNF = FO-DNNF ∩ FO-d-NNF ∩ FO-a-NNF



(a) Circuit for Formula 3.1



(b) Circuit for Formula 3.10

Figure 3.2: Examples of FO-da-DNNF circuits.

Example 3.16. The FO-NNF in Figure 3.1c is a FO-da-DNNF. The FO-NNF in Figure 3.1b is decomposable (Example 3.10) and automorphic (Example 3.14), but not deterministic (Example 3.11). Figure 3.2a depicts a FO-da-DNNF for Formula 3.1 that is similar to Figure 3.1b, but that is deterministic.

Example 3.17. Figure 3.2b depicts a FO-da-DNNF equivalent to the theory

$$\forall X, Y, X \in \text{People} \wedge Y \in \text{People}:$$

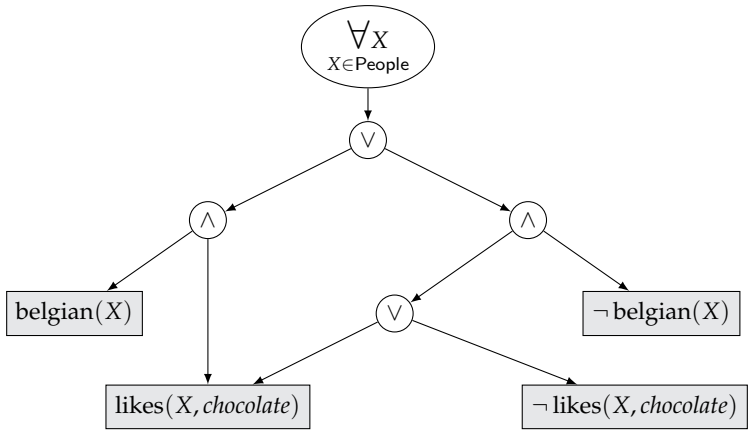
$$\text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y). \tag{3.10}$$

The theory states that smokers are only friends with other smokers. The circuit introduces a new domain variable D, which is a subset of People. It states that there exists such a D for which (i) all people in D are smokers (ii) no other people are smokers and (iii) smokers are not friends with non smokers. Grounding this theory (with any closing substitution), results in a theory equivalent to Formula 3.10.

The FO-sda-DNNF Language

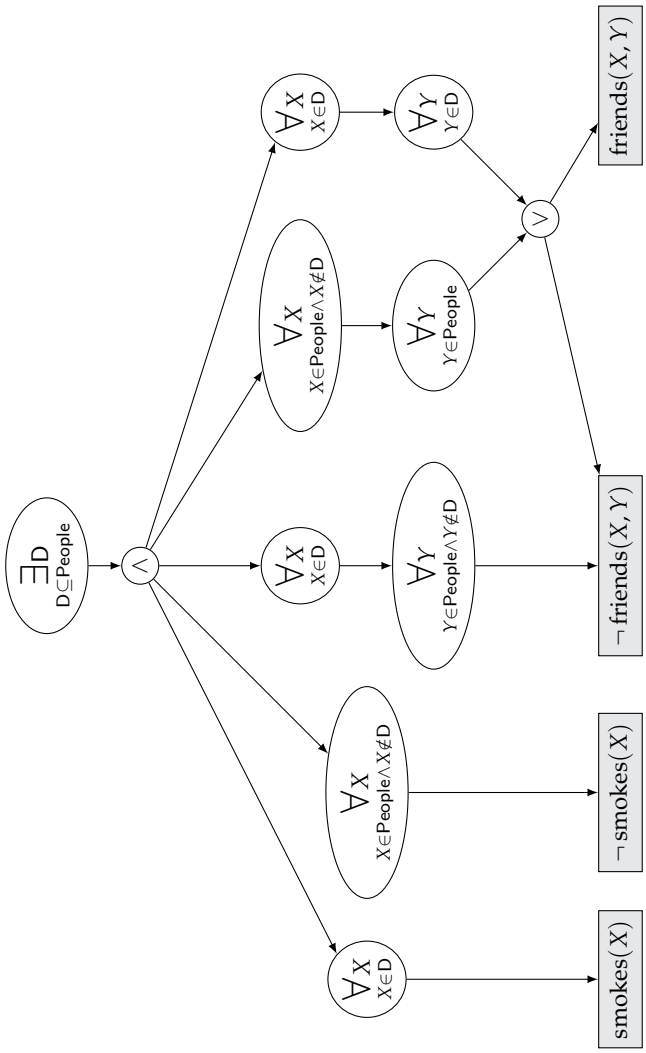
Our second language of practical interest is FO-sda-DNNF. Section 3.4 will present an algorithm to efficiently answer the model counting query (**CT**) on a FO-sda-DNNF circuit. Chapter 4 will introduce an algorithm to transform any FO-da-DNNF circuit into FO-sda-DNNF.

Definition 3.32 (FO-sda-DNNF)**.** FO-sda-DNNF $=$ FO-da-DNNF $\cap$ FO-s-NNF.



(a) Circuit for Formula 3.1

Figure 3.3: Examples of FO-sda-DNNF circuits.



(b) Circuit for Formula 3.10

Figure 3.3: Examples of FO-sda-DNNF circuits.

Example 3.18. Figures 3.1b and 3.2a for Formula 3.1 are not smooth (Example 3.13). Figure 3.3a depicts a FO-sda-DNNF circuit for this formula that is smooth.

Example 3.19. The circuit in Figure 3.2b has an intensional disjunction that is not smooth and is therefore not a FO-sda-DNNF. The reason is that the circuit mentions only atoms $\text{friends}(X, Y)$ for which $X \in \mathsf{D} \wedge Y \in \mathsf{People} \wedge Y \notin \mathsf{D}$. Let for example $\mathsf{People} = \{\textit{tintin}, \textit{rubens}\}$ and consider two solutions for D, being $[\mathsf{D}/\{\textit{tintin}\}]$ and $[\mathsf{D}/\{\textit{rubens}\}]$. In the first case the circuit only mentions the ground atom $\text{friends}(\textit{tintin}, \textit{rubens})$ whereas in the second case it only mentions $\text{friends}(\textit{rubens}, \textit{tintin})$. When considering solutions $[\mathsf{D}/\emptyset]$ and $[\mathsf{D}/\mathsf{People}]$, the circuit does not even mention any friends-atoms. The circuit is not smooth for atoms $\text{friends}(X, Y)$ for which $X \in \mathsf{People} \wedge X \notin \mathsf{D}$ or $Y \in \mathsf{D}$. Figure 3.3b depicts a FO-sda-DNNF for Formula 3.10 that is smooth.

3.4 Properties of Tractable FO-NNF Subsets

This section analyzes the properties of the FO-da-DNNF and FO-sda-DNNF languages within the framework of Darwiche and Marquis (2002). The analysis focuses on the model counting and conditioning tasks. We present an algorithm that computes the model count of an FO-sda-DNNF circuit.

3.4.1 Completeness

It follows from Darwiche and Marquis (2002) and Proposition 3.2 that all subsets of NNF considered here are complete for sentences but not for formulas. This includes the d-DNNF and sd-DNNF languages, which are the propositional counterparts of FO-da-DNNF and FO-sda-DNNF. The completeness of FO-da-DNNF and FO-sda-DNNF for sentences again follows from their grounding semantics.

Proposition 3.9. FO-da-DNNF *and* FO-sda-DNNF *are complete for* Sentence.

Proof. sd-DNNF $\subset$ FO-sda-DNNF $\subset$ FO-da-DNNF and the sd-DNNF language is complete for Sentence. □

However, their completeness for formulas remains an open question.

Open Problem 1. FO-da-DNNF *and* FO-sda-DNNF *are complete for* Formula.

3.4.2 Succinctness

Comparing FO-NNF to FO-da-DNNF, we show that the former is strictly more succinct than the latter.

Theorem 3.10. FO-NNF $<$ FO-da-DNNF *(i.e.,* FO-NNF $\leq$ FO-da-DNNF *and* FO-da-DNNF $\not\leq$ FO-NNF*)*

Proof. In one direction, FO-da-DNNF $\subset$ FO-NNF. For the other, it follows from Darwiche and Marquis (2002) that there exists an $\alpha \in$ NNF for which no polysize representation exists in d-DNNF. Now consider the same α as a member of FO-NNF. To represent it as a FO-da-DNNF circuit, we cannot use any intensional conjunctions or disjunctions, since α contains no logical variables. This means that the subset of FO-da-DNNF we can use is exactly the language d-DNNF, which has no polysize representation of α. □

Comparing FO-da-DNNF to FO-sda-DNNF, we have the following result.

Theorem 3.11. FO-da-DNNF *and* FO-sda-DNNF *are equally succinct (i.e.,* FO-da-DNNF $\leq$ FO-sda-DNNF *and* FO-sda-DNNF $\leq$ FO-da-DNNF*).*

Proof. In one direction, FO-sda-DNNF $\subset$ FO-da-DNNF. In the other, the proof is constructive: Chapter 4 presents an algorithm for smoothing a FO-da-DNNF. □

Comparing to propositional circuits, we have the following results.

Theorem 3.12. FO-da-DNNF $<$ d-DNNF *(i.e.,* FO-da-DNNF $\leq$ d-DNNF *and* d-DNNF $\not\leq$ FO-da-DNNF*)*

Proof. In the one direction, d-DNNF $\subset$ FO-sda-DNNF. The other direction holds because Formula 3.9 is in FO-da-DNNF and has no polysize d-DNNF. □

Theorem 3.13. FO-da-DNNF *and* NNF *are incomparable in size (i.e.,* NNF $\not\leq$ FO-da-DNNF *and* FO-da-DNNF $\not\leq$ NNF*).*

Proof. The first direction holds because Formula 3.9 is in FO-da-DNNF and has no polysize NNF. The proof for the second direction is analogous to the proof of Theorem 3.10: there exists an $\alpha \in$ NNF which has no polysize representation in FO-da-DNNF, because there is none in d-DNNF. □

Because FO-da-DNNF and FO-sda-DNNF are equally succinct, Theorem 3.13 also applies to the comparison of NNF and FO-sda-DNNF.

3.4.3 Model Counting on FO-sda-DNNF Circuits

The motivation for introducing the FO-sda-DNNF language is its support for *tractable model counting* (**CT**). The price we pay for this tractability is that FO-sda-DNNF circuits are less succinct than FO-NNF circuits (Theorem 3.10). The following function computes the model count of a FO-sda-DNNF sentence.

Definition 3.33 (count(.))**.** The base cases of the count function are $\text{count}(\text{F}) = 0$, $\text{count}(\text{T}) = 1$ and $\text{count}(l) = 1$ when l is a *ground literal*. The recursive definition is as follows. For an *extensional conjunction* $\phi_1 \wedge \cdots \wedge \phi_n$,

$$\text{count}(\phi_1 \wedge \cdots \wedge \phi_n) = \text{count}(\phi_1) \times \cdots \times \text{count}(\phi_n).$$

For an *extensional disjunction* $\phi_1 \vee \cdots \vee \phi_n$,

$$\text{count}(\phi_1 \vee \cdots \vee \phi_n) = \text{count}(\phi_1) + \cdots + \text{count}(\phi_n).$$

For an *intensional operator* $(\forall, \exists)\mathbf{X}, cs : \phi$ over a set of *logical* variables $\mathbf{X}$, with $\theta \in \text{solutions}(cs, \mathbf{X})$,

$$\text{count}(\forall \mathbf{X}, cs : \phi) = \text{count}(\phi\theta)^{|\,\text{solutions}(cs,\mathbf{X})|}$$

$$\text{count}(\exists \mathbf{X}, cs : \phi) = |\,\text{solutions}(cs, \mathbf{X})| \times \text{count}(\phi\theta).$$

For an *intensional operator* $(\forall, \exists)\text{D}, cs : \phi$ over a *domain* variable D, with $\Theta_n = \{[\text{D}/d] \mid [\text{D}/d] \in \text{solutions}(cs, \text{D}) \wedge (|d| = n)\}$ and with $\theta_n \in \Theta_n$,

$$\text{count}(\forall \text{D}, cs : \phi) = \prod_n \text{count}(\phi\theta_n)^{|\Theta_n|}$$

$$\text{count}(\exists \text{D}, cs : \phi) = \sum_n |\Theta_n| \times \text{count}(\phi\theta_n).$$

Proposition 3.14. *The* count *function correctly computes the model count of a* FO-sda-DNNF *circuit.*

Proof outline. This follows from (i) the correspondence to the operations of propositional model counting in the grounded circuit and (ii) the fact that a

permutation of constants in the set of models keeps the model count unchanged. □

Example 3.20. The model count of Formula 3.1 (for some concrete set People) can be computed by applying the count function to the FO-sda-DNNF circuit in Figure 3.3a. Its root node is an intensional conjunction, which means that the model count is $\text{count}(\phi\theta)^{|\text{solutions}(cs,X)|}$. We know that the number of solutions to the constraint set $(X \in \text{People})$ is $|\text{People}|$. If we arbitrarily choose the solution θ to be $[X/alice]$, for example, all that remains is to compute the model count of the conjunct corresponding to θ. This circuit is depicted in Figure 3.4. It is

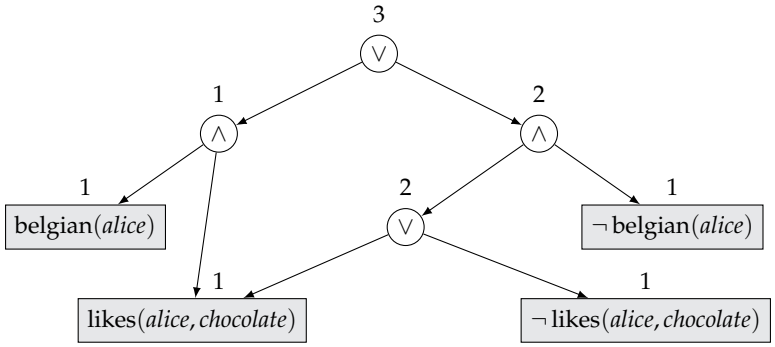


Figure 3.4: Model counting on a conjunct of Figure 3.3a

a ground sentence, and the count function simply performs model counting as in propositional sd-DNNF circuits (see Section 2.2.1). Figure 3.4 labels each subcircuit with its model count. Since the count of the entire conjunct is 3, the model count of Formula 3.1 is $3^{|\text{People}|}$.

Example 3.21. The model count of Formula 3.10 can be computed on the FO-sda-DNNF in Figure 3.3b. The root node is an intensional disjunction of the form $\exists \text{D} \subseteq \text{People} : \phi$. Its model count is $\sum_n |\Theta_n| \times \text{count}(\phi\theta_n)$. The values for n we have to sum over are the possible sizes $|d|$ of solutions $[\text{D}/d]$ of the constraint set $(\text{D} \subseteq \text{People})$. These values range from 0 to $|\text{People}|$. The factor $|\Theta_n|$ represents the number of solutions of size n, which is equal to the binomial coefficient $\binom{|\text{People}|}{n}$. For each n, we can choose θ_n to be some substitution $[\text{D}/d]$ such that $|d| = n$. Then, computing $\text{count}(\phi\theta_n)$ is performed by several computations analogous to the ones in Example 3.20, on circuits rooted in intensional conjunctions. The model count of Formula 3.10 therefore totals $\sum_{n=0}^{|\text{People}|} \binom{|\text{People}|}{n} \times \text{count}(\phi\theta_n)$.

The count function as presented here is naive, in the sense that it does not exploit the circuit structure of the FO-sda-DNNF in order to save computations. It unrolls the circuit into a tree. When *caching* calls to count in each node, or by employing *dynamic programming*, it suffices to compute the model count of each sentence only once and reuse the result for each count call. We now have the following main result.

Theorem 3.15. *The* FO-sda-DNNF *language supports polytime* **CT***, assuming a bounded number of intensional nodes that quantify over a domain variable on all directed paths.*

Proof. The count function recurses on the FO-sda-DNNF structure. In extensional nodes and intensional nodes quantifying over logical variables, it makes one recursive call per child node. In intensional nodes quantifying over a domain variable, however, it makes n recursive calls, for n distinct substitutions. The number of calls made to copies of an individual circuit node this way is exponential in the number of intensional ancestor nodes that quantify over a domain variable. When we bound this number, the count becomes polynomial in the size of the circuit and its labels. □

3.4.4 Conditioning a FO-sda-DNNF Circuit

Next, we show that conditioning a FO-sda-DNNF circuit is #P-hard in general, by showing that #2SAT is reducible to it. We then look at more specific conditioning tasks, when the term being conditioned on consists of propositions or unary literals. Finally, we look at the implications for other languages.

Conditioning on Arbitrary Terms

A kCNF formula is a CNF with k literals per clause. kSAT is the problem of deciding the satisfiability of a kCNF formula. The model counting problem for kCNF formulas is called #kSAT.

Example 3.22. The following formula is in 2CNF:

$$(a \vee b) \wedge (a \vee \neg c) \wedge (\neg c \vee \neg d)$$

Satisfiability of a 2CNF (2SAT) is decidable in polynomial time. However, #2SAT is #P-complete (Valiant, 1979), which implies that it is not solvable in polynomial time unless $P = NP$.

Lemma 3.16. *Each propositional* 2CNF *can be represented by conditioning the theory*[8]

$$\begin{aligned}
&\forall X, Y \in \mathsf{Prop}: \quad \mathrm{p}(X) \vee \quad \mathrm{p}(Y) \vee \neg\, \mathrm{c}_1(X, Y) \\
&\forall X, Y \in \mathsf{Prop}: \quad \mathrm{p}(X) \vee \neg\, \mathrm{p}(Y) \vee \neg\, \mathrm{c}_2(X, Y) \\
&\forall X, Y \in \mathsf{Prop}: \neg\, \mathrm{p}(X) \vee \neg\, \mathrm{p}(Y) \vee \neg\, \mathrm{c}_3(X, Y)
\end{aligned} \tag{3.11}$$

Atoms $\mathrm{p}(X)$ represent propositions from the set $\mathsf{Prop} = \{a, b, \dots\}$ (e.g., $\mathrm{p}(a)$ and $\mathrm{p}(b)$ represent propositions a and b). Atoms $\mathrm{c}_i(X, Y)$ represent clauses, of any of the three types (depending on the number of negated literals), consisting of atoms X and Y (e.g., $\mathrm{c}_2(a, b)$ represents the clause $a \vee \neg b$). The c_i-predicates encode which clauses appear in the 2CNF. Conditioning on a positive c_i literal includes the clause of type i for the given propositions in the 2CNF. For example, conditioning on $\mathrm{c}_1(a, b2)$ adds $\mathrm{p}(a) \vee \mathrm{p}(b)$ to the theory. Conditioning on a negative c-literal omits the clause for the given propositions from the theory. For example, conditioning on $\neg\, \mathrm{c}_2(a, b)$ excludes $\mathrm{p}(a) \vee \neg\, \mathrm{p}(b)$ from the theory.

Example 3.23. Conditioning Theory 3.11 on the term

$$\mathrm{c}_1(a, b) \wedge \neg\, \mathrm{c}_1(a, a) \wedge \cdots \wedge \neg\, \mathrm{c}_1(d, d) \wedge$$

$$\mathrm{c}_2(a, c) \wedge \neg\, \mathrm{c}_2(a, a) \wedge \cdots \wedge \neg\, \mathrm{c}_2(d, d) \wedge$$

$$\mathrm{c}_3(c, d) \wedge \neg\, \mathrm{c}_3(a, a) \wedge \cdots \wedge \neg\, \mathrm{c}_3(d, d)$$

results in the theory

$$(\mathrm{p}(a) \vee \mathrm{p}(b)) \wedge (\mathrm{p}(a) \vee \neg\, \mathrm{p}(c)) \wedge (\neg\, \mathrm{p}(c) \vee \neg\, \mathrm{p}(d)),$$

which is isomorphic to the 2CNF of Example 3.22.

Theorem 3.17. *In any formal system expressive enough to represent Theory 3.11, either conditioning on literals with arity* ≥ 2 *or model counting is #P-hard.*

Proof. Theory 3.11 can be conditioned on binary (arity two) literals to represent any 2CNF. A subsequent model counting step can solve any #2SAT problem.

[8] The notation $\forall X, Y \in \mathsf{Prop}$ is syntactic sugar for $\forall X, Y, X \in \mathsf{Prop} \wedge Y \in \mathsf{Prop}$.

This shows that a #P-complete problem is reducible to conditioning and model counting on Theory 3.11, which means it must be at least as hard as any problem in #P, or #P-hard. □

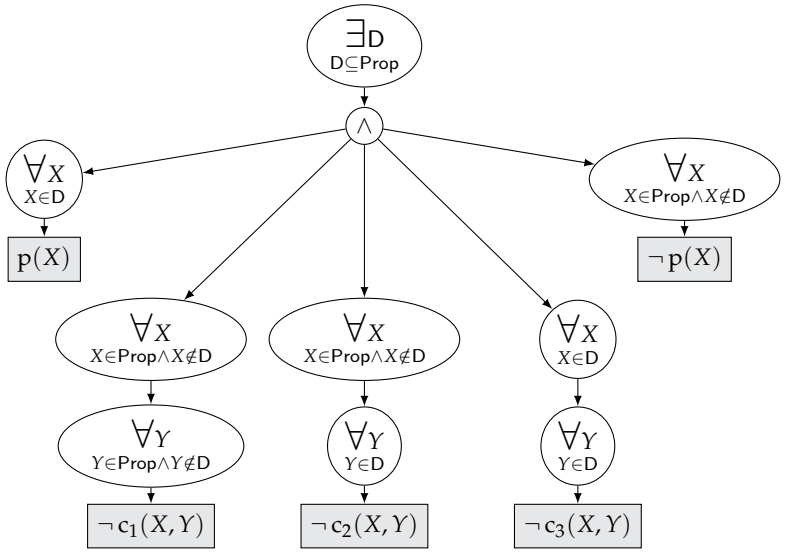


Figure 3.5: FO-da-DNNF circuit for Theory 3.11

Since Theory 3.11 has a representation in FO-da-DNNF, as shown in Figure 3.5, and FO-da-DNNF circuits support polytime model counting (Theorem 3.15), we can specialize Theorem 3.17 as follows.

Corollary 3.18. *There is no polytime algorithm for* **CD** *in* FO-da-DNNF, *unless P=NP.*

Conditioning on Propositions

Theorem 3.17 talks about support for conditioning on binary literals. This leaves the possibility that we can condition a FO-sda-DNNF on propositions (i.e., literals with arity zero) or unary relations (i.e., literals with arity one).

Conditioning a FO-NNF on propositions works the same way as conditioning a NNF circuit. All that it requires is replacing terminal nodes that represent that literal by a T or F terminal.

Proposition 3.19. *A* FO-da-DNNF *circuit can be conditioned on literals with arity zero in polynomial time.*

Proof outline. Following the results in the propositional setting (Darwiche and Marquis, 2002), conditioning on γ preserves the properties of extensional nodes. For determinism, we have $\neg(\alpha \wedge \beta) \Rightarrow \neg(\alpha|\gamma \wedge \beta|\gamma)$ and for decomposability, we have $(\alpha \perp\!\!\!\perp \beta) \Rightarrow (\alpha|\gamma \perp\!\!\!\perp \beta|\gamma)$. Deterministic and decomposable properties of intensional nodes are preserved because, for each closing subsitution, they correspond to an extensional node, which preserves the properties. Automorphism is preserved because conditioning on a literal with arity zero preserves equivalence up to permutation of constants. This is the case because corresponding interpretations are removed from the set of models of both equivalent formulas. □

Conditioning on Unary Relations

Unlike the propositional setting, where each leaf node represents a single proposition, a FO-d-DNNF circuit terminal node can be a non-ground literal, which represents an entire *set* of ground literals. For example, the smokes(X) terminal in Figure 3.2b can refer to any person X. The conditioning term does not necessarily provide the truth value for each grounding of a non-ground terminal. Even if the evidence provides the truth value for each grounding, they could be different. Thus, conditioning a circuit on a subset of the groundings requires partitioning this set of literals (i.e., into those that are true, false and unknown). Each of these partitions needs to be treated separately. Therefore, given an arbitrary FO-d-DNNF, it is unclear if it can be conditioned on any unary literal.

Open Problem 2. *Can a* FO-da-DNNF *circuit be conditioned on literals with arity one in polynomial time?*

However, in Section 5.5.3, we will describe a technique for compiling a type of FO-da-DNNF circuits that do support conditioning on unary relations.

Conditioning and Consistency Checking

Corollary 3.18 raises the question as to whether the inability to efficiently condition on binary relations is a property specific to the FO-da-DNNF language. We can make an analogous argument based on 3CNFs, which can be represented

by conditioning the theory

$$\begin{aligned}
&\forall X, Y, Z \in \mathsf{Prop}: \quad \mathrm{p}(X) \vee \quad \mathrm{p}(Y) \vee \quad \mathrm{p}(Z) \vee \neg\, \mathrm{c}_1(X, Y, Z) \\
&\forall X, Y, Z \in \mathsf{Prop}: \quad \mathrm{p}(X) \vee \quad \mathrm{p}(Y) \vee \neg\, \mathrm{p}(Z) \vee \neg\, \mathrm{c}_2(X, Y, Z) \\
&\forall X, Y, Z \in \mathsf{Prop}: \quad \mathrm{p}(X) \vee \neg\, \mathrm{p}(Y) \vee \neg\, \mathrm{p}(Z) \vee \neg\, \mathrm{c}_3(X, Y, Z) \\
&\forall X, Y, Z \in \mathsf{Prop}: \neg\, \mathrm{p}(X) \vee \neg\, \mathrm{p}(Y) \vee \neg\, \mathrm{p}(Z) \vee \neg\, \mathrm{c}_4(X, Y, Z)
\end{aligned} \tag{3.12}$$

Since 3SAT is NP-complete, we have the following.

Proposition 3.20. *In any formal system expressive enough to represent Theory 3.12, either conditioning on literals with arity ≥ 3 or consistency checking is NP-hard.*

Proof. Theory 3.12 can be conditioned on ternary literals to represent any 3CNF. A subsequent consistency checking step can solve any 3SAT problem. □

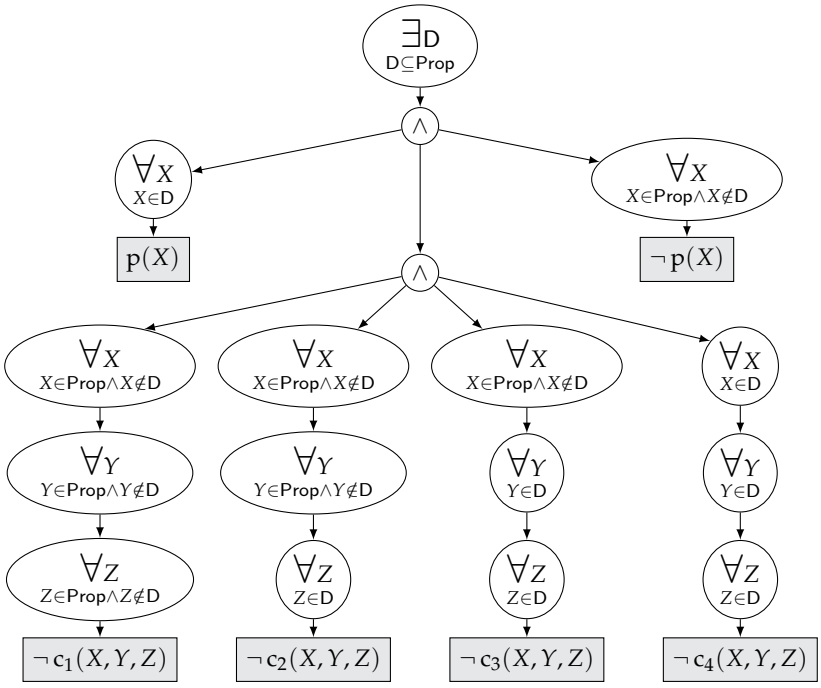


Figure 3.6: FO-da-DNNF circuit for Theory 3.12

Theory 3.12 has a representation in FO-da-DNNF, as shown in Figure 3.6. This result does not provide further insight into FO-da-DNNF circuits, as conditioning them is already #P-hard. However, this result does provide insight into which transformations are efficient in other (yet to be defined) circuit languages that can perform consistency checking, but not model counting, in polynomial time. These correspond to the capabilities of propositional DNNF circuits (Darwiche and Marquis, 2002).

3.4.5 Support for Queries

We will now give a rudimentary classification of queries supported by FO-da-DNNF circuits, other than model counting. Because there exists a polynomial algorithm to convert FO-da-DNNF circuits into FO-sda-DNNF circuits (see Chapter 4), these results (and the ones in the previous sections) apply equally to FO-sda-DNNF circuits.

Proposition 3.21. *There is a polytime algorithm for* **CO** *and* **VA** *in* FO-da-DNNF*, assuming a bounded number of intensional nodes that quantify over a domain variable on all directed paths.*

Proof. Support for **CO** and **VA** are implied by support for **CT** (Theorem 3.15). □

Proposition 3.22. **SE** *is not supported in* FO-da-DNNF*, unless PH collapses.*

Proof. Follows from the same property for d-DNNF and d-DNNF $\subset$ FO-da-DNNF. □

Theorem 3.23. **CE** *is not supported in* FO-da-DNNF*, unless P=NP.*

Proof. Each 3CNF can be represented by a subset of the groundings of Theory 3.12. Recall that atoms $\mathrm{p}(X)$ represent propositions and atoms $\mathrm{c}_i(X, Y, Z)$ represent clauses, of any of the four types. Let γ_1 to γ_n be ground atoms for the c_i predicates. Figure 3.6 depicts a FO-da-DNNF circuit for Theory 3.12. This circuit entails the clause $\neg\gamma_1 \vee \cdots \vee \neg\gamma_n$ iff the 3CNF containing the clauses represented by γ_1 to γ_n is unsatisfiable. This means we can decide 3SAT in polynomial time if **CE** is polynomial in FO-da-DNNF. □

Theorem 3.24. IM *is not supported in* FO-da-DNNF, *unless* FO-da-DNNF *is not complete for* Formula *or P=co-NP.*

Proof. Consider the theory

$$\begin{aligned}
&\exists X, Y, Z \in \mathsf{Prop} : \quad \mathrm{p}(X) \wedge \quad \mathrm{p}(Y) \wedge \quad \mathrm{p}(Z) \wedge \mathrm{t}_1(X, Y, Z) \\
\cdots \vee\ &\exists X, Y, Z \in \mathsf{Prop} : \quad \mathrm{p}(X) \wedge \quad \mathrm{p}(Y) \wedge \neg \mathrm{p}(Z) \wedge \mathrm{t}_2(X, Y, Z) \\
\cdots \vee\ &\exists X, Y, Z \in \mathsf{Prop} : \quad \mathrm{p}(X) \wedge \neg \mathrm{p}(Y) \wedge \neg \mathrm{p}(Z) \wedge \mathrm{t}_3(X, Y, Z) \\
\cdots \vee\ &\exists X, Y, Z \in \mathsf{Prop} : \neg \mathrm{p}(X) \wedge \neg \mathrm{p}(Y) \wedge \neg \mathrm{p}(Z) \wedge \mathrm{t}_4(X, Y, Z)
\end{aligned} \tag{3.13}$$

Similar to the proof of Theorem 3.23, validity of any 3DNF (UN3SAT) can be decided by checking whether a term of t_i atoms implies Theory 3.13 (after substituting the domain variable Prop for a set of propositions). Since Formula 3.13 is a formula and not a sentence (Prop is free), we do not know whether a FO-da-DNNF circuit for it exists (Open Problem 1). If FO-da-DNNF is complete for Formula, a circuit exists and because Theory 3.13 leaves Prop free, its size will be independent of the number of propositions in our 3DNF. After substituting a set of constants for Prop, we could then check validity in polynomial time using the **IM** query. This is not possible for a co-NP-complete problem, unless P=co-NP. □

Table 3.4 gives an overview of all results for supported queries.

Open Problem 3. *Question marks in Table 3.4.*

3.5 Related Work

This section describes the relations between the proposed FO-NNF language, resolution, DPLL search and other first-order knowledge compilation techniques, including other first-order circuit languages.

3.5.1 Relation to the Resolution Principle

Resolution (Robinson, 1965) is an influential algorithm for automated theorem proving in first-order and propositional logic. It will be the subject of

Notation	NNF	d-DNNF	FO-NNF	FO-da-DNNF FO-sda-DNNF
CO	○	✓	○	✓*
VA	○	✓	○	✓*
CE	○	✓	○	○
IM	○	✓	○	○†
EQ	○	?	○	?
SE	○	○	○	○
CT	○	✓	○	✓*
ME	○	✓	○	?

Table 3.4: Queries supported by FO-NNF subsets. The symbol '○' means the query is not supported (contingent on complexity assumptions), '✓' means the query is supported, '?' means unknown, '$\star$' means contingent on a bounded number of intensional nodes on all paths and '†' means contingent on a completeness assumption.

Section 5.3.2, when we talk about lifted inference algorithms. Already here, we want to point out the relationship between propositional d-DNNF circuits, DPLL search (Davis and Putnam, 1960; Davis, Logemann, and Loveland, 1962) and resolution, and the implications for their first-order generalizations.

- Huang and Darwiche (2005) show how d-DNNF circuits represent the trace of an exhaustive DPLL search with *decomposition*, that is, detection of disjoint components.
- It is well known that every tree resolution can be found inside of a DPLL refutation and that every DPLL proof contains a tree resolution (Beame, Kautz, and Sabharwal, 2004). Modern conflict-driven clause-learning SAT solvers even simulate general resolution (Pipatsrisawat and Darwiche, 2011).

This connects those three algorithms at the propositional level and is depicted in Figure 3.7.

In addition, we can individually connect these algorithms to their first-order counterparts.

- A FO-da-DNNF circuit is a template for a large propositional d-DNNF circuit.

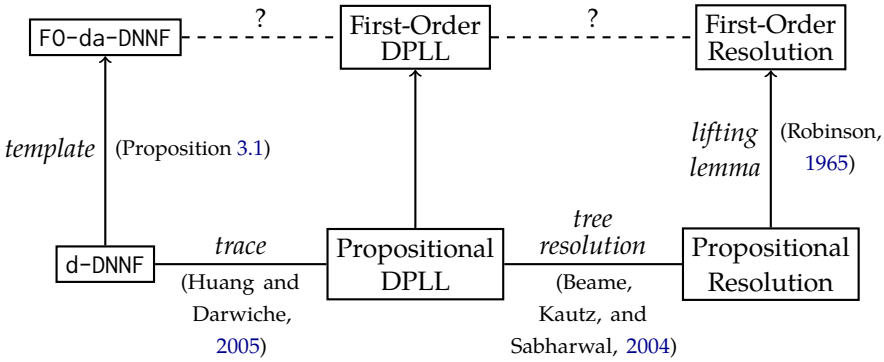


Figure 3.7: Relationship between knowledge compilation, DPLL search and resolution in the propositional and first-order case.

- First-order generalizations of DPLL search have been proposed (Baumgartner, 2000; Baumgartner and Tinelli, 2008; Gogate and Domingos, 2011).
- Robinson (1965) showed that every step of first-order resolution performs a potentially infinite number of propositional resolution steps. This is called the lifting lemma and will be discussed in more detail in Section 5.3.2.

Intuitively, we can argue that the relations between the propositional algorithms also hold between the first-order algorithms. Still, we have the following open problem.

Open Problem 4. *How can we formalize the relationship between first-order knowledge compilation, first-order DPLL and first-order resolution, as represented by the question marks in Figure 3.7?*

3.5.2 Compiling First-Order Logic

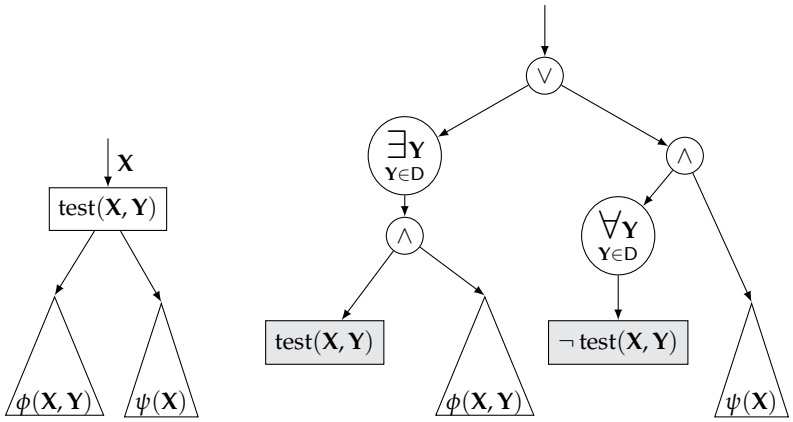
The problem of first-order *approximate* knowledge compilation has previously been considered in the context of *Horn approximations* (Selman, Kautz, et al., 1996; Del Val, 1996; Del Val, 2005). This work compiles a theory into a set of (first-order) Horn clauses whose models are either a superset or a subset of the models of the original theory. These theories permit efficient inference for certain queries. For example, when a Horn upper-bound approximation of a

theory entails a sentence α, the original theory also entails α. Otherwise, no conclusions can be drawn. Hence, these compiled Horn theories are called approximations.

The following circuit languages were developed outside of the broader knowledge compilation context. Therefore, there is no exhaustive analysis of their supported queries, transformations, etc.

First-Order Logical Decision Trees

A *first-order logical decision tree* (Blockeel and De Raedt, 1998) is a binary tree whose leafs are labeled T or F and whose inner nodes are labeled with decisions, that is, conjunctions of first-order literals. The semantics of these trees is in terms of an equivalent logic program. The free variables in the decisions are implicitly existentially quantified. Figure 3.8 shows a decision node and its corresponding FO-NNF circuit (assuming a domain of discourse D).



(a) Decision Node (b) FO-NNF Circuit

Figure 3.8: A first-order decision tree node and its corresponding FO-NNF circuit. The **X**-variables are assumed to appear in ancestor decision nodes.

Gillis and Van den Bussche (2012) showed that the expressivity of first-order decision trees is restricted to Boolean combinations of safe existential sentences. This class of theories is a subset of the theories that have a representation in

prenex normal form, both with quantifiers $\forall^*\exists^*$ and $\exists^*\forall^*$. Hence, first-order decision trees are not a complete language for `Formula` or `Sentence`.

The advantage of first-order decision trees is that they can efficiently be learned from data (logical interpretations). This raises the question whether `FO-NNF` circuits can be learned as well. One approach would be to adapt a first-order decision tree learner to produce a decision graph `FO-NNF`, also consisting of the decision template in Figure 3.8, but where the $\phi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X})$ subcircuits can share nodes.

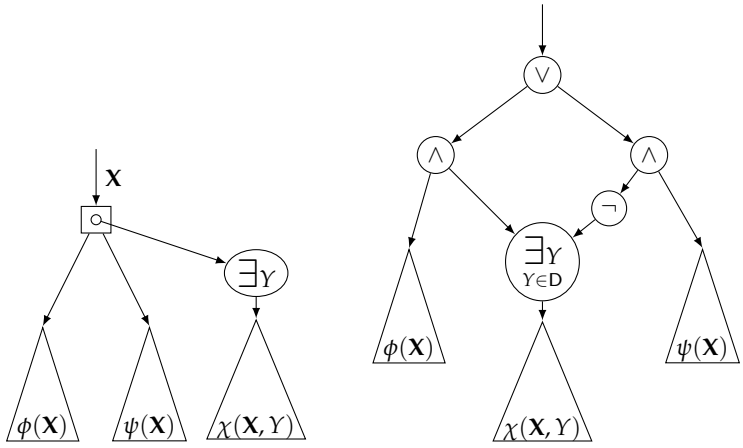
First-Order Binary Decision Diagrams

First-order binary decision diagrams (BDD) (Schneider, Kumar, and Kropf, 1993) were proposed in the hardware verification literature as a tool to decide satisfiability of first-order sentences. Because these circuits have no normal form (contrary to the propositional case), these circuits cannot be used for the same tasks that propositional BDDs are used for. It is not possible to check semantic equivalence of two BDDs, only syntactic equivalence, which implies semantic equivalence. In that regard, this approach is similar to approximate knowledge compilation to Horn theories.

Another language for first-order BDDs was proposed by Posegga (1993) and Goubault (1995). These circuits are also not canonical, even though they have ordered decision nodes. The main difference between these BDDs and the ones of Schneider, Kumar, and Kropf is that now, the decisions are themselves first-order sentences represented by first-order BDDs, as depicted in Figure 3.9. In that regard, these circuits are similar to sentential decision diagrams (Darwiche, 2011). The interaction between logical variables and quantifiers is different in a first-order decision tree and in a first-order BDD, as can be seen from comparing Figure 3.8b to Figure 3.9b.[9] In both cases, the extensional disjunctions in the circuit representation are deterministic. However, the extensional conjunctions are not necessarily decomposable, which precludes model counting on these circuits.

BDD-driven first-order satisfiability procedures (Déharbe and Ranise, 2002) are used to check satisfiability of a ground theory with first-order background

[9] To represent first-order BDDs, we cannot use negation normal form circuits. Instead, we use a negation node in the circuit language, as in Wachter and Haenni (2006).



(a) Decision Node (b) First-Order Circuit

Figure 3.9: A decision node in the BDD language of Goubault and its corresponding first-order circuit. The **X**-variables are assumed to appear in ancestor quantifiers.

knowledge by compiling the ground theory into a binary decision diagram and evaluating the circuit using a theorem prover for the background knowledge.

BDDs for first-order predicate logic (Groote and Tveretina, 2003) can represent sentences in Skolem normal form. For any first-order sentence, one can obtain an equisatisfiable theory in this form by Skolemization. However, the result will not be an equivalent theory and can have a different model count. These BDDs assume an order on the logical tests and therefore correspond to propositional OBDDs. Again, these circuits are not unique. The BDD for a unsatisfiable theory is not necessarily equal to the F terminal. Reducing an unsatisfiable BDD to the F terminal still involves a search procedure that modifies the BDD.

First-order decision diagrams (Wang, Joshi, and Khardon, 2008) are used to solve relational Markov decision processes. They are a first-order generalization of algebraic decision diagrams (Bahar et al., 1997), based on the work of Groote and Tveretina. When used with a Boolean range, these correspond to first-order BDDs. In their original formulation, these circuits only express existential sentences, similar to first-order logical decision trees (albeit with a slightly modified semantics). Joshi and Khardon (2011) recently proposed *generalized* first-order decision diagrams which can additionally express

universal quantification.

3.6 Conclusions

This chapter presented a new circuit language for first-order knowledge compilation, called first-order negation normal form. These circuits represent theories in a variant of first-order logic with domain constraints associated with the logical variables. We defined the syntax and semantics of this logic and its circuit language. We identified subsets of first-order negation normal form that permit tractable model counting. For the proposed circuit languages, we gave an initial classification of supported queries and transformations and analyzed their relative succinctness. This included a procedure to efficiently compute the model count of a `FO-sda-DNNF` circuit and decide its consistency and validity.

4

Compilation Algorithm

This chapter presents two algorithms that compile theories in FOL-DC into target FO-NNF languages that support tractable inference. We first present an algorithm to compile a FO-CNF into a FO-da-DNNF circuit. Second, we present an algorithm to convert any FO-da-DNNF circuit into an equivalent FO-sda-DNNF circuit. For these circuits, there is an efficient procedure to decide consistency, validity and to compute the model count.

The first compilation algorithm needs to enforce the properties of FO-da-DNNF circuits, that is, *determinism*, *decomposability* and *automorphism*. This is done by transforming the input theory in several steps, introducing new domain variables, reordering quantifiers and decomposing the problem in such a way that these properties hold. An important step in the compilation process is to detect and exploit symmetries in the input model. This step is called shattering.

The second compilation algorithm modifies a given FO-da-DNNF circuit to enforce the *smoothness* property in the circuit. This is done by keeping track of which atoms are represented by each subcircuit and inserting new child nodes at every disjunctive node that is not smooth.

Section 4.1 gives an outline of the compilation algorithm to FO-da-DNNF. The next sections present individual compilation rules used by the algorithm.

Section 4.2 deals with rules that do not require shattering and generate extensional circuit nodes. **Section 4.3** then explains a procedure, called shattering, to make the symmetries in the input model explicit. It shows that after applying this transformation, every intensional conjunction in a theory becomes automorphic. **Section 4.4** presents the next set of compilation rules, that generate intensional, or first-order circuit nodes. **Section 4.5** describes the last compilation rule, which grounds the theory. **Section 4.6** presents the second algorithm, for first-order smoothing. Finally, **Section 4.7** discusses related work.

An early version of this work was published in

> G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011a). "Lifted probabilistic inference by first-order knowledge compilation". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. Menlo Park, California, pp. 2178–2185
>
> G. Van den Broeck (2011b). "On the completeness of first-order knowledge compilation for lifted probabilistic inference". In: *Advances in Neural Information Processing Systems 24 (NIPS)*, pp. 1386–1394

The explanation of preemptive shattering was published in Van den Broeck, Choi, and Darwiche (2012). The discussion here goes beyond the published work by adding support for free variables in the input theories, generalizing certain compilation rules and proving more properties of the algorithms.

4.1 Outline of FO-da-DNNF Compilation

This section presents the outline of the algorithm for compilation to FO-da-DNNF.

4.1.1 Input and Output

The compilation algorithm takes as *input* a FO-CNF, that is, a conjunction of Skolem normal form clauses.

Definition 4.1 (Skolem Normal Form)**.** A FOL-DC formula in *Skolem normal form* is a sequence of intensional conjunctions followed by a formula in FOL.

Hence, a FO-CNF is a conjunction of constrained clauses, where each constrained clause is of the form $\forall \mathbf{X}, cs : l_1 \vee \cdots \vee l_n$, and where $l_1, \ldots, l_n$ are literals. These literals contain logical variables from $\mathbf{X}$ and free logical variables.

Note that the set $\mathbf{X}$ can be empty ($\emptyset$), which means the clause is equivalent to $l_1 \vee \cdots \vee l_n$ when the constraints cs are satisfied, and to true (the neutral element for conjunction) otherwise. In the former case, cs has exactly one solution, which is the empty substitution $[]$. In the latter case it has no solutions.

We will not use the full expressivity of FO-CNF in this chapter, but require that

- the quantified variables $\mathbf{X}$ are all logical variables, and
- the constraint set cs consists of equality constraints $=$, element constraints $\in$, or their negation, and can contain free domain variables, free logical variables, or logical variables from $\mathbf{X}$. We disallow $\subseteq$ and $<$ constraints.

The *output* of the compilation algorithm is a FO-da-DNNF circuit that is equivalent to the original FO-CNF, or contains `nil` in case the algorithm cannot compile its input. Whether or not this can happen is strongly related to the topics of completeness and liftability, which are discussed in more detail in Chapter 6. There, we will show that the algorithm presented in this chapter is guaranteed to be able to compile any input theory with up to two bound logical variables per formula.

4.1.2 Compilation Rules

Algorithm 1 shows the outline of the Compile procedure. It uses a set of compilation rules, which in turn can recursively call the Compile function. The purpose of the compilation rules is to transform a FO-CNF into a set of simplified FO-CNFs that are combined using a FO-da-DNNF operator that satisfies the determinism, decomposability and automorphism properties (cf., Section 3.3.1). The simplified FO-CNFs are then the inputs of recursive calls to Compile. We continue compiling the simplified FO-CNFs until they become literals, which are the leafs of the FO-da-DNNF circuit.

Sections 4.2 to 4.5 will present a collection of compilation rules. A compilation rule applies when its *preconditions* are met. The FindRule function of Algorithm 1 selects the first compilation rule that applies, in the following order:

Algorithm 1 Compile(Δ)

Input.

Δ: A set of universally quantified FOL-DC clauses

Supporting Functions.

CacheGet(.)	Returns cache entry, nil if empty
CacheSet(.,.)	Sets cache entry
FindRule(.)	Finds a compilation rule whose preconditions are satisfied, nil if none
ApplyRule(.,.)	Applies a compilation rule to a set of clauses

Function.

```
1: if CacheGet(Δ) ≠ nil then
2:     return CacheGet(Δ)                    // retrieve precomputed circuit from cache
3: else
4:     nnf ← nil                             // compile new NNF circuit and store it in cache
5:     if Δ = {γ} and (γ = T or γ = F or γ is a literal) then
6:         nnf ← γ                                                    // return leaf node
7:     else if FindRule(Δ) ≠ nil then
8:         rule ← FindRule(Δ)        // there exists an applicable compilation rule
9:         nnf ← ApplyRule(Δ, rule)
10:    else
11:        nnf ← nil                                                  // compilation fails
12:    CacheSet(Δ, nnf)
13:    return nnf
```

1. UnitPropagate (Section 4.2.1)
2. VacuousConjunction (Section 4.4.1)
3. Independence (Section 4.2.2)
4. ShannonDecomposition (Section 4.2.3)
5. ShatteredCompilation (Section 4.3.4)
6. IndependentSingleGroundings (Section 4.4.3)
7. IndependentPairedGroundings (Section 4.4.4)
8. AtomCounting (Section 4.4.6)
9. (Ground (Section 4.5))

Similarly, after performing ApplyRule, the *postconditions* of the given rule hold. We describe the pre- and postconditions of every compilation rule in detail.

Before compiling a theory, we attempt to retrieve its circuit from a cache of previously compiled theories. Reusing previously computed FO-da-DNNFs turns the output 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. A more sophisticated caching scheme could attempt to reuse a cached circuit that θ-subsumes (Robinson, 1965) the theory to be compiled, as in for instance Tamaki and Sato (1986).

4.1.3 Terminology and Notation

As some additional terminology, we will call an atom, literal or clause a *constrained atom, literal or clause*, if it is prefixed by an intensional conjunction in FOL-DC. The function $\text{atom}_c(.)$ maps a constrained clause $\forall \mathbf{X}, cs : l_1 \vee \cdots \vee l_n$ to a set of constrained atoms $\{(\forall \mathbf{X}, cs : a_1), \ldots, (\forall \mathbf{X}, cs : a_n)\}$, where a_i is the atom of literal l_i. For ease of notation, we will interchangeably represent theories as sets of constrained clauses, clauses as sets of literals, conjunctions of constraints as sets of constraints and substitutions as sets of equality constraints. The function $\text{bvars}(.)$ maps a formula in FOL-DC to its *bound variables* (set of variables quantified by the intensional operators in the formula).

We will furthermore assume that certain types of *constraint propagation* take place in the input FO-CNF. This includes the removal of equality constraints. For example, a clause γ whose constraint set includes the constraint $X_i = X_j$ between quantified variables is replaced by $\gamma[X_i / X_j]$, which no longer contains the variable X_j. A similar propagation applies when the constraint set includes constraints of the form $X = t$, where X is a quantified variable and t some logical term. Another example of constraint propagation is the constraint set $X \in \mathsf{D} \wedge \mathsf{D} \subseteq \mathsf{F}$, which simplifies to $X \in \mathsf{F}$.

4.2 Compilation to Extensional Nodes

This section describes three compilation rules that create extensional circuit nodes: UnitPropagate, Independence and ShannonDecomposition.

4.2.1 Unit Propagation

The unit propagation rule applies when the CNF contains a *unit clause*, that is, a clause with a single literal. For example, the unit clause

$$\forall X \in \mathsf{People} : \mathrm{friends}(X, X)$$

expresses that people are friends with themselves. If in addition, the theory contains formulas that mention the same atoms, such as

$$\forall X, Y, X \in \mathsf{People} \wedge Y \in \mathsf{People} : \mathrm{friends}(X, Y) \vee \mathrm{dislikes}(X, Y)$$

the information in the unit clauses can be propagated to the rest of the theory. In this case, we can conclude that the latter formula is always satisfied when $X = Y$.

First, we describe auxiliary operators to *split* a clause w.r.t. a constrained atom and to *condition* a constrained clause on a constrained literal. Then we define the unit propagation compilation rule, which uses splitting and conditioning to remove literals from the theory.

Splitting

Unit propagation requires that all constrained clauses γ in the theory be split w.r.t. a given constrained atom a (Algorithm 2). The purpose of splitting w.r.t. a is to divide a clause into an equivalent set of clauses (Postcondition 4.2) such that for each atom a_γ in each clause, either a_γ is independent from a or is subsumed by it, because it covers a subset of the ground atoms of a (Postcondition 4.1).

The first step of the algorithm involves computing the *most general unifier* (mgu) of two constrained atoms. In `FOL`, this is the most general substitution that, when applied to two atoms, makes them identical. For two constrained atoms, the mgu corresponds to the mgu of the two unconstrained atoms, provided that the conjunction of the two constraint sets and the unifier is satisfiable. Otherwise, we say that the two constrained atoms do not unify. This means they are independent.

Proof outlines. Postcondition 4.1 is achieved at fixpoint, when the "if" statement fails. For Postcondition 4.2, case analysis tells us that $\theta \wedge cs_a$ is either satisfied for an instantiation of its variables or one of its domain constraints is not satisfied.

Algorithm 2 $\textsc{Split}(\gamma, a)$

Function.

if $\exists a_\gamma \in \text{atom}_c(\gamma)$ **such that** $(a \not\perp\!\!\!\perp a_\gamma) \wedge \neg(a \Rightarrow a_\gamma)$ **then**
  $\theta \leftarrow \text{mgu}(a, a_\gamma)$
  let $a = \forall \mathbf{X}_a, cs_a : \alpha$
  let $\gamma = \forall \mathbf{X}_\gamma, cs_\gamma : \phi$
  $\mathbf{X} \leftarrow \mathbf{X}_a \cup \mathbf{X}_\gamma$
  $\gamma_{mgu} \leftarrow (\forall \mathbf{X}, cs_\gamma \wedge \theta \wedge cs_a : \phi)$
  $\Gamma_{rest} \leftarrow \{(\forall \mathbf{X}, cs_\gamma \wedge \neg e : \phi) \mid e \in (\theta \wedge cs_a)\} \setminus \{\gamma\}$
  return $\{\textsc{Split}(\gamma_{mgu}, a)\} \cup \bigcup_{\gamma_{rest} \in \Gamma_{rest}} \textsc{Split}(\gamma_r, a)$
else
  return $\{\gamma\}$ *// clause is invariant to splitting*

Postcondition 4.1. $\forall \gamma, a, \forall a_\gamma \in \text{atom}_c(\textsc{Split}(\gamma, a)) : (a \perp\!\!\!\perp a_\gamma) \vee (a \Rightarrow a_\gamma)$

Postcondition 4.2. $\forall \gamma, a : \gamma \equiv \bigwedge_{\gamma_s \in \textsc{Split}(\gamma,a)} \gamma_s$

The constrained clauses that Split generates each correspond to one of these cases. Therefore, any interpretation that does not satisfy the original clause must also not satisfy one of the split clauses and every model of the split clauses is also a model of the original clause. Conversely, every model of the original clause is a model of the split clauses, because constraints are only added to the constraint set. □

Example 4.1. Splitting $\textsc{Split}(\gamma, a)$ with

$$\gamma = (\forall X, X \neq \mathit{kiwi} \wedge X \in \mathsf{Animal} : \ \mathsf{flies}(X) \vee \neg\, \mathsf{haswings}(X)) \ \text{ and}$$

$$a = (\forall X, X \neq \mathit{penguin} \wedge X \in \mathsf{Bird} : \ \mathsf{flies}(X))$$

results in

$$\gamma_{mgu} = (\forall X, X \neq \mathit{kiwi} \wedge X \neq \mathit{penguin} \wedge X \in \mathsf{Animal} \wedge X \in \mathsf{Bird} : \ \ldots),$$

$$\gamma^1_{rest} = (\mathsf{flies}(\mathit{penguin}) \vee \neg\, \mathsf{haswings}(\mathit{penguin})) \ \text{ and}$$

$$\gamma^2_{rest} = (\forall X, X \neq \mathit{kiwi} \wedge X \in \mathsf{Animal} \wedge X \notin \mathsf{Bird} : \ \mathsf{flies}(X) \vee \ldots).$$

After splitting, clauses γ^1_{rest} and γ^2_{rest} are independent from a, while the constrained atom $(\forall X, X \neq \mathit{kiwi} \wedge X \neq \mathit{penguin} \wedge X \in \mathsf{Animal} \wedge X \in$

Bird : flies(X)) in γ_{mgu} is implied by a.

Example 4.2. Splitting $\textsc{Split}(\gamma, a)$ with

$$\gamma = (\forall X, X \in \mathsf{D} : \mathrm{p}(X) \vee \mathrm{q}(X)) \text{ and}$$

$$a = (\forall X, X \in \mathsf{D} \wedge Y \neq Z : \mathrm{p}(X))$$

results in

$$\gamma_{mgu} = (\forall X, X \in \mathsf{D} \wedge Y \neq Z : \mathrm{p}(X) \vee \mathrm{q}(X)) \text{ and}$$

$$\gamma_{rest} = (\forall X, X \in \mathsf{D} \wedge Y = Z : \mathrm{p}(X) \vee \mathrm{q}(X)).$$

Here, the domain for X does not change. The split is a case analysis of the constraints between Y and Z, which are free variables.

Poole (2003) introduced splitting for parfactors. We apply it to constrained clauses and extend it with set membership constraints and free variables.

Conditioning

Conditioning a constrained clause γ on a constrained literal l removes the clause when it is satisfied (or subsumed) by the literal. Otherwise, it removes those literals from γ that are not satisfied by l. The conditioning procedure is shown in Algorithm 3.

Proof outlines. Postcondition 4.4 follows from the fact that (for "if") the empty set is independent of l and (for "else"), the remaining literals are all independent of l. Postcondition 4.5 holds because (for "if"), both sides of the equivalence evaluate to l, and (for "else"), in any model of $\gamma \wedge l$, the literals removed from γ are unsatisfied. □

Unit Propagation Rule

The unit propagation compilation rule (Algorithm 4) first splits all clauses w.r.t. the unit clause and then uses conditioning to propagate the information in the unit clause. It returns a circuit whose root node is a decomposable extensional conjunction.

Algorithm 3 $\textsc{Condition}(\gamma, l)$

Precondition 4.3. *Clause γ has been split w.r.t. the atom contained in literal l:* $\{\gamma\} = \textsc{Split}(\gamma, \text{atom}_c(l))$.

Function.

if $(l \Rightarrow \gamma)$ **then**
  return $\emptyset$ // *γ is satisfied by propagating l*
else
  let $\gamma = \forall \mathbf{X}, cs : \lambda_1 \vee \cdots \vee \lambda_n$
  $\Lambda \leftarrow \{\lambda_i \mid \neg(l \Rightarrow (\forall \mathbf{X}, cs : \neg\lambda_i))\}$ // *remove unsatisfied literals*
  return $\forall \mathbf{X}, cs : \Lambda$

Postcondition 4.4. $\textsc{Condition}(\gamma, l) \perp\!\!\!\perp l$

Postcondition 4.5. $(\textsc{Condition}(\gamma, l) \wedge l) \equiv (\gamma \wedge l)$

Algorithm 4 $\textsc{UnitPropagate}(\Delta)$

Precondition 4.6. *The theory has a unit clause u:* $\exists u \in \Delta$ *s.t.* $|\text{atom}_c(\gamma)| = 1$.

Function.

$\Delta_{up} \leftarrow \bigcup_{\gamma \in \Delta} \bigcup_{\gamma_s \in \textsc{Split}(\gamma, \text{atom}_c(u))} \textsc{Condition}(\gamma_s, u)$
return $\textsc{Compile}(\Delta_{up}) \wedge \textsc{Compile}(u)$

Postcondition 4.7. *$\Delta_{up} \wedge u$ is decomposable.*

Postcondition 4.8. $\Delta \equiv \Delta_{up} \wedge u$

Proof outlines. Postcondition 4.7 is a direct consequence of Postcondition 4.4. Postcondition 4.8 follows from Postconditions 4.2 and 4.5. □

Example 4.3. Figure 4.1 shows propagation of the unit clause $\forall X, X \in \text{People} : \text{friends}(X, X)$. Shaded rectangles and circles are again FO-NNF nodes. White rectangles show the theories before and after applying the rule. The first two clauses require splitting w.r.t. the unit clause. This creates two copies of

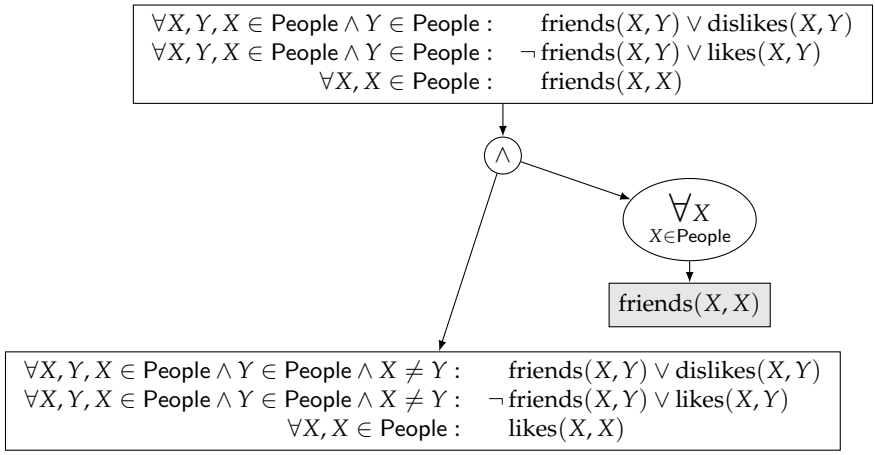


Figure 4.1: Unit propagation of $\forall X, X \in \text{People} : \text{friends}(X, X)$

each clause: one where $X = Y$ and one where $X \neq Y$:

$$\forall X \in \text{People} : \quad \text{friends}(X, X) \vee \text{dislikes}(X, X)$$

$$\forall X, Y \in \text{People} \wedge X \neq Y : \quad \text{friends}(X, Y) \vee \text{dislikes}(X, Y)$$

$$\forall X \in \text{People} : \neg\,\text{friends}(X, X) \vee \text{likes}(X, X)$$

$$\forall X, Y \in \text{People} \wedge X \neq Y : \neg\,\text{friends}(X, Y) \vee \text{likes}(X, Y)$$

Next, the first copy of the first clause is removed, because the unit clause entails (or subsumes) it. 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 new clause $\forall X, X \in \text{People} : \text{likes}(X, X)$. The original unit clause becomes independent of the other clauses and can be split off in a decomposable conjunction. The unit clause is then compiled by other rules into an intensional conjunction and a leaf literal, while the remainder of the theory requires further compilation.

One way to view unit propagation is as follows. It first removes subsumed clauses. Then, for the remaining clauses, it computes their *resolvent* with the unit clause, as is done by the well-known resolution algorithm (Robinson, 1965). Afterwards it decomposes the unit clause and the rest of the theory because they are independent.

4.2.2 Independence

The independence rule (Algorithm 5) applies when the theory consists of two independent sets of clauses.

Algorithm 5 INDEPENDENCE(Δ)

Precondition 4.9. $\Delta = \Delta_1 \cup \Delta_2$ *and* $\Delta_1 \perp\!\!\!\perp \Delta_2$

Function.

1: **return** COMPILE(Δ_1) $\wedge$ COMPILE(Δ_2)

Postcondition 4.10. $\Delta_1 \wedge \Delta_2$ *is decomposable*

Two FOL-DC sentences are independent ($\perp\!\!\!\perp$, Definition 3.18) when their groundings consist of disjoint sets of ground atoms. Two formulas are independent when they are independent for any closing substitution. Independence of formulas can be determined by checking whether the one formula contains a constrained atom that *unifies* with a constrained atom in the other formula.

For example, the two clauses in

$$\forall X, Y \in \mathsf{People} \wedge X \neq Y : \quad \mathrm{friends}(X, Y) \vee \mathrm{dislikes}(X, Y)$$

$$\forall Z \in \mathsf{People} : \neg\, \mathrm{friends}(Z, Z) \vee \mathrm{likes}(Z, Z)$$

are independent. The atom $\forall X, Y \in \mathsf{People} \wedge X \neq Y :\ \mathrm{friends}(X, Y)$ in the first clause does not unify with the atom $\forall Z \in \mathsf{People} :\ \mathrm{friends}(Z, Z)$ in the second clause. The most general unifier of the unconstrained atoms is $[X/Z, Y/Z]$ and the corresponding unified constraint set $(X, Y \in \mathsf{People} \wedge X \neq Y \wedge Z \in \mathsf{People} \wedge X = Z \wedge Y = Z)$ is unsatisfiable, because the constraints $(X \neq Y \wedge X = Z \wedge Y = Z)$ have no solution.

4.2.3 Shannon Decomposition

The *Shannon decomposition* (Shannon, 1949) is an essential tool for propositional automated reasoning. It states that a Boolean function $f(x_1, x_2, \ldots, x_n)$ over propositions $x_1, x_2, \ldots, x_n$ is equivalent to $(x_1 \wedge f(\mathrm{T}, x_2, \ldots, x_n)) \vee (\neg x_1 \wedge f(\mathrm{F}, x_2, \ldots, x_n))$. Shannon decomposition is used in the influential DPLL

algorithm (Davis, Logemann, and Loveland, 1962) for satisfiability checking and in the construction of Binary Decision Diagrams (Bryant, 1986).

The *first-order* Shannon decomposition[1] (Algorithm 6) applies to a theory that contains a literal whose arguments are *constants* or *free variables* (not bound by the enclosing intensional conjunction). The compilation rule returns a circuit whose root node is a deterministic disjunction.

Algorithm 6 ShannonDecomposition(Δ)

Precondition 4.11. *The theory contains an atom without bound logical variable arguments: There is a* $(\forall \boldsymbol{X}, cs : a(\boldsymbol{Y})) \in \text{atom}_c(\Delta)$ *such that* $\boldsymbol{X} \cap \boldsymbol{Y} = \emptyset$.

Function.

1: **return** Compile$(a(\boldsymbol{Y}) \wedge \Delta) \vee$ Compile$(\neg a(\boldsymbol{Y}) \wedge \Delta)$

Postcondition 4.12. $(a(\boldsymbol{Y}) \wedge \Delta) \vee (\neg a(\boldsymbol{Y}) \wedge \Delta)$ *is deterministic.*

Postcondition 4.13. $((a(\boldsymbol{Y}) \wedge \Delta) \vee (\neg a(\boldsymbol{Y}) \wedge \Delta)) \equiv \Delta$

Proof outlines. Postcondition 4.12 holds because the disjuncts contain opposing literals for a. Postcondition 4.13 follows from the application of classical Shannon decomposition for any closing substitution. □

Example 4.4. Figure 4.2 depicts a Shannon decomposition of fun(bob) for a single constrained clause. The intermediate theories are shown in white rectangles. These are the inputs of a subsequent application of UnitPropagate, which results in the circuit shown.

4.3 Shattering: Exposing Symmetries of the Model

The notion of automorphism is central to the definition of FO-da-DNNF circuits. Algorithms that operate on these circuits exploit the symmetries that are guaranteed to exist (e.g., the count function of Section 3.4.3). This guarantee

[1] In Van den Broeck et al. (2011a), Shannon decomposition is a special case of a compilation rule called *Inclusion-Exclusion*. It applies when the theory contains a clause whose literals can be divided into two sets that do not share any bound logical variables or constraints. We do not allow for this rule because it requires a more expressive circuit language (with inclusion-exclusion nodes) and because the operator is not strictly required for the results presented here.

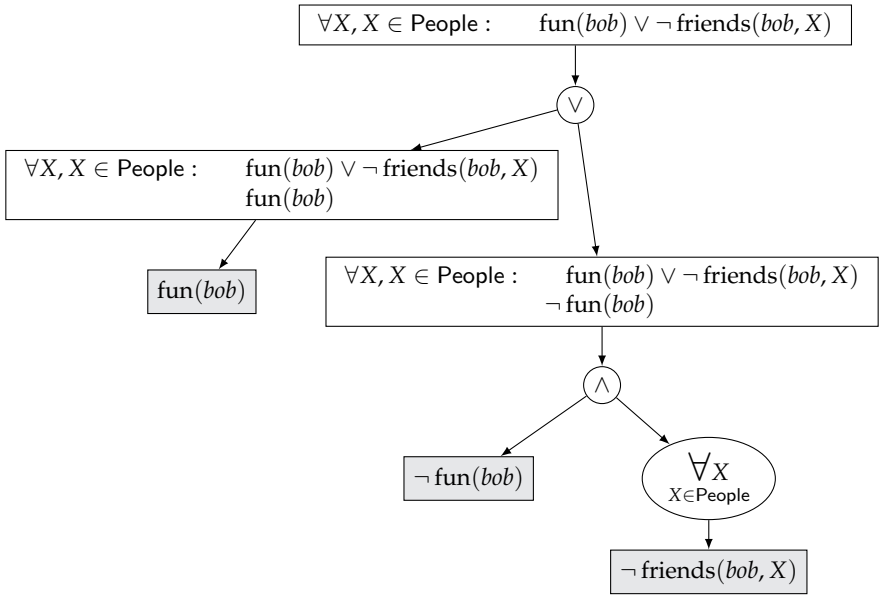


Figure 4.2: Shannon decomposition of $\text{fun}(bob)$

comes from the automorphism property that must hold for each intensional node in the circuit. Even though these symmetries also exist implicitly in the input FO-CNF, they cannot be read off directly from the model.

Example 4.5. Consider the following formula.

$$\forall X, X \in \text{People}: \ \text{fun}(bob) \vee \neg\,\text{friends}(bob, X)$$

The two solutions $[X/alice]$ and $[X/bob]$ of the constraint set yield the sentences $\text{fun}(bob) \vee \neg\,\text{friends}(bob, alice)$ and $\text{fun}(bob) \vee \neg\,\text{friends}(bob, bob)$. These sentences are not equivalent up to a permutation of constants, because there is no bijection that maps the models of one into the models of the other. Therefore, this intensional conjunction is not automorphic. Yet, implicitly, there are many symmetries in this model. Two other solutions of the constraint set, $[X/alice]$ and $[X/charlie]$, yield the sentences $\text{fun}(bob) \vee \neg\,\text{friends}(bob, alice)$ and $\text{fun}(bob) \vee \neg\,\text{friends}(bob, charlie)$ which are equivalent up to the permutation $\pi = \{alice \mapsto charlie, charlie \mapsto alice\}$. We can now write

a theory that makes these symmetries explicit, as follows.

$$\forall X, X \in \mathsf{People} \wedge X \neq bob : \mathrm{fun}(bob) \vee \neg\,\mathrm{friends}(bob, X)$$

$$\forall \emptyset, bob \in \mathsf{People} : \mathrm{fun}(bob) \vee \neg\,\mathrm{friends}(bob, bob)$$

In this theory, both intensional conjunctions are automorphic.

This section describes an algorithm for transforming FOL-DC theories into equivalent theories where the symmetries are explicit in the syntax. This type of transformation is called *shattering* and will be frequently used in the following sections and chapters.

We present two shattering procedures. The first, called *preemptive shattering*, is based on the procedure of the same name given by Poole, Bacchus, and Kisyński (2011). It is meant to be a conceptually simpler version of the influential original *shattering* algorithm proposed by Poole (2003) and de Salvo Braz, Amir, and Roth (2005). For the preemptive shattering algorithm, we go into more detail about the type of symmetries exposed by the algorithm. The second procedure we present is based on the original shattering algorithm and is called *shattering by splitting*. Finally, we define a compilation rule, called ShatteredCompilation, that shatters its input.

Both shattering algorithms were proposed in the exact lifted inference literature, where they were applied to parfactors and where domain constraints are restricted to (in)equalities. Since we work with constrained clauses in FOL-DC, which have set membership constraints, we extend the existing shattering procedures with support for this type of constraints.

4.3.1 Preemptive Shattering

We start by presenting preemptive shattering. Even though it is not the most advanced shattering algorithm, it is powerful enough to prove all theoretical properties in the following chapters, and it is relatively simple. The algorithm operates at three levels: partitioning constants, shattering clauses and shattering theories.

Partition Constants

Preemptive constant shattering is an algorithm that partitions the set of constants in the signature of a sentence. Each element of this partition is defined by a constraint set.

We start with an intuitive description of the algorithm. Preemptive constant shattering of logical variable X returns a set of constraint sets such that all constraint sets have either disjoint or identical solutions. Its inputs are a set of constants and variables (terms) T and a set of domains $\mathcal{D}$. We want to distinguish between all *constants and variables* in T. Each returned constraint set forces the variable X to be equal to one of these terms or different from all of them. Similarly, preemptive constant shattering looks at the partition of constraints induced by the *domains* in $\mathcal{D}$. By adding set membership constraints, all solutions of a returned constraint set are member of the same set of domains. In other words, for each domain, X is forced to be part of it, or to be outside of it.

Algorithm 7 $\textsc{ShatterVar}(X, T, \mathcal{D})$

Input.

X: A logical variable
T: A set of terms (constants and logical variables)
$\mathcal{D}$: A set of domains

Function.

1: $CS_{eq} \leftarrow \{(X = t) \mid t \in T\} \cup \{\bigwedge_{t \in T}(X \neq t)\}$
2: $CS_{in} \leftarrow \left\{ \left(\bigwedge_{\mathsf{D} \in \mathcal{E}}(X \in \mathsf{D}) \wedge \bigwedge_{\mathsf{D} \in (\mathcal{D} \setminus \mathcal{E})}(X \notin \mathsf{D}) \right) \;\middle|\; \mathcal{E} \subseteq \mathcal{D} \text{ s.t. } |\mathcal{E}| > 0 \right\}$
3: **return** $\left\{ cs_{eq} \wedge cs_{in} \;\middle|\; cs_{eq} \in CS_{eq}, cs_{in} \in CS_{in} \right\}$

Postcondition 4.14. $\forall X, T, \mathcal{D} : (\exists \mathsf{D} \in \mathcal{D} : X \in \mathsf{D}) \equiv \bigvee_{cs \in \textsc{ShatterVar}(X,T,\mathcal{D})} cs$

Algorithm 7 describes the shattering procedure more formally. By enforcing $|\mathcal{E}| > 0$ on Line 2, we guarantee that each logical variable is member of at least one domain, which is a requirement for syntactically valid constraint sets. Many of the generated constraints $cs_{eq} \wedge cs_{in}$ on Line 3 will have no solutions and can be dropped. Postcondition 4.14 states that shattering partitions the solutions to the variable X. For any solution to X (in the domains $\mathcal{D}$), at least one of the returned constraint sets is satisfied.

Shattering Clauses

Preemptive shattering of a clause applies preemptive constant shattering to each bound variable in the clause. In addition, it enforces inequality constraints between each pair of bound variables that appear in the same literal. When two arguments X_i and X_j of some input literal can take on the same value, preemptive shattering splits the constrained clause into two: one with the constraint $X_i = X_j$ and another with $X_i \neq X_j$.

Algorithm 8 $\textsc{ShatterClause}(\gamma, T, \mathcal{D})$

Input.

γ: A constrained clause $\forall \mathbf{Y}, cs_\gamma : l_1(\mathbf{Z}_1) \vee \cdots \vee l_m(\mathbf{Z}_m)$
T: A set of terms (constants and logical variables)
$\mathcal{D}$: A set of domains

Supporting Functions.

$\textsc{Partition}(.)$ Generates a set of all possible partitions of the given set into non-empty subsets

Function.

1: $\mathbf{Z} \leftarrow \mathbf{Z}_1 \cup \cdots \cup \mathbf{Z}_m$ *// Shatter individual variables*
2: $CS_A = \{(c_1 \wedge \cdots \wedge c_n) \mid (c_1, \ldots, c_n) \in \bigtimes_{Z \in \mathbf{Z}} \textsc{ShatterVar}(Z, T, \mathcal{D})\}$
// Generate constraints between bound logical variables
3: **for** $i = 1$ to m **do**
4: $\mathbf{B}_i \leftarrow \mathbf{Y} \cap \mathbf{Z}_i$ *// Bound variable arguments of the ith literal*
5: $CS_B^i \leftarrow \emptyset$
6: **for each** $\mathcal{P}_i \in \textsc{Partition}(\mathbf{B}_i)$ **do**
// Variables in the same subset are equal
7: $cs_= \leftarrow \bigwedge_{\mathbf{E} \in \mathcal{P}_i} \bigwedge_{X_1, X_2 \in \mathbf{E}} (X_1 = X_2)$
// Variables in different subsets are not equal
8: $cs_\neq \leftarrow \bigwedge_{\mathbf{E} \in \mathcal{P}_i} \bigwedge_{X_1 \in \mathbf{E}} \bigwedge_{X_2 \in \mathbf{B}_i \setminus \mathbf{E}} (X_1 \neq X_2)$
9: $CS_B^i \leftarrow CS_B^i \cup \{cs_= \wedge cs_\neq\}$
10: $CS_B \leftarrow \{(c_1 \wedge \cdots \wedge c_m) \mid (c_1, \ldots, c_m) \in \bigtimes_{i=1}^m CS_B^i\}$
11: $CS \leftarrow \{(cs_\gamma \wedge cs_a \wedge cs_b) \mid cs_a \in CS_A, cs_b \in CS_B\}$
12: **return** $\{(\forall \mathbf{Y}, cs : l_1(\mathbf{Z}_1) \vee \cdots \vee l_m(\mathbf{Z}_m)) \mid cs \in CS\}$

Postcondition 4.15. $\forall \gamma, T, \mathcal{D} : \gamma \equiv \textsc{ShatterClause}(\gamma, T, \mathcal{D})$

The preemptive clause shattering procedure is given more formally by Algorithm 8. In Lines 2 and 10, the $\bigtimes$ operator represents the Cartesian product of its arguments. The (in)equality constraints between bound logical variables are

generated by enumerating all partitions of the logical variables into non-empty subsets. This corresponds to enumerating all possible equivalence relations on these variables. Postcondition 4.15 states that the output theory is equivalent to the input clause.

Shattering Theories

Finally, we extend the definition of preemptive clause shattering to a general shattering procedure that operates on a FO-CNF in Algorithm 9.

In this algorithm, the set of terms T to distinguish between is not input to the procedure. It is defined to be the union of two other sets. The first set consists of the constants that appear as logical terms in the theory. The second set consists of the free logical variables in the theory. In both cases, these terms can appear as arguments of atoms or in a constraint set. The procedure also initializes the set of domain terms $\mathcal{D}$ to the set of domain terms in the theory. These can either be free domain variables or sets of constants. Given these two sets as input, the procedure then shatters each clause in the FO-CNF separately. We implicitly perform the necessary renamings to avoid clashes between the same logical variable symbols appearing in multiple formulas.

Algorithm 9 $\textsc{Shatter}(\Delta)$

Function.

1: **let** K be the set of constants appearing as logical terms in Δ
2: **let** V be the set of free logical variables in Δ
3: **let** $\mathcal{D}$ be the set of domain terms in Δ
4: **return** $\{\textsc{ShatterClause}(\gamma, K \cup V, \mathcal{D}) \mid \gamma \in \Delta\}$

Postcondition 4.16. *Bound logical variables in* $\textsc{Shatter}(\Delta)$ *have identical or disjoint sets of solutions.*

Postcondition 4.17. $\forall \Delta, \forall a_1, a_2 \in \text{atom}_c(\textsc{Shatter}(\Delta)) : (a_1 \perp\!\!\!\perp a_2) \vee (a_1 \equiv a_2)$

Postcondition 4.18. *Intensional conjunctions in* $\textsc{Shatter}(\Delta)$ *are automorphic.*

Proof of Postcondition 4.16. This property follows from the fact that the solutions for any variable X are either singletons, or a set of constants that do not appear as logical terms in the theory (Line 1 of Algorithm 7). In the latter case, the sets

of solutions correspond to an element of the partition induced by the domain terms in the theory (Line 2 of Algorithm 7). □

The following section presents a proof of Postcondition 4.18, where it is restated as Theorem 4.4.

Example 4.6. Consider the FO-CNF with two clauses

$$\forall X, Y \in \mathsf{People} : \neg\,\mathrm{smokes}(X) \vee \neg\,\mathrm{friends}(X, Y) \vee \mathrm{smokes}(Y)$$

$$\mathrm{smokes}(\mathit{alice})$$

which has $K = \{\mathit{alice}\}$, $V = \emptyset$ and $\mathcal{D} = \{\mathsf{People}\}$. Preemptive shattering of the variable X returns $\{(X \in \mathsf{People} \wedge X = a), (X \in \mathsf{People} \wedge X \neq a)\}$. Preemptive shattering of the first clause returns

$$\forall X, Y \in \mathsf{People} \wedge X = a \wedge Y = a : \neg\,\mathrm{smokes}(X) \vee \ldots$$

$$\forall X, Y \in \mathsf{People} \wedge X = a \wedge Y \neq a : \neg\,\mathrm{smokes}(X) \vee \ldots$$

$$\forall X, Y \in \mathsf{People} \wedge X \neq a \wedge Y = a : \neg\,\mathrm{smokes}(X) \vee \ldots$$

$$\forall X, Y \in \mathsf{People} \wedge X \neq a \wedge Y \neq a \ \wedge X = Y : \neg\,\mathrm{smokes}(X) \vee \ldots$$

$$\forall X, Y \in \mathsf{People} \wedge X \neq a \wedge Y \neq a \ \wedge X \neq Y : \neg\,\mathrm{smokes}(X) \vee \ldots$$

The second clause does not change with preemptive shattering.

Preemptive constant shattering can be implemented in time that is exponential in the number of domain variables and logical variables per clause, and polynomial in the number of constants and free variables in Δ.

4.3.2 Automorphisms Introduced by Shattering

The purpose of shattering is to make the symmetries of the given theory explicit in its syntax. We will now precisely show how preemptive constant shattering creates automorphisms.

Automorphism for One Variable

The input theory consists of intensional conjunctions that quantify over logical variables. Let us first assume that the number of logical variables quantified over is exactly one. For this type of intensional node, automorphism was defined as follows: $(\forall X, cs : \phi)$ is automorphic iff for all solutions $[X/c], [X/c'] \in$ solutions(cs, X), we have that $\phi[X/c]$ and $\phi[X/c']$ are equivalent up to a permutation of constants (Definition 3.26).

Two sentences ϕ and ψ were in turn defined to be equivalent up to a permutation of constants C when there exists a permutation $\pi : C \mapsto C$ which turns the set of models of ϕ into the one of ψ (Definition 3.25). Hence in order to prove automorphism, we first need to prove the following.

Lemma 4.1. *In a shattered theory, two conjuncts $\phi[X/c]$ and $\phi[X/c']$ of an intensional conjunction ϕ, are equivalent up to the permutation of constants $\pi = \{c \mapsto c', c' \mapsto c\}$.*

We prove Lemma 4.1 by proving a stronger type of equivalence: there exists a permutation of constants π that *syntactically* maps ϕ into ψ. Syntactic equivalence up to a permutation of constants is a sufficient but not necessary condition for equivalence up to a permutation of constants (for example, two syntactically different unsatisfiable theories are also equivalent).

We will now prove that $\phi[X/c]$ and $\phi[X/c']$ are syntactically equivalent up to the permutation $\pi = \{c \mapsto c', c' \mapsto c\}$. To syntactically map $\phi[X/c]$ into $\phi[X/c']$ with π, we need to show that the following *objectives* are achieved:

1. π indeed maps all positions of c where X occurred in ϕ into c', and
2. π does not modify the syntax of ϕ elsewhere.

If we apply π to the sentence $\phi[X/c]$, the modifications to the theory occur in three *positions* in the sentence:

1. where the constant c appears as a logical term (e.g., in $Y \neq c$ or $\mathrm{p}(c, Y)$),
2. where the constant c' appears as a logical term, and
3. where c and c' occur in a domain term (e.g., in $Y \in \{c, c'\}$).

The first position accounts for the first objective: to substitute c by c' where X occurred. We will now prove the second objective, that everything else remains unchanged.

First, assume that the constraint set has at least two solutions for X. Otherwise, the intensional conjunction is trivially automorphic. This means that cs contains the inequality constraints $\bigwedge_{t \in T}(X \neq t)$ from Line 1 of Algorithm 7. Otherwise, the constraint set would contain a constraint of the form $X = t$ and the solution set would have size zero or one. These inequality constraints enforce that solutions(cs, X) contains only constants from the domain that do not occur in the theory as logical terms. These constants only occur in the domains (sets of constants). Hence, the first position coincides with those positions where X occurred in ϕ and replacing c by c' here is intended by the first objective. Furthermore, the second position does not exist: c' does not occur as a logical term in the theory.

Finally, we need to prove that applying the permutation π in the third position, inside domain terms, does not modify the theory. This follows from the domain constraints constructed in Line 2 of Algorithm 7. They express that for any domain term D in the theory, all solutions(cs, X) are member of the domain, or none are. Applying the permutation π to a set of constants that contains both c and c' does not alter the set. Therefore, modifications in the third position do not change the theory. This proves Lemma 4.1 and leads us to the following result.

Theorem 4.2. *After preemptive constant shattering, every intensional conjunction over a single logical variable is automorphic.*

Automorphism for Multiple Variables

We now consider the case where the intensional conjunction $(\forall \mathbf{X}, cs : \phi)$ is over a set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. In this case, to prove automorphism, we need to show that for all solutions $[X_1/c_1, \ldots, X_n/c_n], [X_1/c'_1, \ldots, X_n/c'_n] \in$ solutions$(cs, \mathbf{X})$, the following holds.

Lemma 4.3. *In a shattered theory, the two conjuncts $\phi[X_1/c_1, \ldots, X_n/c_n]$ and $\phi[X_1/c'_1, \ldots, X_n/c'_n]$ of an intensional conjunction ϕ, are equivalent up to a permutation of constants. This permutation is $\pi = \{c_1 \mapsto c'_1, c'_1 \mapsto c_1, \ldots, c_n \mapsto c'_n, c'_n \mapsto c_n\}$.*

The argumentation is largely similar to the proof of Lemma 4.1:

- The constants c_i and c'_i do not appear as logical terms in ϕ.
- Where c_i and c'_i appear inside domain terms, they occur together, so that π does not change domain terms.

The main difference is in the requirement that π is a bijection. This requirement is only true when all constants c_i and c'_i are distinct. Otherwise, π might map a single constant in the first solution onto two constants in the second solution and π is not a bijection. If π is a bijection, it also achieves the first objective: it maps every c_i in a position where an X_i occurred in ϕ into a c'_i.

The property that all c_i and c'_i are distinct is enforced by having an inequality constraint between all X_i. These constraints are added on Line 8 of Algorithm 8. Hence, we have the following result.

Theorem 4.4. *After preemptive constant shattering, every intensional conjunction* $(\forall \boldsymbol{X}, cs : \phi)$ *is automorphic.*

From another point of view, the set of constrained atoms in a shattered theory represents a partition of atoms. That is, the groundings of constrained atoms are guaranteed to be disjoint and cover the set of all ground atoms (Postcondition 4.17). The atoms that are grouped together are those that can be mapped onto each other by a permutation of constants, as shown by Lemma 4.3.

4.3.3 Shattering by Splitting

The second shattering algorithm is called *shattering by splitting* and is based on the Split function presented in Section 4.2.1. It recursively applies splitting to all clauses in the theory w.r.t. some constrained atom in the same theory, until convergence. We present it here because of its significance in the lifted probabilistic inference literature (Poole, 2003; de Salvo Braz, Amir, and Roth, 2005).

Algorithm 10 SHATTERBYSPLIT(Δ)

Function.

```
if ∃γ ∈ Δ, a ∈ atom_c(Δ) such that SPLIT(γ, a) ≠ {γ} then
    return SHATTERBYSPLIT(⋃_{γ∈Δ} SPLIT(γ, a))        // Split recursively
else
    return Δ                                          // Reached fixpoint of splitting
```

Postcondition 4.19. $\forall \Delta, \forall a_1, a_2 \in \text{atom}_c(\textsc{ShatterBySplit}(\Delta)) : (a_1 \perp\!\!\!\perp a_2) \vee (a_1 \equiv a_2)$

Postcondition 4.20. $\forall \Delta : \textsc{ShatterBySplit}(\Delta) \equiv \Delta$

Proof outlines. Postcondition 4.19 follows from applying Postcondition 4.1 in two directions. Postcondition 4.20 follows from Postcondition 4.2 and the observation that shattering merely applies the splitting operator multiple times. □

Postcondition 4.19 states that after shattering, all constrained atom groundings are either disjoint or identical. The specific assignments to the logical variables cannot be distinguished any further. This property also holds for the preemptive shattering procedure (Postcondition 4.17). In a sense, shattering by splitting is a weaker form of shattering, as is shown in the following proposition.

Proposition 4.5. *A theory that is preemptively shattered is also shattered by splitting* ($\forall \Delta : \textsc{Shatter}(\Delta) = \textsc{ShatterBySplit}(\textsc{Shatter}(\Delta))$*, but a theory that is shattered by splitting is not necessarily preemptively shattered* ($\exists \Delta : \textsc{ShatterBySplit}(\Delta) \neq \textsc{Shatter}(\textsc{ShatterBySplit}(\Delta))$).

Example 4.7. The formula $\forall X, X \in \mathsf{People} : \neg\,\mathrm{smokes}(X) \vee \mathrm{friends}(X, \mathit{alice})$ is shattered by splitting (no two atoms unify). However, preemptive shattering results in two clauses: $\forall X, X \in \mathsf{People} \wedge X \neq \mathit{alice} : \neg\,\mathrm{smokes}(X) \vee \mathrm{friends}(X, \mathit{alice})$ and $\forall \emptyset, \mathit{alice} \in \mathsf{People} : \neg\,\mathrm{smokes}(\mathit{alice}) \vee \mathrm{friends}(\mathit{alice}, \mathit{alice})$.

Despite this difference in the size of the output of the two shattering algorithms, their worst-case complexity is the same: exponential in the number of domain variables and logical variables per clause, and polynomial in the number of constants and free variables in Δ.

Both the compilation algorithm presented in this chapter and the definition of the `FO-da-DNNF` circuit language can be extended to work with shattering by

splitting instead of preemptive shattering. The disadvantage of the 'shattering by splitting' algorithm, however, is that it is then still not clear which symmetries it exposes exactly. They certainly do not correspond to the notion of automorphism used here, or to equivalence up to a permutation of constants. The symmetries introduced instead correspond to a permutation of ground atoms, which complicates the analysis of the compilation algorithm and algorithms that operate on FO-da-DNNF circuits.

Example 4.8. The formula $\forall X, X \in \mathsf{D} : \neg\, \mathrm{p}(X) \vee \mathrm{q}(a)$ is shattered by splitting, but the intensional conjunction is not automorphic. Consider two conjuncts $\neg\, \mathrm{p}(a) \vee \mathrm{q}(a)$ and $\neg\, \mathrm{p}(b) \vee \mathrm{q}(a)$. There is no permutation of constants that maps the first into the second. There does exist a permutation of atoms that achieves this, namely $\pi = \{\mathrm{p}(a) \mapsto \mathrm{p}(b)\}$. The notion of automorphism can be extended to work with such permutations instead of permutations of constants.

For these reasons we choose to work with preemptive shattering in this dissertation. Yet, we believe shattering by splitting is an interesting topic of research and better understanding it is essential to building efficient lifted inference algorithms.

4.3.4 Shattered Compilation

The *shattered compilation* rule (Algorithm 11) does not itself introduce new circuit nodes. It merely modifies the theory to enable other rules. The theory being shattered is a precondition of the compilation rules in Section 4.4. Furthermore, shattering might introduce new opportunities for applying the independence and Shannon decomposition rules. Because we only shatter in this stage of the compilation process, the operators presented above use what is called *splitting as needed* (Kisyński and Poole, 2009a).

Algorithm 11 ShatteredCompilation(Δ)

Precondition 4.21. Shatter$(\Delta) \neq \Delta$

Function.

1: **return** Compile(Shatter(Δ))

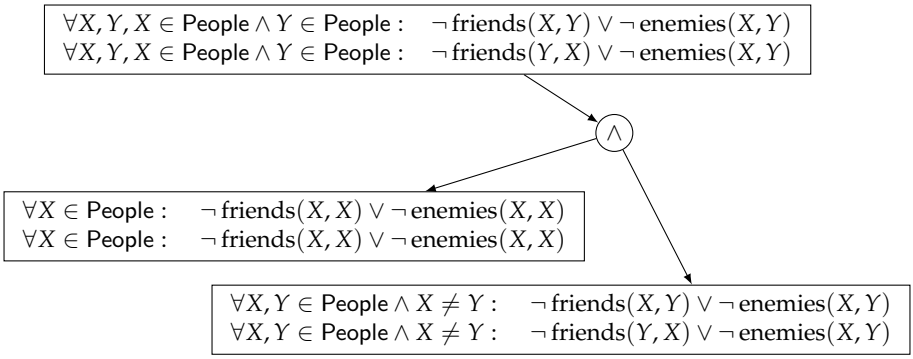


Figure 4.3: Shattered compilation

Example 4.9. The formula

$$\forall X, Y \in \text{People} : \text{enemies}(X, Y) \Rightarrow \neg\, \text{friends}(X, Y) \wedge \neg\, \text{friends}(Y, X), \tag{4.1}$$

has the following representation in FO-CNF:

$$\forall X, Y, X \in \text{People} \wedge Y \in \text{People} : \neg\, \text{friends}(X, Y) \vee \neg\, \text{enemies}(X, Y)$$

$$\forall X, Y, X \in \text{People} \wedge Y \in \text{People} : \neg\, \text{friends}(Y, X) \vee \neg\, \text{enemies}(X, Y)$$

Shattering this theory splits each formula into two cases, one where $X = Y$ and one where $X \neq Y$. After shattering, these split clauses contain no unifying atoms anymore and have become independent. Figure 4.3 depicts this example of shattered compilation, followed by the independence compilation rule (Algorithm 5, p. 91).

4.4 Compilation to Intensional Nodes

This section describes the following compilation rules that introduce intensional, or first-order, circuit nodes: VacuousConjunction, IndependentSingleGroundings, IndependentPairedGroundings, and AtomCounting.

4.4.1 Vacuous Conjunction

The first intensional compilation rule, called *vacuous conjunction* (Algorithm 12), applies when the input theory Δ consists of a single constrained clause (intensional conjunction) $\forall \emptyset, cs : \phi$ without any bound variables. This theory is equivalent to ϕ when the constraints cs are satisfied, and to true otherwise. Hence, this type of intensional conjunction cannot be ignored without changing the semantics of the formula. The rule compiles the constrained clause by removing the intensional conjunction and its constraint set from the theory and adding them as circuit nodes.

Algorithm 12 VacuousConjunction(Δ)

Precondition 4.22. $\Delta = \{(\forall \emptyset, cs : \phi)\}$

Function.

1: **return** $\forall \emptyset, cs :$ Compile(ϕ)

Postcondition 4.23. *The returned intensional conjunction is decomposable.*

Postcondition 4.24. *The returned intensional conjunction is automorphic.*

Proof outlines. Postcondition 4.23 and Postcondition 4.24 are trivially satisfied because the intensional conjunction has zero or one solutions. □

4.4.2 Logical Variable Properties

To formally define the operators we propose next, and prove their correctness, we first introduce some mathematical concepts related to the bound logical variables in a theory (partly after Jha et al. (2010)).

Definition 4.2 (Unifying Variables)**.** Two bound logical variables X and Y are *directly unifying*[2] if they are equated by unifying a pair of atoms in the theory. The *unifying* relationship is the transitive closure of the directly unifying relation.

[2]Unifying is called *binding* in Van den Broeck (2011b). We change terminology here to avoid confusion with the concept of free and bound logical variables.

Example 4.10. In the theory

$$\forall X, Y, X \in \mathsf{D} \wedge W \in \mathsf{D} : \neg\, \mathrm{p}(W, X) \vee \neg\, \mathrm{q}(X)$$

$$\forall Y, Y \in \mathsf{D} : \quad \mathrm{r}(Y) \vee \neg\, \mathrm{q}(Y)$$

$$\forall Z, Z \in \mathsf{D} : \neg\, \mathrm{r}(Z) \vee \mathrm{s}(Z)$$

the variable pairs (X, Y) and (Y, Z) are directly unifying. The variables X, Y and Z are unifying. Variable W does not unify with any other variable. Note that the unifying relationship is an equivalence relation that defines two equivalence classes: $\{X, Y, Z\}$ and $\{W\}$.

Lemma 4.6 (Unifying Domains)**.** *After shattering, unifying logical variables have identical solutions.*

Proof. Shattering constrains the solutions of two logical variables to be identical or disjoint (Postcondition 4.16). When two variables unify, their solutions cannot be disjoint. □

Definition 4.3 (Root Unifying Class)**.** A *root variable* is a variable that appears in all the atoms in its clause. A *root unifying class* is an equivalence class of unifying variables where all variables are root.

Example 4.11. In the theory of Example 4.10, $\{X, Y, Z\}$ is a root unifying class and $\{W\}$ is not.

We now present two compilation rules that generate automorphic and decomposable intensional conjunctions. In the definition of these rules, **U** will denote a root unifying class. Lemma 4.6 states that all variables in **U** have identical solutions after shattering. We will represent these solutions for some variable X by the constraint set $cs_{\mathbf{U}}(X)$.

4.4.3 Independent Single Groundings

Let us first consider the case where the root unifying class contains exactly one logical variable per clause, as in the theory

$$\forall X, Y, X \in \mathsf{People} \wedge Y \in \mathsf{People} \wedge X \neq Y : \mathrm{dislikes}(X, Y) \vee \mathrm{friends}(X, Y)$$

$$\forall X, Y, X \in \mathsf{People} \wedge Y \in \mathsf{People} \wedge X \neq Y : \mathrm{fun}(X) \vee \neg\, \mathrm{friends}(X, Y).$$

Here, the X variables from both clauses together form a root unifying class. Independent single groundings replaces the quantifiers for these variables by a single intensional conjunction:

$$\forall X \in \mathsf{People} : \left[\begin{array}{r} \forall Y \in \mathsf{People} \wedge X \neq Y : \mathrm{dislikes}(X,Y) \vee \mathrm{friends}(X,Y) \\ \wedge\, \forall Y \in \mathsf{People} \wedge X \neq Y : \mathrm{fun}(X) \vee \neg\, \mathrm{friends}(X,Y) \end{array} \right]$$

It then removes this conjunction from the theory to be compiled and adds it to the output circuit. We will look at this example in more detail in Figure 4.4 and show the entire compiled circuit in Figure 4.5.

If after shattering, the theory contains a root unifying class $\mathbf{U}$ with one variable per formula, this technique can be applied. The solutions of each variable in $\mathbf{U}$ are identical and each clause is grounded w.r.t. the same set of constants $\mathrm{solutions}(cs_{\mathbf{U}}(Y), Y)$. Substituting all these solutions in a naive way would create an extensional decomposable conjunction over $|\mathrm{solutions}(cs_{\mathbf{U}}(Y), Y)|$ subcircuits, which is a potentially very large number. However, because the subcircuits are equivalent up to a renaming of constants, the theory is more succinctly represented by compiling only one child theory Δ' and enclosing it in a decomposable automorphic intensional conjunction. The independent single groundings rule[3] is more formally described in Algorithm 13.

Algorithm 13 IndependentSingleGroundings(Δ)

Precondition 4.25. *The theory is shattered:* $\textsc{Shatter}(\Delta) = \Delta$.

Precondition 4.26. *There exists a root unifying class* $\mathbf{U}$ *with one variable per clause:* $\forall \gamma \in \Delta : |\mathrm{bvars}(\gamma) \cap \mathbf{U}| = 1$.

Function.

1: **let** Y be a new logical variable
2: $\theta \leftarrow [\mathbf{U}/Y]$ // *A substitution from the variables in* $\mathbf{U}$ *to* Y
3: $\Delta' \leftarrow \{(\forall(\mathbf{X} \setminus \mathbf{U}), cs : \phi)\theta \mid (\forall \mathbf{X}, cs : \phi) \in \Delta\}$
4: **return** $(\forall Y, cs_{\mathbf{U}}(Y) : \textsc{Compile}(\Delta'))$

Postcondition 4.27. *The returned intensional conjunction is decomposable.*

Postcondition 4.28. *The returned intensional conjunction is automorphic.*

Postcondition 4.29 (Equivalence). $\Delta \equiv (\forall Y, cs_{\mathbf{U}}(Y) : \Delta')$

[3] This operator is called *independent partial grounding* in Van den Broeck et al. (2011a).

Proof of Postcondition 4.27. Precondition 4.26 guarantees that every literal in the theory contains exactly one variable from $\mathbf{U}$. These variables must appear in the same positions of the argument lists of two unifying atoms. Consequently, any pair of atoms that previously unified, becomes independent after substituting a different constant in this position. Hence, for all distinct $\theta_1, \theta_2 \in \text{solutions}(cs_{\mathbf{U}}(Y), Y)$, we have that $\Delta'\theta_1 \perp\!\!\!\perp \Delta'\theta_2$. □

Proof of Postcondition 4.28. It follows from Precondition 4.25 and Theorem 4.2 that $(\forall Y, cs_{\mathbf{U}}(Y) : \Delta')$ is automorphic. □

Proof outline of Postcondition 4.29. Independent single groundings replaces a conjunction of universally quantified formulas by a universally quantified conjunction, which is equivalent. Furthermore, it associates the constraint set $cs_{\mathbf{U}}(Y)$ with this new quantifier. Looking at the reduction semantics of FOL-DC (Equation 3.4), this corresponds to applying the "distributivity of disjunction over conjunction" property to an equivalent FOL theory. □

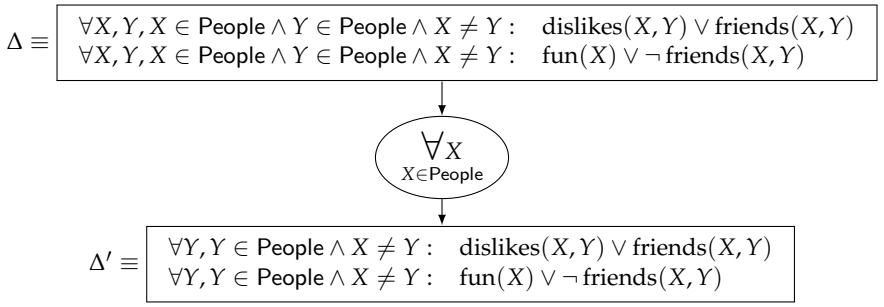


Figure 4.4: Independent Single Grounding

Example 4.12. Figure 4.4 shows a theory Δ that is shattered and has a root unifying class containing the X variable from both clauses. When these variables are grounded to different constants ($\Delta[X/alice]$ and $\Delta[X/bob]$), the different groundings are independent, which means they form a decomposable conjunction. This naive approach would generate $|\text{People}|$ subcircuits. We can do better by observing that the grounded theories are identical up to a renaming of the constants, and so are the circuits. Therefore, it suffices to compile a single theory Δ' where the X variables are free and represent a single constant from the domain People.

The advantage of using an intensional node instead of grounding the variables in $\mathbf{U}$ is that we can also use it to compile formulas with free variables in $cs_{\mathbf{U}}(Y)$, for example when the domain of the logical variable is not specified. With free variables in $cs_{\mathbf{U}}(Y)$, we cannot compute solutions$(cs_{\mathbf{U}}(Y), Y)$ and compile an extensional conjunction as in the naive approach above. Compiling into an intensional conjunction has the additional advantage that the compiled FO-da-DNNF size is independent of $|\text{solutions}(cs_{\mathbf{U}}(Y), Y)|$.

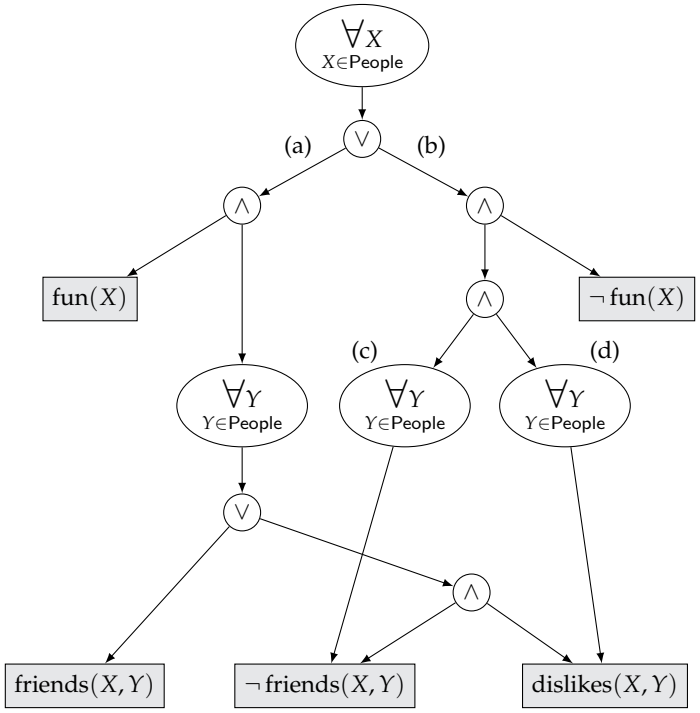


Figure 4.5: Circuit after continuing compilation of Figure 4.4

Example 4.13. Independent single grounding does not simplify the theory, except for removing quantifiers. By doing that, however, it creates new opportunities for other rules. For example, Δ' of Figure 4.4 contains an atom $\text{fun}(X)$, which contains no bound logical variables and therefore allows for Shannon decomposition. This is shown in Figure 4.5. In the branch where $\text{fun}(X)$ is false (labeled with (b)), subsequent steps of unit propagation simplify the theory in a decomposable conjunction of constrained literals. The other branch (labeled with (a)) requires one more step of independent single grounding followed by Shannon decomposition.

An important use of independent single grounding is the compilation of constrained literals. When the theory consists of a single clause with a single literal, every variable is root, and each variable is in its own root unifying class.

Example 4.14. The circuits rooted in nodes (c) and (d) of Figure 4.5 are examples of how independent single grounding compiles constrained literals. Node (c), represents the formula $\forall Y, Y \in \mathsf{People} : \neg\, \mathrm{friends}(X, Y)$. The set $\{Y\}$ is a root unifying class here. Similarly for node (d), $\forall Y, Y \in \mathsf{People} : \mathrm{dislikes}(X, Y)$ is a constrained literal, which guarantees that all variables are root, and independent single grounding applies.

4.4.4 Independent Paired Groundings

When a logical theory contains symmetric, anti-symmetric or total relations, such as

$$\forall X, Y \in \mathsf{People} \wedge X \neq Y : \ \mathrm{friends}(X, Y) \Rightarrow \mathrm{friends}(Y, X), \tag{4.2}$$

$$\forall X, Y \in \mathsf{People} \wedge X \neq Y : \ \mathrm{parent}(X, Y) \Rightarrow \neg\, \mathrm{parent}(Y, X), \tag{4.3}$$

$$\forall X, Y \in \mathbb{Z}_n : \ \leq (\mathrm{X}, \mathrm{Y}) \vee \leq (\mathrm{Y}, \mathrm{X}), \tag{4.4}$$

or more general formulas, such as Formula 4.1, none of the previously presented rules apply. Intuitively, the underlying problem is the presence of either:

- Two unifying (not independent) atoms in the same clause which contain the same logical variable in different positions of the argument list. Examples are (the FO-CNF of) Formulas 4.2, 4.3 and 4.4 above, where the X and Y variable are unified when unifying two atoms from the same clause.
- Two logical variables that unify when unifying one pair of atoms but appear in different positions of the argument list of two other unifying atoms. An examples is Formula 4.1, which in FO-CNF is

$$\forall X, Y, X \in \mathsf{People} \wedge Y \in \mathsf{People} : \neg\, \mathrm{friends}(X, Y) \vee \neg\, \mathrm{enemies}(X, Y)$$

$$\forall X, Y, X \in \mathsf{People} \wedge Y \in \mathsf{People} : \neg\, \mathrm{friends}(Y, X) \vee \neg\, \mathrm{enemies}(X, Y)$$

Here, unifying the enemies(X, Y) atoms unifies the X variables from both clauses, which appear in different positions of the argument lists of the unifying atoms friends(X, Y) and friends(Y, X).

Both of these properties preclude the use of the independent single groundings rule. What they have in common is the presence of a root unifying class with more than one variable per clause. In Formulas 4.2, 4.3 and 4.4 the class is $\mathbf{U} = \{X, Y\}$. In the case of Formula 4.1, the class contains all four variables (two times X and Y).

The *independent paired groundings* compilation rule[4] (Algorithm 14) applies in these cases, where the root unifying class contains exactly two logical variables per clause. For example, Formula 4.2 can equivalently be written as

$$\forall X, Y \in \mathsf{People} \wedge X < Y : \left[\begin{array}{l} (\neg\, \mathrm{friends}(X, Y) \vee \mathrm{friends}(Y, X)) \\ \quad \wedge (\neg\, \mathrm{friends}(Y, X) \vee \mathrm{friends}(X, Y)) \end{array} \right]$$

where the clause is split in two: (i) a copy where $X < Y$, and (ii) a copy where initially $X > Y$, but where the X and Y variables were syntactically swapped, so that also $X < Y$. Recall that $<$ is interpreted according to the natural order of the constants in the signature of `FOL-DC`. The disjunction of $X < Y$ and $X > Y$ is equivalent to $X \neq Y$. These two copies are denoted by Δ_{YZ} and Δ_{ZY} in Algorithm 14. Their conjunction is equivalent to the original clause. The intensional conjunction over these variables can then be taken out of the theory to be compiled and added to the output circuit.

Proof of Postcondition 4.32. We need to show that for all distinct $[Z/c_Z, Y/c_Y]$ and $[Z/c'_Z, Y/c'_Y]$ in solutions$((cs_{\mathbf{U}}(Y) \wedge cs_{\mathbf{U}}(Z) \wedge (Y < Z)), \{Z, Y\})$, it holds that $\Delta'[Z/c_Z, Y/c_Y] \perp\!\!\!\perp \Delta'[Z/c'_Z, Y/c'_Y]$. Because of Precondition 4.31, all atoms have the arguments Y and Z. Assume now that there are two atoms in Δ' that unify. Because of the definition of $\mathbf{U}$, they must either (i) unify Z with Z and Y with Y or (ii) Y with Z. In case (i), after applying the substitutions, this leads to unifications $c_Z = c'_Z$ and $c_Y = c'_Y$, which is impossible because the solutions are distinct. In case (ii), this leads to unifications $c_Z = c'_Y$ and $c_Y = c'_Z$,

[4] Independent paired grounding is based on the *domain recursion* rule of Van den Broeck (2011b) and applies to the same theories. The advantage of independent paired grounding is that it does not require domain recursion nodes in the `FO-da-DNNF` circuit language. It uses intensional conjunction over pairs of variables instead. Furthermore, model counting of these intensional conjunctions is more efficient than model counting of a domain recursion node.

Algorithm 14 IndependentPairedGroundings(Δ)

Precondition 4.30. *The theory is shattered:* Shatter(Δ) $= \Delta$.

Precondition 4.31. *There exists a root unifying class* $\mathbf{U}$ *with two variables per clause:* $\forall \gamma \in \Delta : |\text{bvars}(\gamma) \cap \mathbf{U}| = 2$.

Function.

1: **let** Y, Z be new logical variables
2: **let** $\mathbf{U}_a$ be a subset of $\mathbf{U}$ s.t. each clause has exactly one variable from $\mathbf{U}_a$
3: $\mathbf{U}_b \leftarrow \mathbf{U} \setminus \mathbf{U}_a$
4: $\theta^{YZ} \leftarrow [\mathbf{U}_a/Y, \mathbf{U}_b/Z]$
5: $\theta^{ZY} \leftarrow [\mathbf{U}_a/Z, \mathbf{U}_b/Y]$
6: $\Delta_{YZ} \leftarrow \{(\forall(\mathbf{X} \setminus \mathbf{U}), cs : \phi)\theta^{YZ} \mid (\forall \mathbf{X}, cs : \phi) \in \Delta\}$
7: $\Delta_{ZY} \leftarrow \{(\forall(\mathbf{X} \setminus \mathbf{U}), cs : \phi)\theta^{ZY} \mid (\forall \mathbf{X}, cs : \phi) \in \Delta\}$
8: $\Delta' \leftarrow \Delta_{YZ} \wedge \Delta_{ZY}$
9: **return** $(\forall Y, Z, (cs_{\mathbf{U}}(Y) \wedge cs_{\mathbf{U}}(Z) \wedge (Y < Z)) : \text{Compile}(\Delta'))$

Postcondition 4.32. *The returned intensional conjunction is decomposable.*

Postcondition 4.33. *The returned intensional conjunction is automorphic.*

Postcondition 4.34 (Equivalence)**.** $\Delta \equiv (\forall Y, Z, (cs_{\mathbf{U}}(Y) \wedge cs_{\mathbf{U}}(Z) \wedge (Y < Z)) : \Delta')$

which is impossible, because it would imply that simultaneously $c_Z < c_Y$ and $c_Y < c_Z$. □

Proof of Postcondition 4.33. The constraint $Y < Z$ implies that $Y \neq Z$. Because of this inequality and Precondition 4.30, Theorem 4.4 applies and $(\forall Y, Z, (cs_{\mathbf{U}}(Y) \wedge cs_{\mathbf{U}}(Z) \wedge (Y < Z)) : \Delta')$ is automorphic. □

Proof outline of Postcondition 4.34. Analogous to the proof outline of Postcondition 4.29. □

Example 4.15. Figure 4.6 depicts the compiled circuit for Formula 4.1. The first compilation steps are shattered compilation and independence, as was depicted in Figure 4.3. The root's left branch represents the case where $X = Y$, which can be compiled by the independent single groundings rule. The right branch represents the case where $X \neq Y$. This theory (shown in Figure 4.3) has a root unifying class with two variables per formula and is compiled

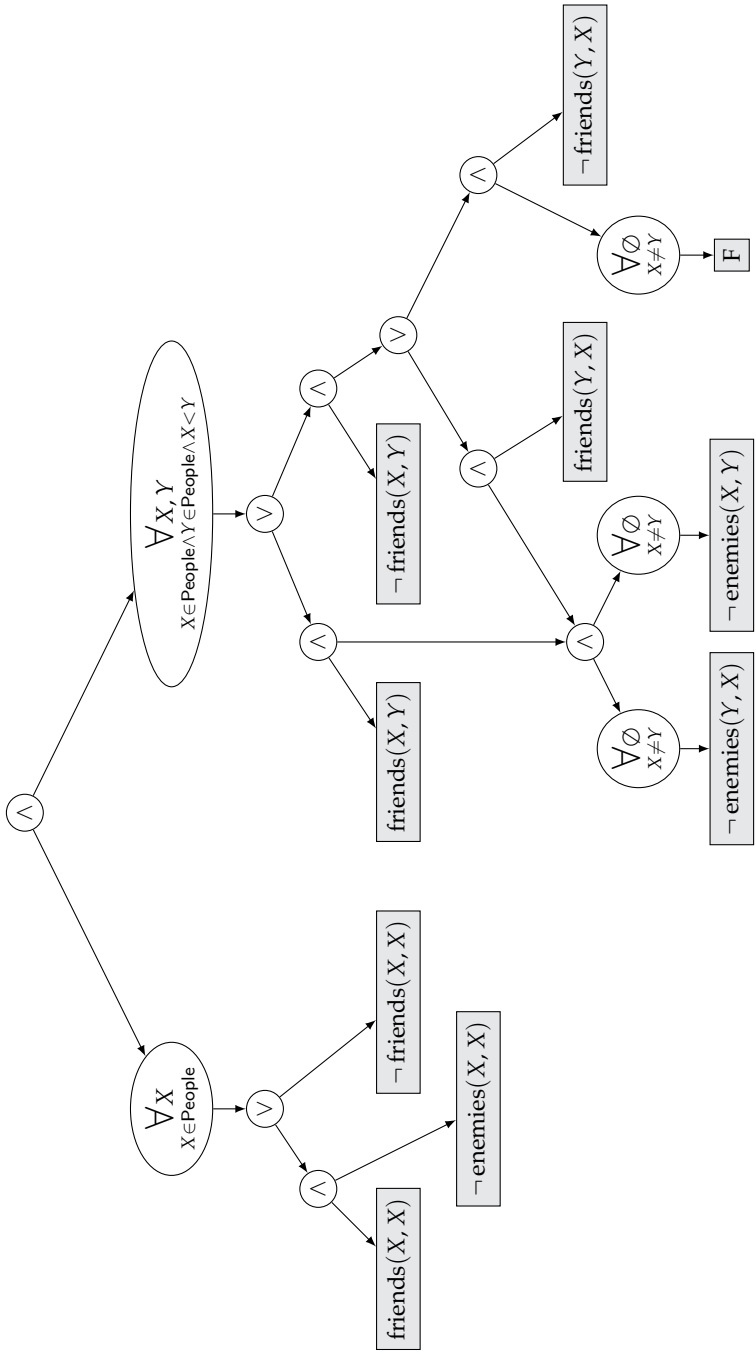


Figure 4.6: Compiled circuit for Formula 4.1

by the independent paired groundings rule, directly followed by Shannon decomposition.

4.4.5 Generalization to Any Root Unifying Class

It is possible to generalize the independent paired groundings compilation rule to apply to any root unifying class of a shattered theory with no independent subtheories. We first prove an additional property of root unifying classes.

Lemma 4.7. *In a shattered theory without any independent subtheories, all clauses and atoms contain the same number of variables from a root unifying class* $\mathbf{U}$*:*

$$\exists n \text{ s.t. } \forall \gamma \in \Delta, \forall a \in \text{atom}_c(\Delta) : |\text{bvars}(\gamma) \cap \mathbf{U}| = |V_a \cap \mathbf{U}| = n, \tag{4.5}$$

where V_a *are the logical variable arguments of* a*.*

Proof. Denote with Δ_n the clauses in Δ that contain n logical variables from $\mathbf{U}$ and with Δ_n^c its compliment in Δ. If Δ is nonempty, there is an n for which Δ_n is nonempty, and every atom in Δ_n contains exactly n variables from $\mathbf{U}$ (Definition 4.3). Since the theory contains no independent subtheories, there must be an atom a in Δ_n which unifies with an atom a' in Δ_n^c, or Δ_n^c is empty. After shattering, all unifications bind exactly one variable from a to exactly one variable from a'. Because a contains exactly n variables from $\mathbf{U}$, a' must also contain exactly n (Definition 4.2), and because $\mathbf{U}$ is a root unifying class, the clause of a_γ also contains exactly n, which contradicts the definition of Δ_n^c. Therefore, Δ_n^c is empty. Because the variables in $\mathbf{U}$ are root, they also appear in all atoms. □

It follows from Lemma 4.7 that when a theory is first shattered and independent subtheories are decomposed, any root unifying class has the same number of variables in each atom. It then follows from Lemma 4.6 that all of these variables share the same set of solutions and from Theorem 4.4 that the intensional conjunction over these variables is automorphic. These properties open up the possibility of generalizing the approach of Algorithm 14 to any root unifying class. This involves conjoining the theories for any permutation of the variables $Y, Z, \ldots$ and enforcing an ordering constraint $<$ in the intensional conjunction over these variables.

The formal specification of this generalized independent grounding rule is straightforward given Algorithm 14 but lengthy and therefore omitted here. We refer the reader to Taghipour et al. (2012b) for the details of a similar inference rule in first-order variable elimination that is also based on domain recursion (Van den Broeck, 2011b).

4.4.6 Atom Counting

We now present a compilation rule that applies to theories that contain an atom with exactly one bound logical variable argument. It generates automorphic and deterministic intensional disjunctions. An example is Formula 3.10, that is,

$$\forall X, Y \in \mathsf{People}: \ \mathrm{smokes}(X) \wedge \mathrm{friends}(X, Y) \Rightarrow \mathrm{smokes}(Y). \qquad (3.10')$$

It contains no root unifying class, because none of the variables are root. Other compilation rules also do not apply. Yet we can compile this formula by quantifying over all possible interpretations of the $\mathrm{smokes}(X)$ atom.

Atom counting (Algorithm 15) partitions the set of models depending on the arguments for which the atom is true. The disjunctions between these partitions are deterministic (Postcondition 4.37), because their models disagree in at least one atom. Because the theory has been shattered, the individual ground atoms are indistinguishable. As a result, the returned deterministic disjunction is automorphic (Postcondition 4.38).

Proof of Postcondition 4.37. We need to prove that for two distinct $[\mathsf{D}/d_1]$, $[\mathsf{D}/d_2] \in \mathrm{solutions}(cs_{\mathsf{D}}, \mathsf{D})$, we have that $\Delta'[\mathsf{D}/d_1] \wedge \Delta'[\mathsf{D}/d_2]$ is unsatisfiable. There exists a constant c which is contained in d_1 but not in d_2 (or vice versa). This means that $\Delta'[\mathsf{D}/d_1]$ is only satisfied in interpretations where $a[X/c]$ is true and $\Delta'[\mathsf{D}/d_2]$ is only satisfied in interpretations where $a[X/c]$ is false. Hence, their conjunction is unsatisfiable. □

Proof of Postcondition 4.38. We need to prove that for all $[\mathsf{D}/\{c_1, \ldots, c_n\}]$ and $[\mathsf{D}/\{c'_1, \ldots, c'_n\}]$ in $\mathrm{solutions}(cs_{\mathsf{D}}, \mathsf{D})$, the two theories $\Delta'[\mathsf{D}/\{c_1, \ldots, c_n\}]$ and $\Delta'[\mathsf{D}/\{c'_1, \ldots, c'_n\}]$ are equivalent up to a permutation of constants. Specifically, we can show that this permutation is $\pi = \{c_1 \mapsto c'_1, c'_1 \mapsto c_1, \ldots, c_n \mapsto c'_n, c'_n \mapsto c_n\}$. The proof is analogous to the proof of Lemma 4.3: (i) The constants $c_1, \ldots, c_n$ and $c'_1, \ldots, c'_n$ do not appear as logical terms in Δ'. (ii) Where c_i and c'_i appear

Algorithm 15 AtomCounting(Δ)

Precondition 4.35. *The theory is shattered:* Shatter(Δ) $= \Delta$.

Precondition 4.36. *There exists an atom in Δ with exactly one bound logical variable argument:* $\exists\,(\forall\{X\}, cs : a) \in \text{atom}_c(\Delta)$

Function.

1: **let** K be the set of constants in Δ
2: **let** V be the set of free logical variables in Δ
3: **let** $\mathcal{D}$ be the set of domain terms in Δ
4: $CS \leftarrow$ ShatterVar$(X, K \cup V, \mathcal{D})$
5: **for some** $cs_a \in CS$ such that $((\forall X, cs_a : a) \Rightarrow (\forall X, cs : a))$
6: **let** cs_D be such that solutions$(cs_\mathsf{D}, \mathsf{D}) = 2^{\text{solutions}(cs_a, X)}$
7: $\gamma_\mathrm{T} \leftarrow (\forall X, (X \in \mathsf{D} \wedge cs_a) : \quad a)$
8: $\gamma_\mathrm{F} \leftarrow (\forall X, (X \notin \mathsf{D} \wedge cs_a) : \neg a)$
9: $\Delta' \leftarrow \Delta \wedge \gamma_\mathrm{T} \wedge \gamma_\mathrm{F}$
10: **return** $\exists \mathsf{D}, cs_\mathsf{D} :$ Compile(Δ')

Postcondition 4.37. *The returned intensional disjunction is deterministic.*

Postcondition 4.38. *The returned intensional disjunction is automorphic.*

Postcondition 4.39 (Equivalence)**.** $\Delta \equiv (\exists \mathsf{D}, cs_\mathsf{D} :\ \Delta')$

inside domain terms other than D, they occur together, so that π does not change these domain terms. (iii) π maps $\gamma_\mathrm{T}[\mathsf{D}/\{c_1, \ldots, c_n\}]$ into $\gamma_\mathrm{T}[\mathsf{D}/\{c'_1, \ldots, c'_n\}]$ and $\gamma_\mathrm{F}[\mathsf{D}/\{c_1, \ldots, c_n\}]$ into $\gamma_\mathrm{F}[\mathsf{D}/\{c'_1, \ldots, c'_n\}]$. □

Proof outline of Postcondition 4.39. The formula $\exists \mathsf{D}, cs_\mathsf{D} :\ \gamma_\mathrm{T} \wedge \gamma_\mathrm{F}$ is a tautology, because it is satisfied in any interpretation by choosing D to be those constants for which a is satisfied by the interpretation. Then, because of distributivity, $\Delta \equiv (\Delta \wedge \mathrm{T}) \equiv (\Delta \wedge (\exists \mathsf{D}, cs_\mathsf{D} :\ \gamma_\mathrm{T} \wedge \gamma_\mathrm{F})) \equiv (\exists \mathsf{D}, cs_\mathsf{D} :\ \Delta \wedge \gamma_\mathrm{T} \wedge \gamma_\mathrm{F}) \equiv (\exists \mathsf{D}, cs_\mathsf{D} :\ \Delta')$. □

Atom counting compiles a `FO-da-DNNF` that is parametrized in D but whose size is independent of $|\,\text{solutions}(cs_\mathsf{D}, \mathsf{D})|$ and $|\,\text{solutions}(cs_a, X)|$. The rule by itself does not simplify the theory, but by introducing two new unit clauses, unit propagation will be able to eliminate the literal a from the theory entirely in subsequent compilation steps.

Line 6 of Algorithm 15 generates a new constraint set cs_{D} such that each solution to D is a subset of all solutions to X according to constraint set cs_a. This new constraint set can be generated as follows:

$$cs_{\mathsf{D}} = (\mathsf{D} \subseteq \mathsf{S}) \wedge \bigwedge_{(X \in \mathsf{F}) \in cs_a} (\mathsf{S} \subseteq \mathsf{F}) \quad \wedge \bigwedge_{(X \notin \mathsf{F}) \in cs_a} (\mathsf{S} \subseteq \mathsf{F}^c)$$

$$\wedge \bigwedge_{(X = Y) \in cs_a} (\mathsf{S} = \{Y\}) \wedge \bigwedge_{(X \neq Y) \in cs_a} (Y \notin \mathsf{S})$$

Note that the empty set $\emptyset$ is always a solution of cs_{D}.

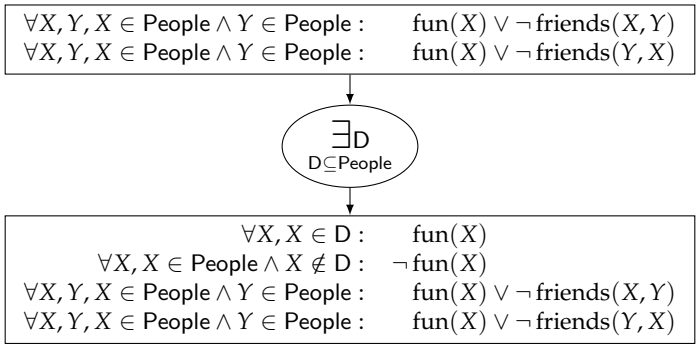


Figure 4.7: Atom Counting of $\forall X, X \in \mathsf{People} : \mathrm{fun}(X)$

Example 4.16. Figure 4.7 shows a theory Δ that only allows for atom counting. The $\mathrm{fun}(X)$ atom has $2^{|\mathsf{People}|}$ partial interpretations (e.g., $\{\mathrm{fun}(alice), \mathrm{fun}(bob)\}$, $\{\mathrm{fun}(alice), \neg\,\mathrm{fun}(bob)\}$,..., $\{\neg\,\mathrm{fun}(alice), \neg\,\mathrm{fun}(bob)\}$ when the number of people is 2). We can create a theory equivalent to Δ by conjoining each partial interpretation with Δ, 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 $2^{|\mathsf{People}|}$ subcircuits. However, we can observe that all subcircuits are isomorphic. They are identical up to the constants for which $\mathrm{fun}(X)$ is true. We need to compile only one subcircuit, parametrized in the subset D of People. Evaluating the model count of this circuit is only linear in $|\mathsf{People}|$, which makes this operator lifted.

Example 4.17. Atom counting only adds unit clauses to the theory. Figure 4.8 depicts the entire compiled circuit following Figure 4.8. It shows three

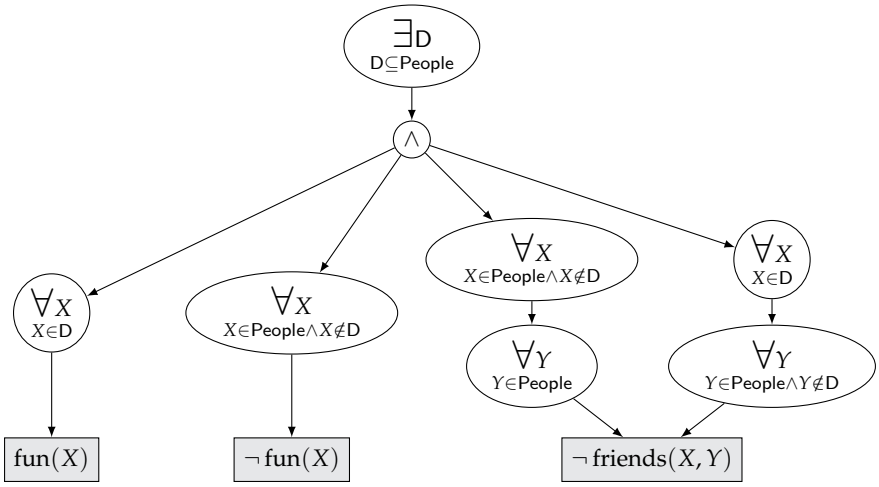


Figure 4.8: Circuit after continuing compilation of Figure 4.7

subsequent steps of unit propagation, generating four constrained literals, which are individually compiled into intensional conjunctions.

Example 4.18. The result of applying atom counting to Formula 3.10′ was depicted in Figure 3.2b. After atom counting, which introduces the root node, subsequent steps of independent single grounding compile the rest of the circuit.

4.5 Grounding

If no other compilation rule applies, there is one exceptional compilation rule, called Ground (Algorithm 16), that can be used to compile any sentence.

The Ground compilation rule grounds out a logical variable in the theory, after which compilation is retried. Postcondition 4.41 follows from the grounding semantics of sentences (Definition 3.11). A good heuristic for selection the variable X to ground out is to select the variable with the smallest number of solutions $|\text{solutions}(X, cs)|$.

The grounding rule always applies to theories that are sentences, until all logical variables are removed. Then, propositional knowledge compilation algorithms can compile it into a `d-DNNF` circuit, which is also a `FO-da-DNNF`

Algorithm 16 GROUND(Δ)

Precondition 4.40. *There exists a constraint set cs in Δ without free variables.*

Function.

1: **let** γ be the constrained clause containing *cs*
2: **for some** $X \in \text{bvars}(\gamma)$
3: $\Gamma \leftarrow \{\gamma\theta \mid \theta \in \text{solutions}(cs, X)\}$
4: **return** COMPILE $(\Delta \setminus \gamma) \cup \Gamma$

Postcondition 4.41. $((\Delta \setminus \gamma) \cup \Gamma) \equiv \Delta$

circuit. A disadvantage of using the grounding rule is the shift in complexity it causes. Whereas the complexity of compiling a FO-da-DNNF circuit with other rules is *independent of the size of the domain terms* in the theory, grounding the theory will cause the complexity of compilation to be potentially exponential in this number. After grounding every logical variable in the theory, the complexity of compilation is *exponential in the treewidth* of the input CNF (Darwiche, 2001b).

4.6 First-Order Smoothing

Compilation steps such as unit propagation and atom counting may remove literals from branches of the FO-da-DNNF. The groundings of these literals may go unaccounted for when doing model counting in the circuit. For propositional NNF circuits, this problem is solved by *smoothing* the circuit (Darwiche and Marquis, 2002). This section presents a lifted smoothing algorithm for *first-order* circuits. It compiles FO-da-DNNF circuits into FO-sda-DNNF by enforcing the smoothness property (Definitions 3.23 and 3.24).

Smoothing first propagates a set of constrained atoms upwards in the circuit, representing the ground atoms that are accounted for in the circuit. In the propositional case, these constrained atoms are just sets of propositions and during propagation, the set of propositions represented by a parent node is the union of the sets of the children.

For first-order circuits, this step is performed by the following function. It takes as input a FO-NNF circuit and returns back a set of constrained atoms.

Definition 4.4 (atom$_c$ of a FO-NNF Circuit)**.** The base cases of the atom$_c$ function are count(F) = count(T) = $\emptyset$ and atom$_c(l) = \{a\}$ when l is a *literal* and a its atom. The recursive definition is as follows. For an *extensional conjunction* $\psi = \phi_1 \wedge \cdots \wedge \phi_n$ or *disjunction* $\psi = \phi_1 \vee \cdots \vee \phi_n$,

$$\text{atom}_c(\psi) = \text{atom}_c(\phi_1) \cup \cdots \cup \text{atom}_c(\phi_n).$$

For an *intensional conjunction* or *disjunction* $\psi = ((\forall, \exists)\mathbf{V}, cs : \phi)$, atom$_c$:

$$\text{atom}_c(\psi) = \{(\forall(\mathbf{Y} \cup \mathbf{V}), (cs_a \wedge cs) : a) \mid (\forall \mathbf{Y}, cs_a : a) \in \text{atom}_c(\phi)\}$$

For intensional nodes, atom$_c$ adds the constraints of the intensional node to the constrained atoms of the child circuit. Each first-order atom represents a set of ground atoms. A set of first-order atoms in turn represents the union of these groundings. We will implicitly assume that the returned constrained atoms are mutually independent. This can be enforced by a procedure similar to the Split function (Algorithm 2). Note that the atom$_c$ function is related to the grounding function gr (Definition 3.11) as follows: The atoms appearing in the grounding gr of a sentence are the ground atoms represented by the atom$_c$ of the same sentence. The difference is that atom$_c$ also takes formulas as input that are not sentences.

Second, smoothing modifies the circuit to make every disjunctive node smooth. In the propositional case, let P_p be the set of propositions represented by a parent node and P_c be the ones represented by one of its child disjuncts ϕ. Assume that there is a non-empty set of propositions $\{p_1, \ldots, p_n\} = P_p \setminus P_c$ that are not represented by the child. Smoothing then replaces that child node by the circuit $\phi \wedge (p_1 \vee \neg p_1) \wedge \cdots \wedge (p_n \vee \neg p_n)$. Note that the conjunctions in this formula are decomposable and the disjunctions are deterministic. When doing this for every child node, the parent node becomes smooth.

The procedure for first-order circuits is similar. Let ϕ_p be a disjunctive parent node (extensional or intensional) and ϕ_c its child. Smoothing then constructs a set of independent constrained atoms $A_c = \{(\forall \mathbf{V}_1, cs_1 : a_1), \ldots, (\forall \mathbf{V}_m, cs_m : a_m)\}$ such that $\forall i : \phi_c \perp\!\!\!\perp (\forall \mathbf{V}_i, cs_i : a_i)$ and atom$_c(\phi_c) \cup$ atom$_c(A_c) =$ atom$_c(\phi_p)$. These are then inserted into the circuit by replacing the child ϕ_c by

$$\phi_c \wedge (\forall \mathbf{V}_1, cs_1 : a_1 \vee \neg a_1) \wedge \cdots \wedge (\forall \mathbf{V}_m, cs_m : a_m \vee \neg a_m).$$

Doing this for every node makes the circuit smooth. Because the $(\forall \mathbf{V}_i, cs_i : a_i \vee$

$\neg a_i$) are tautologies, adding them does not change the semantics of the circuit. After shattering, these additional intensional conjunctions are decomposable and automorphic.

Example 4.19. The circuit in Figure 4.5 is not smooth. The bottom extensional disjunction's operands cover a different set of atoms. The left branch covers only friends(X, Y) while the right branch covers friends(X, Y) and dislikes(X, Y). In this case, first-order smoothing is identical to propositional smoothing. It substitutes the left branch by friends$(X, Y) \wedge$ (dislikes$(X, Y) \vee \neg$ dislikes(X, Y)).

Example 4.20. The circuit in Figure 4.8 is not smooth. The theory in the root node (Figure 4.7) covers atoms fun(X) and friends(X, Y). The circuit below the root node covers fun(X) entirely, but is not accounting for the atoms friends$(X, Y), X \in \mathsf{D}, Y \in \mathsf{D}$. First-order smoothing compensates for this by inserting a conjunction below the intensional disjunction. The operands of the new conjunction are (i) the original child of the intensional disjunction and (ii) the compiled circuit for the theory $\forall X, Y, X \in \mathsf{D}, Y \in \mathsf{D}:$ friends$(X, Y) \vee \neg$ friends(X, Y).

Example 4.21. Figure 3.3b depicted the smoothed circuit of Figure 3.2b for Formula 3.10.

4.7 Related Work

Our first-order knowledge compilation algorithm is related to and draws inspiration from algorithms for related problems. We will briefly describe the relation to propositional knowledge compilation and first-order search algorithms.

4.7.1 Relation to Propositional Knowledge Compilation

On propositional input, our compilation algorithm uses the same top-down approach as the c2d (Darwiche, 2004) and Dsharp (Muise et al., 2010; Muise et al., 2012) compilers to propositional `d-DNNF`, although we do not use the same advanced techniques for caching, clause learning, etc. For these inputs, our algorithm uses the Shannon decomposition, unit propagation and decomposition compilation rules only. These rules are first-order generalizations of the same compilation rules in propositional compilers.

The main difference between the c2d and Dsharp compilers is in the decomposition step. The c2d compiler first builds a decomposition tree over the set of clauses, which is then used to guide a *static* decomposition during the top-down search. In contrast, Dsharp attempts to find a *dynamic* decomposition during compilation. Our independence compilation rule follows this dynamic decomposition approach.

4.7.2 Relation to Lifted Search Algorithms

Huang and Darwiche (2005) observed that there is a deep connection between propositional knowledge compilation and the trace of exhaustive DPLL search (Davis and Putnam, 1960; Davis, Logemann, and Loveland, 1962), in the sense that the trace of certain DPLL search algorithms corresponds to a `d-DNNF` circuit. Similarly, the trace of a top-down `d-DNNF` compiler corresponds to an exhaustive DPLL search. This deep connection also exists at the first-order level, between first-order DPLL search (Baumgartner, 2000; Baumgartner and Tinelli, 2008; Gogate and Domingos, 2011), lifted AND/OR search (Gogate and Domingos, 2010) and `FO-sda-DNNF` circuits, as we show next.

Lifted AND/OR Search and Probabilistic Theorem Proving

Gogate and Domingos (2010) present a *lifted AND/OR search* algorithm for counting the solutions to a (weighted) first-order constraint satisfaction problem. The trace of this algorithm corresponds to a multi-valued equivalent of `FO-sda-DNNF`. The *AND* and *OR* nodes in their search tree correspond to *decomposable conjunctions* and *deterministic disjunctions*, as in the propositional case.[5] Gogate and Domingos extend the propositional AND/OR search tree with *POWER-AND* and *POWER-OR* nodes. The former correspond to our *intensional conjunctions*.

AND/OR search was reformulated as *Probabilistic Theorem Proving* (PTP) (Gogate and Domingos, 2011), also an algorithm for (weighted) model counting. Although there are differences in the way lifted AND/OR search and PTP assign weights to models, their search spaces do not fundamentally differ. PTP uses two main operations of *decomposition* and *splitting* for lifted model counting, which

[5] Although Dechter and Mateescu (2007) note that deterministic disjunctions are theoretically more expressive than OR nodes, this difference does not show in the nodes compiled by our algorithm.

are similar to our *independent single groundings* and *atom counting* compilation rules. Similar to our approach, PTP also employs *unit* or *constraint propagation* to simplify the theory.

The first-order language used for PTP and lifted AND/OR search cannot express *free variables*, neither domain nor logical, and allows only for (in)equality constraints to be associated with logical variables. A consequence of the lack of free variables is that the theories whose solutions are being counted at each step of the search algorithm have to be sentences. This leads to three important differences.

1. PTP and lifted AND/OR search compute the model count of a specific theory with specific domains. By compiling theories with free variables, we can later on substitute constants and sets of constants in the compiled circuits in order to *reuse* them to compute many different model counts (e.g., for different domain sizes). This is an important motivation for using the knowledge compilation approach.

2. POWER-OR and splitting nodes are at first sight similar to our intensional disjunctions, but are essentially different. When reaching a POWER-OR/splitting node in the search tree, PTP performs a case analysis of the assignments to the groundings of some constrained atom a. It branches the search into $n + 1$ subproblems, one for each number of true atoms in a truth assignment to the n groundings of a. In each branch, it computes the model count for *one specific interpretation* of a. In contrast to that, an intensional `FO-sda-DNNF` disjunction has one single child node. Each interpretation of a is represented by a single domain variable D, quantified by the intensional node. When further compiling this child node, the domain D is a free variable. Hence, the AND/OR and PTP search space can be polynomially larger in terms of the domains of the logical variables, and even *exponentially larger* in terms of the number of recursive POWER-OR/splitting nodes.

3. For the specific goal of counting models, it is reasonable to consider only $n + 1$ specific interpretations. But looking at the trace of these algorithms, the logical equivalence between the theories at various levels of the search tree is lost, because many interpretations are not represented in the search trace. This makes it *hard to extract a* `FO-da-DNNF` *circuit* from a given

trace of these search algorithms.[6] In contrast, our compilation algorithm explicitly represents only a single subdomain, using a domain variable, but implicitly represents all instantiations by existentially quantifying that variable.

Despite these differences in the compilation phase, when using a FO-da-DNNF for model counting, we will iterate over all $n + 1$ subcircuits (Definition 3.33) in an intensional disjunction generated by the atom counting operator. Consequently, the nodes in the trace of our *counting algorithm* correspond one-to-one to the search space of PTP and lifted AND/OR search.

The techniques used in our compilation rules, in PTP and lifted AND/OR search were inspired by the work on lifted probabilistic inference algorithms. We will look at these algorithms in more detail in the following chapter, but already note some relations. Independent single grounding is based on the same underlying ideas as *partial inversion* in FOVE (de Salvo Braz, Amir, and Roth, 2006) and the *power rule* in CPs (Jha et al., 2010). Atom counting is inspired by *counting formulas* and *counting elimination* in FOVE and the *generalized binomial rule* in CPs.

Even with these shared sources of inspiration, there are notable differences between our compilation rules and PTP or AND/OR search operators. The search algorithms do not have a corresponding lifted operator for our *independent paired grounding* compilation rule, and thus resort to grounding for (sub-)problems that we can solve with first-order knowledge compilation. As we will see in Chapter 6, this missing operator is essential in obtaining theoretical results of completeness. PTP and lifted AND/OR algorithms are described at a high level of generality. For instance, in Gogate and Domingos (2011), no details are given on how to find decomposers and atoms to split on. Therefore, it is difficult to make further claims about the relation to our compilation rules at a more technical level.

4.8 Conclusions and Future Work

This chapter presented an algorithm to compile a theory in FOL-DC into a FO-da-DNNF circuit. This algorithm is composed of a set of compilation rules,

[6] In the worst case, all $n + 1$ chidren of an POWER-OR/splitting node have a different search trace, and no single FO-da-DNNF subcircuit can represent them all.

which recursively simplify the theory and introduce nodes to the circuit. Each compilation rule was described with preconditions and postconditions, precisely stating which theories it applies to and what its effects are. We showed that by applying the smoothing operation to a FO-da-DNNF circuit, it can be transformed into a FO-sda-DNNF circuit.

There are several directions of future work. The input language supported by our compilation algorithm is restrictive (see Section 4.1.1). It has to be a FO-CNF without intensional disjunctions and a specific type of constraint sets. Relaxing some of the assumptions can widen the applicability of first-order knowledge compilation. In particular, adding support for *intensional disjunctions* would enable the use of first-order knowledge compilation for inference in directed probabilistic models (see Chapter 5).

As discussed in Section 4.3.3, *shattering* is still poorly understood, yet essential to building efficient algorithms for first-order knowledge compilation and lifted inference. It should be possible to avoid shattering in many more cases than we do now.

Propositional SAT solvers and knowledge compilation algorithms use certain techniques that are not always required for the algorithms to be correct, but that lead to significant speedups, including *caching*, *clause learning*, *non-chronological backtracking* and heuristics for finding *variable orderings* and *tree decompositions* (Zhang et al., 2001; Darwiche, 2004; Muise et al., 2010). One of the next steps for first-order knowledge compilation and lifted inference research is to also lift these techniques to the first-order level. Our compilation algorithm already uses a naive caching approach. Otherwise, the problem of lifting these techniques is largely unexplored.

To better understand the complexity of compilation and in order to optimize the order in which we apply the compilation rules, a more thorough analysis is required. We believe that analogous to the propositional case, notions of lifted tree decomposition and lifted treewidth can be defined to more precisely determine the complexity of these algorithms.

5

Exact Lifted Probabilistic Inference

Propositional logic and probabilistic graphical models suffer from the same limitations: they make it difficult to express general, *high-level knowledge* and to model complex interactions between objects in the world. For propositional logic, this problem is solved by using first-order logic, which can express knowledge at a higher level of generality, talking about properties of objects and the relations that hold between them.

Nonetheless, first-order logic retains another limitation of propositional logic: it is difficult to express commonsense knowledge and *uncertain information*. Probabilistic graphical models provide a solution to this problem, with their ability to concisely model probability distributions over a large number of propositional random variables. For these reasons, there is a lot of interest in combining probabilities and first-order logic for dealing with both uncertain knowledge and complex relational domains. This interest has resulted in the fields of Statistical Relational Learning (SRL) and Probabilistic Logic Learning (PLL). This chapter is concerned with the type of models used in these fields and the problem of performing efficient exact inference in these models.

Due to the increased expressivity of SRL models, inference is often *intractable* when using probabilistic graphical model inference algorithms. The alternative

is to develop new approaches that exploit the first-order aspect of SRL models to speed up probabilistic inference. These algorithms are called *lifted* inference algorithms. They exploit the abundant symmetries in SRL models to avoid repeated computations. We will show how to use weighted first-order model counting as an approach to exact lifted probabilistic inference. In particular, we show in this chapter that first-order knowledge compilation to `FO-sda-DNNF` can be used to perform weighted first-order model counting and thereby lifted inference.

Section 5.1 briefly reviews the literature on SRL and PLL. It introduces the representation languages used in this and the following chapters. Then, **Sections 5.2 and 5.3** discuss inference algorithms for these languages. The first class of algorithms reduce the problem to inference in propositional models. The second class are lifted inference algorithms that reason at the first-order level. **Section 5.4** describes the first contribution of this chapter: lifted inference by weighted first-order model counting. The second contribution is described in **Section 5.5**. It is an algorithm to perform weighted first-order model counting by first-order knowledge compilation. These two sections constitute a new exact lifted inference algorithm. **Section 5.6** describes the relationship between this work and existing exact lifted inference algorithms. Finally, our approach is evaluated empirically in **Section 5.7**.

The general first-order knowledge compilation approach to lifted inference was previously published in

> G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011a). "Lifted probabilistic inference by first-order knowledge compilation". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. Menlo Park, California, pp. 2178–2185

Our lifted algorithm for efficiently computing conditional probabilities (Section 5.5.3) was previously published in

> G. Van den Broeck and J. Davis (2012). "Conditioning in first-order knowledge compilation and lifted probabilistic inference". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence,* Palo Alto, California, USA

Some additional experiments we report on were published in Van den Broeck (2011b).

5.1 Statistical Relational Learning

Statistical relational learning (Getoor and Taskar, 2007) and Probabilistic logic learning (De Raedt et al., 2008) are research areas at the intersection of three fields. The first is probability theory or probabilistic graphical models, the second is first-order logic or relational databases, and the third is machine learning. While the two approaches essentially study the same problem, there are differences in emphasis. Whereas SRL is rooted in relational databases, relational learning and probabilistic graphical models, PLL emerged from the inductive logic programming (Muggleton and De Raedt, 1994) community, with attempts of adding probabilistic constructs to logic programming languages. In essence, both fields have converged to being concerned with modeling and learning complex logical and probabilistic interactions between large numbers of objects. We will here refer to all SRL and PLL formalisms as SRL, even when they originated in the PLL community.

Similar to the difference between directed and undirected probabilistic graphical models (e.g., Bayesian networks vs. Markov random fields, see Section 2.3), we can distinguish between directed and undirected SRL languages. First, we describe Markov logic networks and parfactor graphs, which are undirected models. Second, we describe relational extensions of Bayesian networks and probabilistic logic programming languages, which are directed models.

5.1.1 Markov Logic Networks

Markov logic (Richardson and Domingos, 2006) combines Markov networks with first-order logic. Formally, a Markov logic network (MLN) is a set of pairs, (w_i, ψ_i), where ψ_i is a first-order formula and $w_i \in \mathbb{R}$. MLNs soften logic by associating a weight with each formula. Possible worlds that violate formulas become less likely, but not impossible. Intuitively, as w_i increases, so does the strength the constraint ψ_i imposes on the world. Formulas with infinite weights, called *hard formulas*, represent pure logic formulas. The probability of worlds that do not satisfy a hard formula are set to zero.

The features ψ_i are formulas in typed function-free first-order logic. These formulas may contain free variables. They represent a set of sentences, one for each way of instantiating the free variables. We will refer to this set as the *instances* of a MLN formula. In the literature, these instances are sometimes called the *groundings* of a formula. This terminology is not precise, because these sentences might still contain quantifiers and bound logical variables.[1]

MLNs provide a template for constructing Markov networks. When given a finite set of constants (the domain), the MLN formulas define a Markov network. Nodes in the network (random variables) are ground atoms. Edges connect literals that appear in the same instance of a formula. An MLN induces the following probability distribution:

$$\Pr(\omega) = \frac{1}{Z} \exp\left(\sum_i^{|\Psi|} w_i n_i(\omega)\right) \tag{5.1}$$

where Ψ is the set of formulas in the MLN, w_i is the weight of the i^{th} formula, and $n_i(\omega)$ is the number of true instances of formula ψ_i in possible world ω.

Note the correspondence to Equation 2.3 for log-linear Markov networks. The fundamental difference in semantics between first-order and propositional Markov networks is that a MLN formula has multiple instances with the same weight. Whereas a feature in propositional logic is either true or false, a first-order feature has an associated count n_i of how many of its instances are true.

Example 5.1. Consider the following MLN (Singla and Domingos, 2008):

$$1.5 \quad \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y) \tag{5.2}$$

$$1.3 \quad \text{smokes}(X) \Rightarrow \text{cancer}(X) \tag{5.3}$$

The purely logical meaning of Formula 5.2 is that smokers are only friends with other smokers. By associating a high weight 1.5 with this formula, its meaning becomes that smokers are more likely to be friends with other smokers. Formula (5.3) states that smokers are more likely to get cancer. This MLN will be a running example throughout this thesis. Assuming a domain of two

[1] It is a common misconception that all free variables in a MLN formula are universally quantified. This is not the case: there is a clear difference in semantics between a formula that has free variables and a formula where variables are universally quantified. The formula $(2, \text{p}(X))$ encodes that the probability of a possible world increases by a factor e^2 with every true p-atom in the world. The formula $(2, \forall X : \text{p}(X))$ encodes that the world in which all p-atoms are true has a probability that is e^2 times more likely than all other worlds.

constants, *alice* and *bob*, this MLN induces a distribution over eight random variables, which correspond to all groundings of atoms smokes(X), cancer(X) and friends(X, Y), such as smokes$(alice)$, cancer(bob) and friends$(alice, bob)$. This leads to a distribution over 2^8 possible worlds. Formula 5.2 has four instances and Formula 5.3 has two. The MLN above is equivalent to the following MLN, where each formula is replaced by its instances.

1.5 $\text{smokes}(alice) \wedge \text{friends}(alice, alice) \Rightarrow \text{smokes}(alice)$

1.5 $\text{smokes}(alice) \wedge \text{friends}(alice, bob) \Rightarrow \text{smokes}(bob)$

1.5 $\text{smokes}(bob) \wedge \text{friends}(bob, alice) \Rightarrow \text{smokes}(alice)$

1.5 $\text{smokes}(bob) \wedge \text{friends}(bob, bob) \Rightarrow \text{smokes}(bob)$

1.3 $\text{smokes}(alice) \Rightarrow \text{cancer}(alice)$

1.3 $\text{smokes}(bob) \Rightarrow \text{cancer}(bob)$

When n instances of Formula 5.2 are true in a possible world, and m instances of Formula 5.3 are true, the weight of that possible world is $e^{1.5n+1.3m}/Z$.

See Domingos and Lowd (2009) for a detailed overview of the literature on Markov logic, including inference and learning algorithms, and applications.

5.1.2 Parfactor Graphs

In the lifted probabilistic inference literature, parfactor graphs are a popular representation language. A parfactor graph is a collection of parametric factors, or parfactors (Poole, 2003).

Definition 5.1 (Parfactor)**.** A parfactor has the form $(\mathbf{X}, cs, A, f)$ where $\mathbf{X}$ is a set of logical variables, cs is a constraint set on variables $\mathbf{X}$, A is a list of atoms, and f is a potential on A.

Parfactors are often represented with the syntax $\forall \mathbf{X}, cs : f(a_1, \ldots, a_m)$, where the a_i are atoms in A. Each parfactor compactly describes a set of factors. The parfactor $\forall \mathbf{X}, cs : f(a_1, \ldots, a_m)$ represents a set of factors of the form $f(a_1\theta, \ldots, a_m\theta)$, one for each $\theta \in \text{solutions}(cs, \mathbf{X})$. Each substitution grounds the atoms in A, turning them into propositional random variables. What is left

is a propositional factor graph. The semantics of a parfactor graph consisting of the parfactors $(\mathbf{X}_i, cs_i, A_i, f_i)$ directly follow from the semantics of factor graphs (Equation 2.2):

$$\Pr(\omega) = \frac{1}{Z} \prod_i \prod_{\theta \in \text{solutions}(cs_i, \mathbf{X}_i)} f_i(\omega(A_i\theta)),$$

Here, $\omega(A_i\theta)$ denotes the assignments the possible world ω makes to the variables $A_i\theta$.

As for MLNs, it is assumed that predicates are typed and that there is a given domain for each type, in the form of a finite set of constants. For the sake of simplicity, the constraint sets used are often normal form constraints (Milch et al., 2008), that is, conjunctions of inequality constraints only. However, more expressive constraint languages can also be used, as in Kisyński and Poole (2009a) and Taghipour et al. (2012).

Example 5.2. Consider the parfactor

$$\forall X, Y : f(\text{smokes}(X), \text{smokes}(Y), \text{friends}(X, Y))$$

Assuming constants *alice* and *bob*, it represents the propositional factor graph

$$f(\text{smokes}(\mathit{alice}), \text{friends}(\mathit{alice}, \mathit{alice}))$$

$$f(\text{smokes}(\mathit{alice}), \text{smokes}(\mathit{bob}), \text{friends}(\mathit{alice}, \mathit{bob}))$$

$$f(\text{smokes}(\mathit{bob}), \text{smokes}(\mathit{alice}), \text{friends}(\mathit{bob}, \mathit{alice}))$$

$$f(\text{smokes}(\mathit{bob}), \text{friends}(\mathit{bob}, \mathit{bob}))$$

When we add the constraint that $X \neq \mathit{alice}$, we obtain parfactor $\forall X, Y, X \neq \mathit{alice} : f(\text{smokes}(X), \text{smokes}(Y), \text{friends}(X, Y))$, which corresponds to a factor graph with only the last two factors shown above. All the above is independent

of the precise potential chosen for f. If we choose it to be

f =

smokes(X)	smokes(Y)	friends(X, Y)	f
F	F	F	$e^{1.5}$
F	F	T	$e^{1.5}$
F	T	F	$e^{1.5}$
F	T	T	$e^{1.5}$
T	F	F	$e^{1.5}$
T	F	T	1
T	T	F	$e^{1.5}$
T	T	T	$e^{1.5}$

this parfactor graph models the same distribution as MLN Formula 5.2.

Milch et al. (2008) extend the parfactor formalism with counting formulas. For the case of binary random variables, these formulas represent a count of how many groundings of an atom are true in the world. These counts are themselves random variables, so the parfactor associates a different potential with each of its values. The concept of counting formulas was further extended by Apsel and Brafman (2011) and Taghipour and Davis (2012).

Example 5.3.

5.1.3 Probabilistic Logic Progamming

Most probabilistic logic programming (PLP) languages, including PHA (Poole, 1993), SLP (Muggleton, 1996), PRISM (Sato and Kameya, 1997), ICL (Poole, 1997), LPAD (Vennekens, Verbaeten, and Bruynooghe, 2004), ProbLog (De Raedt, Kimmig, and Toivonen, 2007) and CP-logic (Vennekens, Denecker, and Bruynooghe, 2009), are based on the *distribution semantics* of Sato (1995). As a representative of probabilistic logic progamming, we will now discuss ProbLog in more detail.

ProbLog

A ProbLog program consists of two parts: a set of probabilistic facts $\mathcal{F}$ and a set of rules $\mathcal{R}$. The rules are written in standard Prolog syntax. The syntax for the probabilistic facts is $p_i :: f_i$, meaning that fact $f_i\theta$ is true with probability p_i

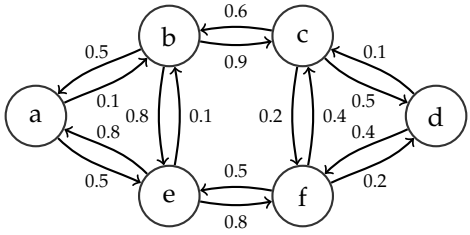


Figure 5.1: Probabilistic Network Example

for all substitutions θ that ground f_i. Probabilistic facts represent independent random variables. Let $\mathcal{G}$ be the set of all groundings of f_i facts in $\mathcal{F}$.[2] The ProbLog program represents a probability distribution over logic programs $F \subseteq \mathcal{G}$, which consist of ground facts.

$$\Pr(F) = \prod_{f_i \in F} p_i \prod_{f_i \in \mathcal{G} \setminus F} (1 - p_i) \tag{5.4}$$

For a fixed set of rules $\mathcal{R}$, each set of facts F corresponds to a unique Herbrand interpretation of the logic program $\mathcal{R} \cup F$. Therefore, $\Pr(F)$ also defines a probability distribution over these Herbrand interpretations.

The *success probability* of a query q is the probability that it succeeds in a randomly sampled logic program. This is defined as

$$\Pr(q) = \sum_{F \subseteq \mathcal{G}} \mathbf{1}_q(F) \cdot \Pr(F), \tag{5.5}$$

where $\mathbf{1}_q(F) = 1$ if $\mathcal{R} \cup F \models q\theta$ for some substitution θ, and 0 otherwise.

Example 5.4. The following ProbLog program represents the small probabilistic network of Figure 5.1. The network itself is defined by probabilistic facts $\mathcal{F}$. The

[2] We assumed that $\mathcal{G}$ is finite. See Sato (1995) for an extension of these semantics to the infinite case.

definition of a path in the network is encoded by the rules $\mathcal{R}$.

$$\mathcal{F} = \left\{ \begin{array}{ll} 0.1 :: \texttt{edge(a,b)}. & 0.5 :: \texttt{edge(b,a)}. \\ 0.9 :: \texttt{edge(b,c)}. & 0.6 :: \texttt{edge(c,b)}. \\ 0.5 :: \texttt{edge(c,d)}. & 0.1 :: \texttt{edge(d,c)}. \\ \vdots & \vdots \end{array} \right\}$$

$$\mathcal{R} = \left\{ \begin{array}{l} \texttt{path(X,Y) :- edge(X,Y).} \\ \texttt{path(X,Y) :- edge(X,Z),path(Z,Y).} \end{array} \right\}$$

Because of the syntactic restrictions of logic programs, it is tempting to believe that FOL is more 'expressive' than logic programming. This is wrong because of the difference in semantics. In the knowledge representation literature, it has been shown that certain concepts that can be expressed in (non-ground) logic programs cannot be expressed in (non-ground) FOL, such as inductive definitions (Denecker, Bruynooghe, and Marek, 2001) and transitive closure. For example, the relation between two nodes in a graph that expresses whether there exists a path between them cannot be expressed in first-order logic and its probabilistic extensions (Fierens et al., 2012a). This additional expressivity motivates the work on (probabilistic) logic programming.

5.1.4 Other Approaches

The early work of Nilsson (1986) on combining logic and probability was instrumental to the rise of SRL and PLL. Several people followed up on it, including Halpern (1990), who introduced the notions of type 1 and type 2 probabilities, corresponding to different types of statistical information, and Bacchus (1991); Bacchus et al. (1996). The disadvantage of these early works was that they could not yet provide a straightforward reduction from their formalisms to probabilistic graphical models, which were being developed around the same time. Consequently, there were no efficient systems or algorithms for inference and learning. From a knowledge representation point of view, MLNs can be regarded as a *maximum-entropy* probabilistic logic in the tradition of Nilsson (Paskin, 2002).

In a later phase, a multitude of first-order generalization of Bayesian networks were proposed, starting with plate models (Buntine, 1994), which can express repeated substructures in graphical models, to what is sometimes referred

Intermezzo 2: Statistical Relational Decision Making

In order to act rationally, intelligent agents need to solve decision-theoretic problems. In these type of problems, one has to choose actions from a set of alternatives, given a utility function. The goal is to select the strategy (set or sequence of actions) that maximizes the agent's expected utility. While decision theory commonly deals with uncertainty (e.g., using influence diagrams (Shachter, 1986)), there are only few approaches that cope with both *uncertainty* and rich logical or *relational* representations. One approach combines the ProbLog language with elements of decision theory. The resulting probabilistic programming language *Decision-Theoretic ProbLog* (DTProblog, Van den Broeck et al., 2010) is able to elegantly represent decision problems in complex relational and uncertain environments. A DTProbLog program consists of a ProbLog program, that is, a set of Prolog rules $\mathcal{R}$ and a set of probabilistic facts $\mathcal{F}$ with, in addition, a set of decision facts $\mathcal{D}$, specifying which decisions are to be made, and a set of utility attributes $\mathcal{U}$, specifying the rewards that can be obtained.

Example 5.5. Consider the following viral marketing problem. The decisions are whether to market to people in a social network consisting of $\texttt{trusts}(\texttt{X},\texttt{Y})$ relations. A reward of 5 is given for anyone buying the product and marketing to one person costs 2. People that are marketed or that trust someone who bought the product may buy the product. In DTProbLog, this is modeled as

$$\mathcal{F} = \left\{ \begin{array}{l} 0.3 :: \texttt{marketing\_influence}(\texttt{X}). \\ 0.4 :: \texttt{viral\_influence}(\texttt{X},\texttt{Y}). \end{array} \right\}$$

$$\mathcal{R} = \left\{ \begin{array}{l} \texttt{buys}(\texttt{X}) \texttt{ :- } \texttt{market}(\texttt{X}), \texttt{marketing\_influence}(\texttt{X}). \\ \texttt{buys}(\texttt{X}) \texttt{ :- } \texttt{trusts}(\texttt{X},\texttt{Y}), \texttt{buys}(\texttt{Y}), \texttt{viral\_influence}(\texttt{X},\texttt{Y}). \\ \texttt{trusts}(\texttt{a},\texttt{b}). \quad \texttt{trusts}(\texttt{b},\texttt{c}). \quad \cdots \end{array} \right\}$$

$$\mathcal{D} = \left\{ \; ? :: \texttt{market}(\texttt{X}). \; \right\} \qquad \mathcal{U} = \left\{ \begin{array}{l} \texttt{market}(\texttt{X}) \rightarrow -2. \\ \texttt{buys}(\texttt{X}) \rightarrow 5. \end{array} \right\}$$

Knowledge compilation algorithms can compute the optimal strategy for a DTProbLog program exactly and approximately. These scalable approximation algorithms can tackle large decision problems. For example, Van den Broeck et al. report on a viral marketing experiment with half a million trust relations. Examples of other relational decision-theoretic languages are the ICL (Poole, 1997), IBAL (Pfeffer, 2001), DTLPs (Chen and Muggleton, 2009) and MLDNs (Nath and Domingos, 2009). For MLDNs, Apsel and Brafman (2012b) show that WFOMC inference, which we present in Section 5.4.1, can compute the expected utility of a given strategy in a lifted manner.

to as the alphabet soup of SRL, including languages such as RBNs (Jaeger, 1997), PRMs (Friedman et al., 1999), BLPs (Kersting and De Raedt, 2001a), CLP($\mathcal{BN}$) (Santos Costa et al., 2003), and LBNs (Fierens et al., 2005).

More recently, probabilistic programming languages are gaining popularity. These approaches extend a programming language with probabilistic constructs. Examples include BLOG (Milch et al., 2007), Church (Goodman et al., 2008), Figaro (Pfeffer, 2009), Factorie (McCallum, Schultz, and Singh, 2009), IBAL (Pfeffer, 2001) and the probabilistic logic programming languages described in Section 5.1.3. Some of these languages support the modeling of problems with an unknown number of objects or with identity uncertainty.

5.2 Ground Inference for SRL Models

The expressivity of statistical relational models comes at a price: designing efficient inference algorithms for these models is challenging. In this section, we discuss three alternative approaches. The first two approaches reduce the inference task to an inference task on a propositional model. We distinguish between the case where this propositional model is a probabilistic graphical model and the case where it is a logical theory. The third approach, called lifted inference, attempts to generalize and extend propositional inference algorithm to work with first-order models.

5.2.1 Propositionalization to Probabilistic Graphical Models

Many statistical relational formalisms and probabilistic programming languages have a semantics in terms of a probabilistic graphical model that corresponds to the first-order model. It is a straightforward next step to also use these graphical models for inference (see Section 2.4 for an overview of applicable algorithms). This approach is sometimes called *knowledge-based model construction* (KBMC) (Haddawy, 1994). It refers to a program (declarative or imperative) that constructs a graphical model using the knowledge available to the program, usually in the form of a relational database. Examples of languages for which we can construct undirected graphical models are Markov logic, parfactor graphs and Factorie. All forms of first-order Bayesian networks, such as BLPs, can construct directed networks for inference.

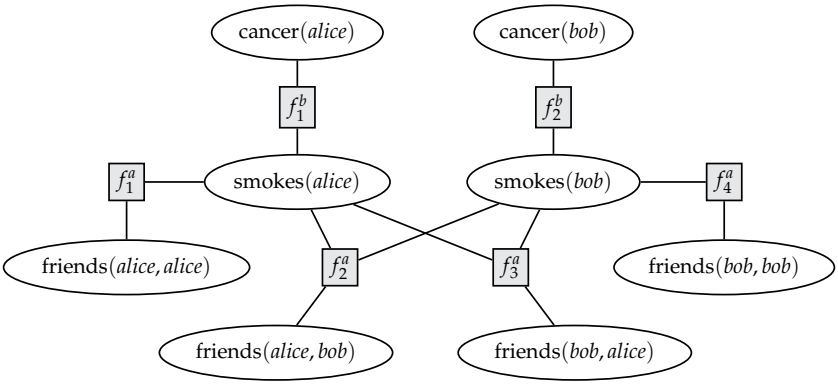


Figure 5.2: Factor graph constructed by KBMC for the MLN of Example 5.1

A common problem with this approach is that the constructed graphical models quickly become too large to admit efficient inference. There are ways of mitigating this problem. The size of the graphical model can be reduced by making the constructed network specific to the query and evidence (Shavlik and Natarajan, 2009; Meert, Taghipour, and Blockeel, 2010). Also with these improvements, exact inference quickly breaks down when using KBMC.

Even executing standard approximate inference algorithms can become a problem, because the networks being generated for first-order models that deal with a large number of objects may not fit into memory. The size of the constructed network is usually polynomial in the number of objects in the first-order model, but the degree of this polynomial can be high, causing the above problem.

Example 5.6. Applying KBMC to the MLN of Example 5.1 and the parfactor graph of Example 5.2 leads to the same factor graph model. With a domain of two people, *alice* and *bob*, these models correspond to the factor graph in Figure 5.2. It consists of eight random variables and six factors. These factors represent the ground formulas shown in Example 5.1. The potential of the f_i^a factors is given by the table in Example 5.2. For this network, it is possible to perform exact inference by KBMC. Now consider the case where we apply the same MLN to a domain with 100 people, leading to a propositional model with $100^2 + 100 = 10,100$ random variables and $100^2 = 10,000$ factors. KBMC methods can still construct this network, but exact inference becomes intractable. If we apply the MLN to a domain with 7 billion people, there are over 10^{19} random variables. Representing the network would take millions of terabytes

of memory. This shows how easy it is to have a concise first-order model (two weighted formulas) that leads to the construction of an enormous graphical model.

5.2.2 Propositionalization to Weighted Logic Theories

The recent success of applying logical inference algorithms such as weighted model counting to the problem of probabilistic inference (see Section 2.4) inspired a similar approach for inference in statistical relational models. Different from classical KBMC, these approaches do not construct a probabilistic graphical model but a weighted theory in propositional logic.

Chavira, Darwiche, and Jaeger (2006) reduce inference for relational Bayesian networks to a weighted model counting problem. They employ knowledge compilation to sd-DNNF for the construction of arithmetic circuits that can efficiently solve the weighted model counting task and thereby compute conditional probabilities in the original RBN (cf., Section 2.4.2).

In probabilistic logic programming, De Raedt, Kimmig, and Toivonen (2007) present an algorithm that uses compilation to binary decision diagrams to compute unconditional marginal probabilities (i.e., success probabilities of a query). They implicitly perform a reduction to a weighted model counting problem, even though they do not describe their approach as such. Fierens et al. (2011a) explicitly reduce the conditional probability query in PLP to a weighted model counting problem. Once a weighted logical theory is obtained, Fierens et al. use standard knowledge compilation techniques for exact inference. Intermezzo 3 discusses this work in more detail.

The advantage of these reductions to weighted logical inference, compared to classical KBMC, is that these algorithms naturally exploit the determinism and context-specific independencies in the first-order model (cf., Section 2.4.1). This permits efficient inference, also in models with high treewidth. Still, this approach can break down easily when the propositional theories being generated become too large. Fierens et al. report that their experiments generate weighted CNFs with close to a million clauses. Even performing approximate weighted model counting for these theories is challenging.

5.3 Different Notions of Lifted Inference

Performing inference in statistical relational models is extremely costly if it is done at the propositional level. This observation motivated a new line of research in *lifted probabilistic inference* algorithms (Poole, 2003). These algorithms exploit first-order structure and symmetries in statistical relational models during inference. They perform inference at a higher level of abstraction, by treating indistinguishable groups of objects as one.

The idea of exploiting symmetries during inference is not uncommon. It is applied in statistics, logical inference, mathematical programming and constraint satisfaction in general. We will review lifted inference approaches in those fields to motivate the general idea of lifted inference. Then we formally define a notion of exact lifted probabilistic inference, called domain-lifted inference. We defer the discussion on approximate lifted inference to Chapter 7.

5.3.1 Lifting in Statistics

We begin with a simple example from probabilistic reasoning.

Example 5.7. Assume we are investigating a rare disease, and the only information we have is that the disease presents itself in one in every billion people. The probability that somebody in the world has the disease is then computed as

$$1 - (1 - 1/1000000000)^{7000000000} \approx 0.999 \tag{5.6}$$

Here, we have used the fact that, for the level of detail we have knowledge about, all people are indistinguishable and independent. Therefore, we can exponentiate the probability that a single person is healthy to obtain the probability that all people are healthy. When encoding this distribution in a probabilistic graphical model, in the best case, inference for this query ends up multiplying 7 billion numbers. In the worst case, naively representing the query variable's dependency on each person's health requires a conditional probability table with $2^{7000000000}$ rows. In either case, despite the simplicity of the statistical problem, graphical model inference algorithms do not exploit the symmetries we have exploited in writing Equation 5.6.

The following example shows that not only people, but also possible worlds are symmetric for this problem.

Example 5.8. The probability that exactly five people have the disease is

$$\binom{7 \cdot 10^9}{5} \left(1 - 10^{-9}\right)^{7 \cdot 10^9 - 5} \left(10^{-9}\right)^5 \approx 0.13$$

The probability that more than five people have the disease is

$$1 - \sum_{n=0}^{5} \binom{7 \cdot 10^9}{n} \left(1 - 10^{-9}\right)^{7 \cdot 10^9 - n} \left(10^{-9}\right)^n \approx 0.7$$

Here, we have not only used the indistinguishability of all people in the exponentiation, but also in the binomial coefficients. The coefficient $\binom{7 \cdot 10^9}{n}$ counts the number of possible worlds where exactly n people have the disease. This generalization from one possible world to all of its symmetries is a form of lifted inference. It is not clear how one would encode the above problem in a probabilistic graphical model. Even if we use a concise representation of conditional probabilities, inference algorithms will likely have to enumerate all $\binom{7 \cdot 10^9}{5}$ possible worlds.

The following example illustrates how grouping indistinguishable objects into equivalence classes and reasoning about these groups of objects as a whole can speed up inference. This observation is essential to the development of lifted inference algorithms.

Example 5.9. If we now additionally know that the disease is more rare in women, presenting only in one in every two billion women and one in every billion men. Then, assuming there are 3.4 billion men and 3.6 billion women in the world, the probability that more than five people have the disease is

$$1 - \sum_{n=0}^{5} \sum_{f=0}^{n} \binom{3.6 \cdot 10^9}{f} \left(1 - 0.5 \cdot 10^{-9}\right)^{3.6 \cdot 10^9 - f} \left(0.5 \cdot 10^{-9}\right)^f$$
$$\times \binom{3.4 \cdot 10^9}{(n-f)} \left(1 - 10^{-9}\right)^{3.4 \cdot 10^9 - (n-f)} \left(10^{-9}\right)^{(n-f)}$$

Here, we have split up the computation in two factors, for women on the first line and men on the second. The variable n represents the number of people with the disease and the variable f counts the number of women among them.

In all these examples, our knowledge of the symmetries of the problem allow us to compute probabilities more efficiently. We have seen statistical problems whose textual description is very simple and that can be solved using elementary statistical techniques. However, no classical automated reasoning algorithms can apply the same techniques for general-purpose inference. They all fail to solve these simple problems.

5.3.2 Lifting in First-Order Logic

A second example of lifted inference is the *resolution* algorithm (Robinson, 1965) for automated theorem proving in first-order logic. This algorithm is the basis for many automated reasoning algorithms.

Example 5.10. Consider the following clauses

$$\forall X : \neg \text{human}(X) \vee \text{mortal}(X)$$

$$\forall X : \neg \text{greek}(X) \vee \text{human}(X),$$

which state that all humans are mortal and that Greeks are human. The clauses contain opposing literals that unify. This means that the resolution principle can be applied as follows.

$$\frac{\forall X : \neg \text{human}(X) \vee \text{mortal}(X), \qquad \forall X : \neg \text{greek}(X) \vee \text{human}(X)}{\forall X : \text{mortal}(X) \vee \neg \text{greek}(X)}$$

The resulting formula, or *resolvent*, stating that all Greeks are mortal, is a logical consequence of the initial theory. In a single step of automated reasoning, resolution has proven a property of all Greeks in the world. This was possible because, at the level of abstraction of the logical theory, all Greeks are indistinguishable.

Resolution can perform inference independent of how many objects there are in the world. It is even capable of reasoning about an infinite number of objects. To prove a theorem in graph theory, for example, an automated reasoning algorithm needs to reason about an infinite number of possible graphs of all possible sizes. This property makes first-order resolution so powerful and allows it to prove mathematical theorems.

Its ability for lifted inference was an important realization of first-order resolution. In his original paper on resolution, Robinson wrote what is now known as the *lifting lemma*.

Lemma 5.1 (Robinson's Lifting Lemma)**.** *Let γ_1 and γ_2 be two first-order clauses whose resolvent is γ. Let δ_1 be a ground instance of γ_1 and δ_2 a ground instance of γ_2. If δ_1 and δ_2 have a resolvent δ, then δ is a ground instance of γ.*

This lemma establishes the connection between propositional and first-order resolution, showing that one step of first-order resolution corresponds to a potentially infinite number of propositional resolution steps.

Example 5.11. The first-order resolution step of Example 5.10 corresponds to the propositional resolution steps

$$\frac{\neg\,\text{human}(\mathit{alice}) \vee \text{mortal}(\mathit{alice}), \qquad \neg\,\text{greek}(\mathit{alice}) \vee \text{human}(\mathit{alice})}{\text{mortal}(\mathit{alice}) \vee \neg\,\text{greek}(\mathit{alice})}$$

$$\frac{\neg\,\text{human}(\mathit{bob}) \vee \text{mortal}(\mathit{bob}), \qquad \neg\,\text{greek}(\mathit{bob}) \vee \text{human}(\mathit{bob})}{\text{mortal}(\mathit{bob}) \vee \neg\,\text{greek}(\mathit{bob})}$$

$$\vdots$$

Both ground resolvents are instances of the first-order resolvent.

The lifting lemma reflects the more general notion of lifted inference which says that a single lifted inference step performs multiple steps of some corresponding ground inference algorithm. Although this is not a precise definition of lifting, it has until recently been the sole argument for classifying algorithms as being "lifted".

5.3.3 Lifting in Constraint Satisfaction

Our final example of lifted inference is in the fields of constraint satisfaction and mathematical programming. These fields are concerned with finding assignments to a set of variables such that a set of constraints is satisfied, possibly also optimizing an objective function of the variables.

When permuting variables or values in the model leaves the structure of the model unchanged, the model is said to have symmetries. The presence of a large

number of these symmetric states can complicate inference, by unnecessarily increasing the size of the search space. A search algorithm potentially has to visit all symmetries of a state that does not satisfy the constraints before it finds a satisfying solution. Methods to remove symmetries add random perturbations to the model, fix certain variables or add constraints that are only satisfied in a subset of symmetric states. These are called symmetry breaking constraints.

Lifted algorithms along these lines were developed for mathematical programming (Margot, 2010; Liberti, 2012; Mladenov, Ahmadi, and Kersting, 2012), constraint satisfaction (Crawford et al., 1996; Cohen et al., 2005) and constraint programming (Gent and Smith, 2000; Fahle, Schamberger, and Sellmann, 2001).

Lifted inference by symmetry breaking is *fundamentally different* from the lifted inference examples outlined above and the lifted algorithms for probabilistic inference we will discuss next. In constraint satisfaction and mathematical programming, we are usually only interested in finding a single solution, even though there may exist many symmetric ones. Symmetry breaking techniques therefore do not keep track of which solutions are removed from the model. For these other inference tasks, we do want to keep track of symmetric solutions. For example, in resolution, we want the resolvent to imply all consequences of a pair of clauses. In lifted probabilistic inference, we also have to count the probability mass of symmetric possible worlds when estimating beliefs.

5.3.4 Domain-Lifted Probabilistic Inference

The concept of lifted inference is mostly introduced at an informal and intuitive level:

- "*...lifted, that is, deals with groups of random variables at a first-order level*" (de Salvo Braz, Amir, and Roth, 2005)
- "*The act of exploiting the high level structure in relational models is called lifted inference*" (Apsel and Brafman, 2011)
- "*The idea behind lifted inference is to carry out as much inference as possible without propositionalizing*" (Kisyński and Poole, 2009a)
- "*lifted inference, which deals with groups of indistinguishable variables, rather than individual ground atoms*" (Singla, Nath, and Domingos, 2010)

While the term lifted inference emerges as a quite coherent algorithmic metaphor, it is not immediately obvious what its precise formal meaning should be. Since a rich variety of different algorithmic approaches are collected under the label "lifted", and since most of them can degenerate for certain models to ground, or propositional, inference, it is difficult to precisely define the class of lifted inference techniques in algorithmic terms.

Definition

A more fruitful approach is to formalize the concept of lifted inference in terms of its objectives, rather than in terms of its algorithmic means. Here one observes that exact lifted inference techniques very consistently are evaluated on, and compared against each other, by how well inference complexity scales as a function of the domain for which the general model is instantiated. Empirical evaluations of lifted inference techniques are usually presented in the form of domain size vs. inference time plots, as shown in Figure 5.3.

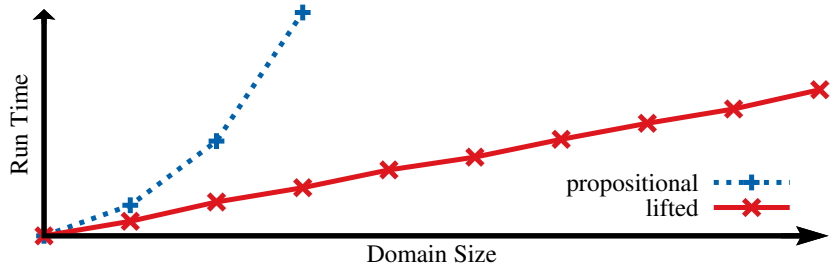


Figure 5.3: A typical performance evaluation of lifted inference algorithms

We therefore propose a formal definition of *domain-lifted inference* in terms of polynomial time complexity in the domain size parameter.

Definition 5.2 (Domain-Lifted Inference)**.** A probabilistic inference procedure is *domain-lifted* for a model Δ, query q and evidence e iff it computes $\Pr(q|e)$ in time *polynomial* in each $|D_1|, \ldots, |D_k|$, where D_i is the domain of the logical variable X_i appearing in Δ, q or e.

Discussion

The requirements for domain-lifted inference are quite *strict*. Any algorithm that saves some computations by avoiding a grounding step or exploiting a symmetry could conceivably be called lifted. However, it is difficult to precisely define such a notion of lifted inference, which is less strict than domain-lifted inference. Chapter 6 will define an alternative notion that is even more strict, also looking at complexity in terms of the size of the query and evidence.

On the other hand, for certain models, the requirements for domain-lifted inference are *too flexible*. There is the undesirable effect that many classical algorithms are domain-lifted for models that have a bounded treewidth with increasing domain size, such as models with singly connected groundings. Interestingly, on these models, lifted inference algorithms often have a complexity that is logarithmic in the domain size (Poole, Bacchus, and Kisyński, 2011). Hence, there is also an exponential complexity separation between lifted algorithms, and for instance variable elimination (VE). Therefore, one way to remove this undesirable effect is to require that lifted inference algorithms are exponentially faster than VE. The disadvantage of this definition is that it turns algorithms such as VE into a universal complexity baseline. Furthermore, algorithms that exploit local structure to attain exponential speedups over VE could by this alternative definition also be called lifted.

Domain-lifted inference is not the only notion one might have of exact lifted inference. Despite its drawbacks, it does formalize a useful common intuition. For example, first-order resolution can be regarded as a domain-lifted inference algorithms for 0/1 probabilities. Its complexity is independent of the number of objects in the world. Also note that the problem of precisely defining what lifted *approximate* inference could mean is still open.

5.4 Lifted Inference by Weighted Model Counting

Reduction to weighted model counting is a state-of-the-art approach for inference in probabilistic graphical models (Section 2.4.1) and statistical relational models (Section 5.2.2). We will first show how to lift the weighted model counting task on logical theories to the first-order level. We will then reduce inference in popular statistical relational models to this new task.

5.4.1 Weighted First-Order Model Counting

Weighted First-Order Model Counting (WFOMC) takes as input a sentence Δ in FOL (or FOL-DC) and two functions w_T and $w_F : \mathcal{P} \rightarrow \mathbb{R}_{\geq 0}$ that associates a non-negative weight with every predicate in the sentence ($\mathcal{P}$). Consider the interpretation $\omega = \omega_T \cup \omega_F$ consisting of positive literals ω_T and negative literals ω_F. The weight associated with this interpretation is $\prod_{l \in \omega_T} w_T(\text{pred}(l)) \prod_{l \in \omega_F} w_T(\text{pred}(l))$, where pred(.) maps a literal to its predicate symbol. These interpretation weights are aggregated into the weighted first-order model count of a sentence as follows.

Definition 5.3 (WFOMC)**.** The weighted first-order model count of sentence Δ with models $\mathcal{MOD}_\Delta$ is

$$\text{WFOMC}(\Delta, w_T, w_F) = \sum_{\omega \in \mathcal{MOD}_\Delta} \prod_{l \in \omega_T} w_T(\text{pred}(l)) \prod_{l \in \omega_F} w_F(\text{pred}(l)).$$

5.4.2 Reductions to Weighted First-Order Model Counting

We will now show that computing the partition function of a MLN or Parfactor graph can be reduced to a WFOMC task. This lifts the approach outlined in Section 2.4.1, which explains the reduction of the partition function computation for factor graphs to a propositional weighted model counting problem. Each reduction outputs a theory Δ and weight functions w_T and w_F, which are in turn the inputs to the WFOMC function.

Reducing Markov Logic Networks

Let us first convert an example MLN into a WFOMC problem.

Example 5.12. Consider the MLN of Example 5.1:

1.5 $\quad \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y)$

1.3 $\quad \text{smokes}(X) \Rightarrow \text{cancer}(X)$

Its corresponding WFOMC problem counts the models of the theory Δ:

$$\forall X, Y \in \text{People} : \theta_1(X, Y) \Leftrightarrow [\text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y)]$$

$$\forall X \in \text{People} : \theta_2(X) \Leftrightarrow [\text{smokes}(X) \Rightarrow \text{cancer}(X)]\,,$$

The weight function w_T maps $\text{w}_\text{T}(\theta_1)$ to $e^{1.5}$, $\text{w}_\text{T}(\theta_2)$ to $e^{1.3}$ and all other predicates to 1. The function w_F maps all predicates to 1.

In general, the WFOMC-reduction constructs a theory Δ using the predicates in the MLN and a set of additional predicates, here denoted by θ_i. The predicate θ_i represents the truth value of the ith MLN formula (w_i, ϕ_i), for each of its instances. Let $\mathbf{X}_i$ be the set of free variables in ϕ_i and let cs_i be the constraint set that enforces that each variable in $\mathbf{X}_i$ be a member of its domain (as defined by the MLN). The theory Δ then contains the formula $\forall \mathbf{X}_i, cs_i : \theta_i(\mathbf{X}_i) \Leftrightarrow \phi_i$. The weight function w_T maps each θ_i to e^{w_i} and all other predicates to 1. The function w_F maps all predicates to 1.

Proposition 5.2. *The partition function Z of an MLN equals the weighted first-order model count* $\text{WFOMC}(\Delta, \text{w}_T, \text{w}_F)$.

Proof outline. Each possible world of the MLN corresponds to exactly one interpretation of Δ. When the possible world contains n_i true groundings of formula ϕ_i, the weight of this possible world according to Equation 5.1 is $\exp\left(\sum_i^{|\Psi|} w_i n_i(\omega)\right)$. In the weighted model counting problem, the corresponding interpretation will contain exactly n_i positive θ_i literals, which increase the weight of the interpretation by a factor $\text{w}_\text{T}(\theta_i) = e^{w_i}$. Hence, the weight of an interpretation equals the weight of its corresponding possible world. □

Reducing Parfactor Graphs

A similar reduction is possible for parfactor graphs by reducing each parfactor $(\mathbf{X}, cs, A, f)$ individually. Here we have to distinguish between Boolean parfactors, where the atoms range over truth values, and general parfactors, where atoms may be assigned elements from any finite range of values.

In the first case, the parfactor can be reduced to a single weighted formula per row of the potential table f. The row that assigns potential value $f(\text{T}, \ldots, \text{F})$ to the state $a_1 = \text{T}, \ldots a_n = \text{F}$ is reduced to the formula $\forall \mathbf{X}, cs : \theta(\mathbf{X}) \Leftrightarrow$

$a_1 \wedge \cdots \wedge \neg a_n$ and the weight functions map $\mathrm{w_T}(\theta)$ to $f(\mathrm{T}, \ldots, \mathrm{F})$ and $\mathrm{w_F}(\theta)$ to 1. The weights of all original parfactor predicates are set to 1.

In the second case, the multi-valued atoms in A need to be encoded in Boolean logic. We can represent the atom $\mathrm{p}(\mathbf{X})$ taking on value v with the atom $\mathrm{p}(\mathbf{X}, v)$. First, we encode in Δ that the atom must take on exactly one value from its range R:

$$\forall \mathbf{X}, cs : \exists Y \in \mathsf{R} : \quad \mathrm{p}(\mathbf{X}, Y)$$

$$\forall \mathbf{X}, cs : \forall Y, Z \in \mathsf{R}, Y \neq Z : \neg\, \mathrm{p}(\mathbf{X}, Y) \vee \neg\, \mathrm{p}(\mathbf{X}, Z).$$

With this addition, the reduction is similar to the reduction of Boolean parfactors. The row for state $\mathrm{p}_1(\mathbf{X}_1) = v_1, \ldots \mathrm{p}_n(\mathbf{X}_n) = v_n$ with potential $f(v_1, \ldots, v_n)$ is reduced to the formula $\forall \mathbf{X}, cs : \theta(\mathbf{X}) \Leftrightarrow \mathrm{p}_1(\mathbf{X}_1, v_1) \wedge \cdots \wedge \mathrm{p}_n(\mathbf{X}_n, v_n)$ and the weight functions map $\mathrm{w_T}(\theta)$ to $f(v_1, \ldots, v_n)$ and $\mathrm{w_F}(\theta)$ to 1.

We can show that the partition function of the parfactor graph again corresponds to the weighted model count $\mathrm{WFOMC}(\Delta, \mathrm{w_T}, \mathrm{w_F})$.

Reducing Probabilistic Logic Programs

Clark's *completion* (Clark, 1978) is a transformation from logic programs to first-order logic. For certain classes of logic programs, called *tight* logic programs (Fages, 1994), this transformation is correct, in the sense that every model of the logic program is a model of the completion, and vice versa.

Example 5.13. The Prolog rules in Example 5.4 have the following completion.

$$\forall X, Y \in \mathsf{N} : \mathrm{path}(X, Y) \Leftrightarrow [\mathrm{edge}(X, Y) \vee (\exists Z \in \mathsf{N} : \mathrm{edge}(X, Z) \wedge \mathrm{path}(Z, Y))], \tag{5.7}$$

where N is the set of nodes in the probabilistic graph. In case the edge/2 relation is well-founded (Van Gelder, Ross, and Schlipf, 1991), when the probabilistic network is acyclic, the models of the completion correspond one-to-one to the models of the logic program.

We can reduce inference in tight probabilistic logic programs into a weighted first-order model counting problem. For ProbLog programs, the reduced logical

theory Δ is obtained by taking the completion of the rules $\mathcal{R}$ conjoined with a single formula per probabilistic fact in $\mathcal{F}$. The reduction of probabilistic facts is similar to the reduction of unit clauses in Markov logic. The fact ϕ_i with probability p_i and free variables $\mathbf{X}_i$ is represented by the formula $\forall \mathbf{X}, cs_i : \theta_i(\mathbf{X}_i) \Leftrightarrow \phi_i$. The associated weight function sets all $w_T(\theta_i) = p_i$, $w_F(\theta_i) = 1 - p_i$ and all other weights to 1. This reduction to a weighted FOL-DC theory depends on the existence of a finite domain for each logical variable, as defined by the constraint set cs_i. One can obtain this domain by exhaustively executing the ProbLog program as a deterministic Prolog program and keeping track of the goals that are called during resolution.

5.4.3 Computing Marginal and Conditional Probabilities

The weighted model counting approach to inference is not restricted to computing partition functions. As for the propositional case (Section 2.4.1), it can be used to compute marginal and conditional probabilities.

Assuming we have obtained a weighted FOL-DC theory Δ with weight functions w_T and w_F, the marginal probability $\Pr(q)$ can be computed as $\text{WFOMC}(\Delta \wedge q, w_T, w_F) / \text{WFOMC}(\Delta, w_T, w_F)$. Conditional probabilities can be computed as

$$\Pr(q|e) = \frac{\Pr(q \wedge e)}{\Pr(e)} = \frac{\text{WFOMC}(\Delta \wedge q \wedge e, w_T, w_F)}{\text{WFOMC}(\Delta \wedge e, w_T, w_F)}. \tag{5.8}$$

We will now present a very basic form of lifted inference of marginal and conditional probabilities, by exploiting equiprobability. Even though this approach does not conform to the definition of domain-lifted inference, it can save many redundant computations, even when applying ground inference techniques to compute the weighted model counts in Equation 5.8.

Exploiting Equiprobability

When answering a large number of queries in a single model, we can exploit the knowledge that certain random variables are equiprobable.

Definition 5.4 (Equiprobable Set)**.** A set of random variables V is called equiprobable w.r.t. distribution Pr and evidence e iff for all $v_1, v_2 \in V$: $\Pr(v_1|e) = \Pr(v_2|e)$.

Intermezzo 3: Inference in Probabilistic Logic Programs by Weighted Model Counting and Max-SAT

Fierens, Van den Broeck, Thon, Gutmann, and De Raedt (2011a) propose an approach to inference for probabilistic logic programs by reducing conditional probability queries to weighted model counting tasks and MPE queries to weighted Max-SAT problems. Compared to existing PLP inference algorithms, this approach supports answering of *multiple* queries *with evidence*.

These reductions have their first step in common: to construct an equivalent weighted sentence in propositional logic. For *tight* ProbLog programs, Example 5.13 explained how to reduce a PLP to an equivalent sentence in FOL. We can directly obtain a weighted propositional sentence by grounding the first-order sentence (assuming a finite Herbrand universe). For example, the grounding of Formula 5.7 (for the Prolog rules of Example 5.4) includes

$$\text{path}(a,z) \Leftrightarrow [\text{edge}(a,z) \vee (\text{edge}(a,b) \wedge \text{path}(b,z))$$
$$\vee(\text{edge}(a,c) \wedge \text{path}(c,z)) \vee \dots]$$

The probabilistic facts `p :: f` in the ProbLog program are represented by the corresponding weight function. Each grounding `g` of `f` maps to p and each $\neg$`g` maps to $1 - p$. All other literals get weight 1.

After this reduction to a weighted propositional theory, standard inference algorithms can be used to compute conditional probabilities by weighted model counting and most probable explanations by off-the-shelf weighted Max-SAT solvers. For example, exact probabilities and MPE can be computed by knowledge compilation to d-DNNF (cf., Section 2.4.2) and approximate probabilities can be computed using MC-SAT (Poon and Domingos, 2006).

For more general logic programs programs, however, there exists no equivalent first-order sentence and the transformation to propositional logic becomes more challenging. This is due to the fact that PLPs are more expressive than other SRL formalisms that extend FOL: they can express inductive definitions, transitive closure, etc. (Fierens et al., 2012a). Transforming inductive definitions to propositional logic involves a change of semantics, from well-founded semantics (Van Gelder, Ross, and Schlipf, 1991) to the semantics of classical logic, which complicates the reduction.

The transformation used by Fierens et al. is query- and evidence- specific and employs techniques from answer set programming (Janhunen, 2004) in order to remove positive loops from the logic program. It can lead to an explosion of the size of the model: the size of the weighted logical theory can be exponential in the size of the first-order model and the domain size, whereas for the KBMC approaches outlined in Section 5.2, the size is always polynomial.

Fierens et al. (2012b) also apply this approach to *learning* the parameters of probabilistic logic programs from (partial) interpretations.

After partitioning the set of all random variables into equiprobable sets, it suffices to compute the probability of a single random variable in each element of the partition to obtain the probability of all variables.

In general, partitioning into equiprobable sets is hard, even when there is no evidence.

Proposition 5.3 (Niepert (2012b))**.** *Finding the smallest partition of random variables into equiprobable sets for a given distribution is NP-hard.*

Proof. Deciding satisfiability of a Boolean formula ϕ can be reduced to finding equiprobable sets. The MLN representing $(\phi \vee v_1) \wedge v_2$ induces a probability distribution. If ϕ is unsatisfiable, then in the smallest partition (assuming no evidence), v_1 and v_2 are in the same equiprobable set (with probability 1). If ϕ is satisfiable, there is a non-zero probability that v_1 is false, and v_1 and v_2 must be in different equiprobable sets. □

Despite this, the preemptive shattering operation outlined in Section 4.3.1 gives us a tool to group together equiprobable random variables (ground atoms) on a purely syntactic basis. However, the obtained partition will not be minimal.

To find equiprobable sets for a weighted FOL-DC theory Δ and evidence e, we will partition the atoms for each predicate separately. Consider the predicate $\mathrm{p}(\mathbf{X})$ whose arguments can take on values defined by the constraint set cs. Let K be the set of constants appearing as logical terms in $\Delta \wedge e$, let V be the set of free logical variables in $\Delta \wedge e$ and let $\mathcal{D}$ be the set of domain terms in $\Delta \wedge e$. Consider then the set of constrained atoms returned by the following call to the preemptive shattering procedure (Algorithm 8):

$$\{\gamma_1, \ldots, \gamma_n\} = \textsc{ShatterClause}((\forall \mathbf{X}, cs : \mathrm{p}(\mathbf{X})), K \cup V, \mathcal{D})$$

Each γ_i represents a disjoint set of ground atoms $\mathrm{gr}(\gamma_i)$.

Proposition 5.4. $\{\mathrm{gr}(\gamma_1), \ldots, \mathrm{gr}(\gamma_n)\}$ *is a partition of the atoms* $\mathrm{gr}((\forall \boldsymbol{X}, cs : \mathrm{p}(\boldsymbol{X})))$ *into equiprobable sets.*

Proof. Consider two groundings $\gamma_i\theta_1$ and $\gamma_i\theta_2$ of the same shattered constrained atom. For these ground atoms to be equiprobable (i.e., $\Pr(\gamma_i\theta_1|e) = \Pr(\gamma_i\theta_2|e)$), from Equation 5.8, we need to prove that $\mathrm{WFOMC}(\Delta \wedge \gamma_i\theta_1 \wedge e, \mathrm{w_T}, \mathrm{w_F}) = \mathrm{WFOMC}(\Delta \wedge \gamma_i\theta_2 \wedge e, \mathrm{w_T}, \mathrm{w_F})$. It follows from Lemma 4.3 that $\Delta \wedge \gamma_i\theta_1 \wedge e$

and $\Delta \wedge \gamma_i\theta_2 \wedge e$ are equivalent up to a permutation of constants. Hence, there exists a permutation that maps each model of the one into a model of the other. Furthermore, because each pair of mapped models has the same number of true and false atoms for each predicate, their weights are identical. Therefore, the weighted model counts of both theories are the same. □

In the best case, preemptive shattering of a predicate returns one atom and the induced equiprobable partition has one element. In that case, we can compute the probability of all groundings of the predicate by computing the probability of a single grounding. Since the number of ground atoms for a predicate is a polynomial of the domains of the logical variables, exploiting equiprobability reduces the amount of work by a *factor polynomial in the domain size*. Therefore, exploiting equiprobability does not qualify as domain-lifted inference (there is no exponential speedup). Even so, it does often make a significant practical difference.

5.5 Lifted Inference by Knowledge Compilation

We have established that inference in statistical relational models can be reduced to a weighted first-order model counting problem. However, we did not present algorithms for actually solving this new task. One way would be to propositionalize the problem, using the grounding function $\mathrm{gr}(.)$ and use ground weighted model counting solvers (Section 2.4.1). Knowledge compilation to `sd-DNNF` (Section 2.4.2) is an example of such a solver. This section presents a lifted analogue of that approach, which performs first-order knowledge compilation to `FO-sda-DNNF`.

5.5.1 Weighted Model Count of a `FO-sda-DNNF` Circuit

As propositional `sd-DNNF` circuits, `FO-sda-DNNF` circuits permit efficient weighted model counting. The following function computes the weighted first-order model count of a given `FO-sda-DNNF` sentence and weight functions $\mathrm{w_T}$ and $\mathrm{w_F}$.

Definition 5.5. The base cases of the function are $\mathrm{WFOMC}(\mathrm{F}, \mathrm{w_T}, \mathrm{w_F}) = 0$, $\mathrm{WFOMC}(\mathrm{T}, \mathrm{w_T}, \mathrm{w_F}) = 1$, $\mathrm{WFOMC}(l, \mathrm{w_T}, \mathrm{w_F}) = \mathrm{w_T}(\mathrm{pred}(l))$ when l is a *positive ground literal* and $\mathrm{WFOMC}(l, \mathrm{w_T}, \mathrm{w_F}) = \mathrm{w_F}(\mathrm{pred}(l))$ when l is a *negative*

ground literal. The recursive definition is as follows. For an *extensional conjunction*, $\text{WFOMC}(\phi_1 \wedge \cdots \wedge \phi_n, \text{w}_\text{T}, \text{w}_\text{F})$ equals

$$\text{WFOMC}(\phi_1, \text{w}_\text{T}, \text{w}_\text{F}) \times \cdots \times \text{WFOMC}(\phi_n, \text{w}_\text{T}, \text{w}_\text{F}).$$

For an *extensional disjunction*, $\text{WFOMC}(\phi_1 \vee \cdots \vee \phi_n, \text{w}_\text{T}, \text{w}_\text{F})$ equals

$$\text{WFOMC}(\phi_1, \text{w}_\text{T}, \text{w}_\text{F}) + \cdots + \text{WFOMC}(\phi_n, \text{w}_\text{T}, \text{w}_\text{F}).$$

For an *intensional operator* $(\forall, \exists)\mathbf{X}, cs : \phi$ over a set of *logical* variables $\mathbf{X}$, with some $\theta \in \text{solutions}(cs, \mathbf{X})$ and $n = |\text{solutions}(cs, V)|$,

$$\text{WFOMC}((\forall \mathbf{X}, cs : \phi), \text{w}_\text{T}, \text{w}_\text{F}) = \text{WFOMC}(\phi\theta, \text{w}_\text{T}, \text{w}_\text{F})^n$$

$$\text{WFOMC}((\exists \mathbf{X}, cs : \phi), \text{w}_\text{T}, \text{w}_\text{F}) = n \times \text{WFOMC}(\phi\theta, \text{w}_\text{T}, \text{w}_\text{F}).$$

For an *intensional operator* $(\forall, \exists)\text{D}, cs : \phi$ over a *domain* variable D, with $\Theta_n = \{[\text{D}/d] \mid [\text{D}/d] \in \text{solutions}(cs, \text{D}) \wedge (|d| = n)\}$ and with $\theta_n \in \Theta_n$,

$$\text{WFOMC}((\forall \text{D}, cs : \phi), \text{w}_\text{T}, \text{w}_\text{F}) = \prod_n \text{WFOMC}(\phi\theta_n, \text{w}_\text{T}, \text{w}_\text{F})^{|\Theta_n|}$$

$$\text{WFOMC}((\exists \text{D}, cs : \phi), \text{w}_\text{T}, \text{w}_\text{F}) = \sum_n |\Theta_n| \times \text{WFOMC}(\phi\theta_n, \text{w}_\text{T}, \text{w}_\text{F}).$$

The WFOMC-function is similar to, and has the same complexity as the count-function (Definition 3.33) for solving the model counting task.

5.5.2 A Domain-Lifted Inference Algorithm

We now have all the tools that together form an exact lifted probabilistic inference algorithm, executing the following steps:

1. Reduce the statistical relational model to a weighted first-order model counting problem with theory Δ and weight functions w_T and w_F (see Section 5.4.2)
2. For the query $\Pr(q|e)$, compile two `FO-sda-DNNF` circuits, $C_{qe} = \textsc{Compile}(\Delta \wedge q \wedge e)$ and $C_e = \textsc{Compile}(\Delta \wedge e)$ (see Chapter 4)

3. Compute the required weighted model counts on the compiled circuits (see Section 5.5.1):

$$\Pr(q|e) = \frac{\text{WFOMC}(C_{qe}, \text{w}_\text{T}, \text{w}_\text{F})}{\text{WFOMC}(C_e, \text{w}_\text{T}, \text{w}_\text{F})}. \tag{5.9}$$

We will refer to this algorithm as *probabilistic inference by first-order knowledge compilation*.

An important property of the WFOMC-function on `FO-sda-DNNF` circuits is that it has a polynomial complexity in the size of the circuit node labels, and therefore in the size of the sets of constants, or domains, in these labels. This establishes the following result.

Theorem 5.5. *Probabilistic inference by first-order knowledge compilation is a domain-lifted inference algorithm for those inference tasks it can solve without grounding.*

Proof. The reduction from statistical relational models to a weighted first-order model counting problem is polynomial in the domain sizes of the logical variables. The compilation algorithm (Algorithm 1) has a complexity independent of the domain sizes (they can even be free variables during compilation). Finally, the evaluation of these circuits with the WFOMC-function is polynomial in the domain sizes. □

Note that domain-lifted inference is always defined w.r.t. a class of inference tasks. In Chapter 6, we will define the notion of completeness and identify classes of models and inference tasks for which probabilistic inference by first-order knowledge compilation is domain-lifted.

5.5.3 Conditional Probabilities

The main motivation for using knowledge compilation is the ability to reuse compiled circuits to answer multiple queries. Section 2.2.2 explained that in propositional knowledge compilation, any circuit can be conditioned on a term. Section 2.4.2 explained how to use that ability to perform probabilistic inference. With conditioning, we can evaluate all conditional probabilities $\Pr(q|e)$ on the same compiled circuit, assuming that q and e are terms. This section investigates whether the same applies to probabilistic inference by *first-order* knowledge compilation.

Conditioning on Propositions

Proposition 3.19 states that a FO-sda-DNNF circuit can efficiently be conditioned on propositions, that is, literals with arity zero. We can exploit this in our lifted inference algorithm. Assume we have a set of queries $\Pr(q_i|e_i)$ which only differ in the propositions contained in the terms q_i and e_i, that is $q_i = q^{>0} \wedge q_i^0$ and $e_i = e^{>0} \wedge e_i^0$. Here γ^i denotes all literals of arity i in the term γ and $\gamma^{>i}$ denotes all literals of arity higher than i. For each of these queries, our vanilla inference algorithm compiles circuits for $\Delta \wedge q_i \wedge e_i$ and $\Delta \wedge e_i$.

We can answer all queries above by compiling only two circuits, $C_{qe} = \textsc{Compile}\left(\Delta \wedge q^{>0} \wedge e^{>0}\right)$ and $C_e = \textsc{Compile}\left(\Delta \wedge e^{>0}\right)$. The requested posterior probabilities can then be computed as

$$\Pr\left(q_i|e_i\right) = \frac{\text{WFOMC}\left(\left(C_{qe}|\left(q_i^0 \wedge e_i^0\right)\right), \mathrm{w_T}, \mathrm{w_F}\right) \cdot \prod_{l \in q_i^0} \text{weight}(l)}{\text{WFOMC}\left(\left(C_e|e_i^0\right), \mathrm{w_T}, \mathrm{w_F}\right)}$$

where $\text{weight}(l)$ is $\mathrm{w_T}(\mathrm{p}(l))$ when l is a positive literal and $\mathrm{w_F}(\mathrm{p}(l))$ when l is a negative literal.

Conditioning on Unary Relations

Open Problem 2 states that it is unknown wheter a FO-sda-DNNF circuit can efficiently be conditioned on unary relations, that is, literals with arity one.

However, by modifying the weighted first-order model counting theory Δ with additional domain variables, it is possible to condition on any unary relation term. The applicability of this transformation is independent of the domains the theory is evaluated on and the specific terms on which it will be conditioned. In order to support conditioning for a unary predicate $\mathrm{p}/1$, each clause it appears in must be split into three clauses, representing the cases when a grounding of $\mathrm{p}/1$ is true, false or unknown. We add a domain variable that partitions the domain of a unary literal's logical variable into three sets: true, false or unknown. The compilation algorithm of Chapter 4 allows these domains to be free variables.

Proposition 5.6. *Any theory Δ in* FOL-DC *can be transformed into an equivalent representation Δ' that allows for conditioning on a unary relation $\mathrm{p}(X)$. First, each clause containing $\mathrm{p}(X)$ is split into three copies with additional constraints: (i) $X \in D_T$ (ii) $X \in D_F$ and (iii) $X \notin D_T$, $X \notin D_F$. For clauses with a positive $\mathrm{p}(X)$ literal, copy*

(i) is removed and the literal $\mathrm{p}(X)$ *is removed from copy (ii). Conversely, for clauses with a negative* $\mathrm{p}(X)$ *literal, copy (ii) is removed and* $\neg\,\mathrm{p}(X)$ *is removed from copy (i).*

Conditioning Δ on a term $\gamma = (\mathrm{p}(x_1) \wedge \cdots \wedge \mathrm{p}(x_n) \wedge \neg\,\mathrm{p}(y_1) \wedge \cdots \wedge \neg\,\mathrm{p}(y_m))$ is performed by substituting the newly introduced domain variables in Δ':

$$\Delta|\gamma \equiv \Delta'[D_\mathrm{T}/\{x_1, \ldots, x_n\}, D_\mathrm{F}/\{y_1, \ldots, y_m\}].$$

Example 5.14. To illustrate the procedure, consider the following clause, which omits constraints on Y for readability:

$$\forall X \in D : \mathrm{p}(X) \vee \mathrm{q}(X, Y) \tag{5.10}$$

We now introduce the subdomain $D_\mathrm{T} \subseteq D$ of constants for which we condition on $\mathrm{p}(X)$ being true, and the subdomain $D_\mathrm{F} \subseteq D$ of constants for which we condition on $\mathrm{p}(X)$ being false. Knowing that $D_\mathrm{T} \cap D_\mathrm{F} = \emptyset$, this divides the clause into

$$\forall X, X \in D \wedge X \in D_\mathrm{T} : \mathrm{p}(X) \vee \mathrm{q}(X, Y)$$

$$\forall X, X \in D \wedge X \in D_\mathrm{F} : \mathrm{p}(X) \vee \mathrm{q}(X, Y)$$

$$\forall X, X \in D \wedge X \notin D_\mathrm{T} \wedge X \notin D_\mathrm{F} : \mathrm{p}(X) \vee \mathrm{q}(X, Y)$$

Since $\mathrm{p}(X)$ appears as a positive literal in the initial formula, and we know that $\forall X \in D_\mathrm{T} : \mathrm{p}(X)$ and $\forall X \in D_\mathrm{F} : \neg\,\mathrm{p}(X)$ are true, these can be simplified to

$$\forall X, X \in D \wedge X \in D_\mathrm{F} : \mathrm{q}(X, Y) \tag{5.11}$$

$$\forall X, X \in D \wedge X \notin D_\mathrm{T} \wedge X \notin D_\mathrm{F} : \mathrm{p}(X) \vee \mathrm{q}(X, Y) \tag{5.12}$$

Substituting $[D_\mathrm{T}/\emptyset, D_\mathrm{F}/\emptyset]$ fills in empty domains in the constraint sets, which makes Clause 5.11 and the additional constraints $X \notin D_\mathrm{T} \wedge X \notin D_\mathrm{F}$ in Clause 5.12 trivially satisfied, thereby recovering the original Clause 5.10. However, filling in non-empty domains for D_T or D_F, conditions the theory on a term of literals of $\mathrm{p}/1$. Note that the atoms $\forall X \in D_\mathrm{T} : \mathrm{p}(X)$ and $\forall X \in D_\mathrm{F} : \mathrm{p}(X)$ which we conditioned on are removed from the new theory entirely, in the sense that they do not appear in its grounding.

The procedure outlined in Proposition 5.6 can be repeated to support conditioning on multiple unary relations. In summary, it is possible to condition

FO-d-DNNF circuits on any term of unary literals. However, this comes at the cost of a more complex compilation and a larger circuit.

The implications of this for our lifted inference algorithm are as follows. Assuming that applying the procedure of Proposition 5.6 to the theory Δ results in Δ', we can now answer all queries $\Pr(q_i|e_i)$ that differ only in propositions and unary relations, that is $q_i = q^{>1} \wedge q_i^{0,1}$ and $e_i = e^{>1} \wedge e_i^{0,1}$. This is done by compiling two circuits, for $C_{qe} = \textsc{Compile}\left(\Delta' \wedge q^{>1} \wedge e^{>1}\right)$ and $C_e = \left(\Delta' \wedge e^{>1}\right)$ and computing weighted model counts as

$$\Pr(q_i|e_i) = \frac{\text{WFOMC}\left(\left(C_{qe}\middle|\left(q_i^{0,1} \wedge e_i^{0,1}\right)\right), \text{w}_\text{T}, \text{w}_\text{F}\right) \cdot \prod_{l \in q_i^{0,1}} \text{weight}(l)}{\text{WFOMC}\left(\left(C_e\middle|e_i^{0,1}\right), \text{w}_\text{T}, \text{w}_\text{F}\right)} \tag{5.13}$$

This means that we can compile a single circuit to compute posterior probabilities for all propositions and unary atoms in our distribution.

5.6 Related Work

Probabilistic inference by weighted first-order model counting is based on the propositional *weighted model counting* approach to inference in probabilistic graphical models (Sang, Beame, and Kautz, 2005; Chavira and Darwiche, 2008) and statistical relational models (Chavira, Darwiche, and Jaeger, 2006; Fierens et al., 2011a). Independent of our work, Gogate and Domingos (2011) proposed a similar reduction for inference in Markov logic networks, to what they call *lifted weighted model counting*. Our first-order knowledge compilation approach to weighted model counting lifts the approach of Chavira and Darwiche (2008) to the first-order case.

We will now briefly review existing *exact* lifted probabilistic inference approaches. Approximate lifted inference algorithms are discussed in Chapter 7. See Kersting (2012) for a more general overview.

We note here that there exist lifted versions of the belief propagation algorithm (Jaimovich, Meshi, and Friedman, 2007; Singla and Domingos, 2008; Kersting, Ahmadi, and Natarajan, 2009). On models whose factor graph representation is singly connected, these algorithms are exact. Whenever

faced with a loopy theory, these are approximate algorithms. Moreover, the approximation provided by loopy belief propagation has been shown to deviate from the true distribution in presence of short loops containing near-deterministic dependencies, a case which can happen frequently in statistical relational models. We defer the discussion on lifted belief propagation to Chapter 7.

5.6.1 First-Order Variable Elimination

Poole (2003) posed the problem of lifted inference for parfactor models. He proposed a first solution which lifts the variable elimination algorithm. This lead to a line of work on first-order variable elimination (FOVE) (Poole, 2003; de Salvo Braz, Amir, and Roth, 2005; de Salvo Braz, Amir, and Roth, 2006; Milch et al., 2008; Choi, Amir, and Hill, 2010; Apsel and Brafman, 2011; Taghipour et al., 2012; Taghipour and Davis, 2012). The structured variable elimination algorithm (Pfeffer et al., 1999; Pfeffer, 1999) is an important precursor to FOVE.

Our approach differs from these types of methods in two significant ways. First, we take a logical, model theoretic approach to inference, whereas these existing algorithms lift a probabilistic inference algorithm. Second, our algorithm exploits the local structure, context specific independences and determinism that are abundant in statistical relational models. This can substantially improve the efficiency of inference, as we know from the literature on probabilistic graphical models (Zhang and Poole, 1999). This disadvantage of FOVE is mitigated by the work of (Kisyński and Poole, 2009b; Choi, de Salvo Braz, and Bui, 2011) who only exploit a very specific type of local structure that appears in so-called *aggregate factors*, such as "noisy-or".

5.6.2 Lifted Inference by Search

The approaches of Jha et al. (2010), Gogate and Domingos (2010) and Gogate and Domingos (2011) are more similar in spirit to our approach. Together with the work of Poole, Bacchus, and Kisyński (2011), they have in common that they perform lifted inference by search.

Jha et al. (2010) reduce the problem of computing the partition function of an MLN to finding the generating function of a *counting program*. Their approach focuses on the case when inference can be done fully on the lifted level. They

identify a limited class of models where tractable lifted inference is possible. However, this class is not defined declaratively. It is implicitly the class of models whose counting program can be evaluated using their inference rules. In addition, they do not allow the same predicate to appear more than once in a clause.

Lifted AND/OR search (Gogate and Domingos, 2010) approaches the problem from a different angle and uses weighted constraints instead of a weighted model counting formulation. The *Probabilistic Theorem Proving* (PTP) (Gogate and Domingos, 2011) algorithm is similar to lifted AND/OR search but recasts the problem into weighted first-order model counting, which they call lifted weighted model counting. These approaches are based on logical inference and constraint satisfaction and hence also exploit local structure.

The relation between lifted AND/OR search, PTP and first-order knowledge compilation was discussed in Section 4.7.2. An important difference noted there was that these methods do not allow free variables or domain constraints in their inputs. First-order knowledge compilation is capable of reusing circuits to answer queries about different domains. In contrast, each step of AND/OR search and PTP is specific to one domain. The expressivity of domain constraints allows us to efficiently condition on unary evidence. This technique does not apply to AND/OR search and PTP because they do not support domain constraints. The same holds for the parameters of the probability distribution. In first-order knowledge compilation, these are external to the compiled circuit and only used during evaluation. For example, we can reuse the same circuit for inference with different sets of MLN weights. We will see applications of this in Chapters 7 and 8. In contrast, the parameters of the model influence each step of the AND/OR search and PTP algorithms.

Another method related to our approach is *search-based* lifted inference (Poole, Bacchus, and Kisyński, 2011), also developed in parallel research. This approach uses a parfactor graph model, in contrast to our model counting formulation. Search-based inference is then performed on this model by lifting the method of *recursive conditioning* (Darwiche, 2001c). The similarities between (propositional) recursive conditioning and weighted model counting manifest themselves also in the lifted level between search-based lifted inference and our approach. Also this method does not use a compiled structure.

5.7 Experiments

To complement the theoretical results of the previous sections, this section reports on some empirical results. We obtained these results by implementing our first-order knowledge compilation approach to probabilistic inference in the WFOMC tool.[3]

The experiments are set up to investigate the following questions:

(Q1) How does WFOMC compare to *first-order variable elimination*?

(Q2) How does WFOMC *scale* with domain size?

(Q3) How does probabilistic inference by *propositional* knowledge compilation compare to *first-order* knowledge compilation?

(Q4) Does first-order knowledge compilation still pay off when the compilation algorithm has to *ground* the model during inference?

(Q5) How does computing posterior probabilities by circuit reuse and logical *conditioning* on evidence (Equation 5.13) compare to the approach that compiles a circuit that includes all evidence (Equation 5.9)?

5.7.1 Marginal Probabilities

Methodology

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, Amir, and Roth, 2005) and *friends, smokers and drinkers* (Van den Broeck et al., 2011a). We investigate the task of computing unconditional marginal probabilities and compare the performance of WFOMC with counting first-order variable elimination (C-FOVE) (Milch et al., 2008) and propositional variable elimination (VE) (Zhang and Poole, 1994; Dechter, 1996). We used the publicly available Java implementations of C-FOVE and VE in the BLOG system.[4]

[3] http://dtai.cs.kuleuven.be/wfomc/
[4] http://people.csail.mit.edu/milch/blog/

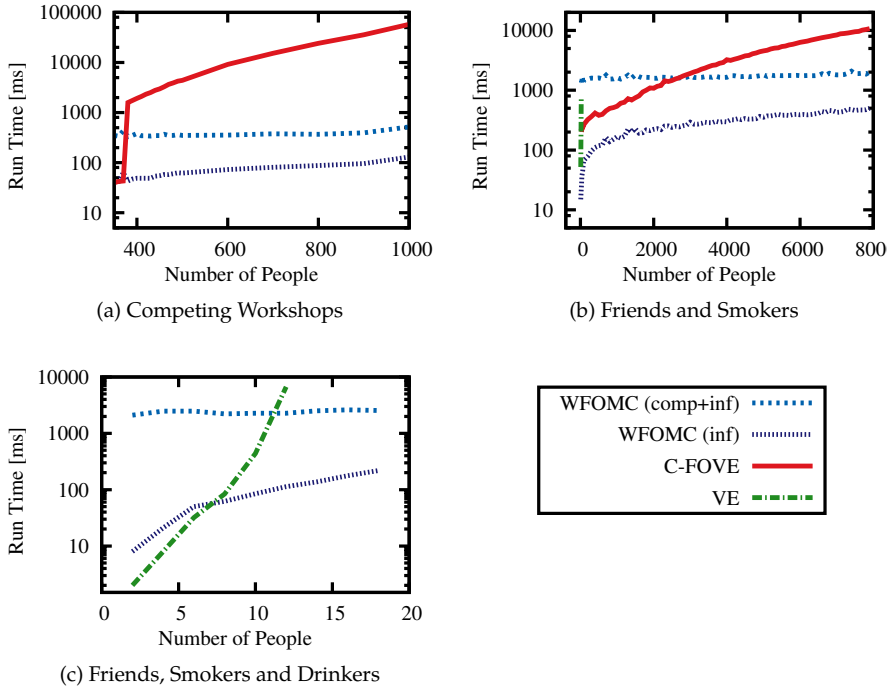


Figure 5.4: Run time comparison of WFOMC, C-FOVE and VE.

Results

Figure 5.4 contains representative results on three different tasks. It plots how inference time varies with the domain size, that is, the number of people in the world. For the competing workshops experiment, we also varied the number of workshops proportionally with the number of people. The figure includes two results for WFMOC, one which only measures circuit evaluation time (inf) and another that includes both compilation and evaluation time (comp+inf). 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 similar to 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.

The *friends, smokers and drinkers* benchmark extends the standard friends and smokers benchmark with an extra MLN formula:

$$1.4 \quad \text{drinks}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{drinks}(Y)$$

WFOMC can lift this theory whereas C-FOVE cannot lift it. The curve for C-FOVE is not shown because the implementation fails.[5] 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. This answers question **(Q1)**.

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). The evaluation of these circuits scales well with the domain size, outperforming C-FOVE on all benchmarks. We can conclude that WFOMC as a whole scales well with the domain size, answering question **(Q2)**.

5.7.2 Influence of Grounding

Methodology

To investigate the influence of the grounding compilation rule (Section 4.5), we performed experiments with the *symmetric friends and smokers* theory (Van den Broeck, 2011b), which is a version of the *friends and smokers* model extended with a hard formula that makes the friends-relation symmetric:

$$2.0 \quad \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y)$$

$$\text{friends}(X, Y) \Rightarrow \text{friends}(Y, X).$$

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

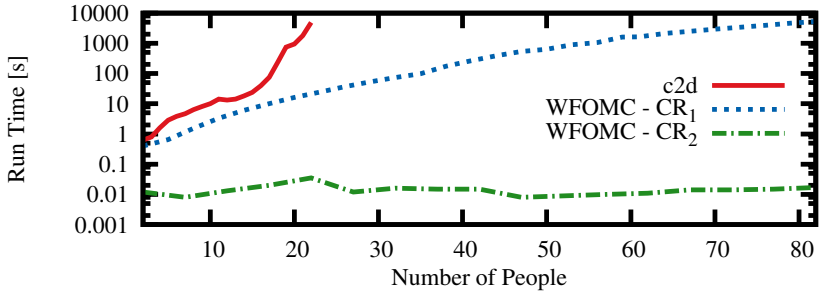


Figure 5.5: Symmetric friends and smokers experiment, comparing propositional (c2d) to first-order (WFOMC) knowledge compilation, for different sets of compilation rules.

We evaluate the performance of querying $\Pr(\text{smokes}(\textit{alice}))$ with increasing domain size, comparing the following algorithms for knowledge compilation:

c2d First ground the theory and then compile it to `d-DNNF` with the c2d compiler[6] (Darwiche, 2004).

WFOMC-CR$_1$ The Compile algorithm of Chapter 4, without the independent paired grounding compilation rule of Section 4.4.4.

WFOMC-CR$_2$ The Compile algorithm of Chapter 4 with all compilation rules.

The effect of removing the independent paired grounding rule from WFOMC-CR$_1$ is that the compilation algorithm will at some point fail to apply any lifted inference rule. It then resorts to using the grounding compilation rule. This corresponds to a partially lifted, partially ground inference algorithm. We did not compare to C-FOVE because it cannot perform lifted inference on this model.

Results

Figure 5.5 shows the experimental results. Propositional inference quickly becomes intractable when there are more than 20 people. The lifted inference algorithms scale much better. This answers question **(Q3)**.

[6] `http://reasoning.cs.ucla.edu/c2d/`

The CR_1 rules can exploit some symmetries in the model. For example, they eliminate all the smokes-atoms from the theory. They do, however, resort to grounding at a later stage of the compilation process. With all compilation rules active in CR_2, there is no need for grounding. This advantage is clear in the experiments, having an almost constant inference time in this range of domain sizes. We answer question **(Q4)** by observing that there can still be an advantage in performing first-order knowledge compilation, even if grounding is needed at some stage, as can be seen from comparing run time of c2d and WFOMC-CR_1.

Note that the run times for c2d and WFOMC-CR_1 include compilation and evaluation of the circuit, whereas the WFOMC-CR_2 run times only represent evaluation of a single compiled `FO-d-DNNF`. Compilation takes a constant two seconds for WFOMC-CR_2.

5.7.3 Conditional Probabilities

We will now investigate the task of computing conditional probabilities. More specifically, we compare the utility of conditioning a `FO-sda-DNNF` on evidence (cf. Equation 5.13) versus compiling a circuit that includes all evidence (cf. Equation 5.9).

Methodology

We compare the performance of three systems:

Conditioning Compiles `FO-sda-DNNF` circuits that support conditioning on unary literals, using Proposition 5.6.

Naive Compiles a separate `FO-sda-DNNF` circuit for each query and evidence set.

Ground Compiles a separate `d-DNNF` circuit for each query and evidence set using the c2d compiler.

Even though `d-DNNF` circuits support conditioning, we do not compile a single `d-DNNF` to reuse it for each query. The reason is that the unconditioned, grounded models are too large to compile, so the evidence must be used to simplify them, after which they can be compiled.

We evaluate these algorithms on the *competing workshops* and *friends and smokers* benchmarks. In all experiments, the goal is to compute the partition function (i.e., the denominator of Equation 5.8).

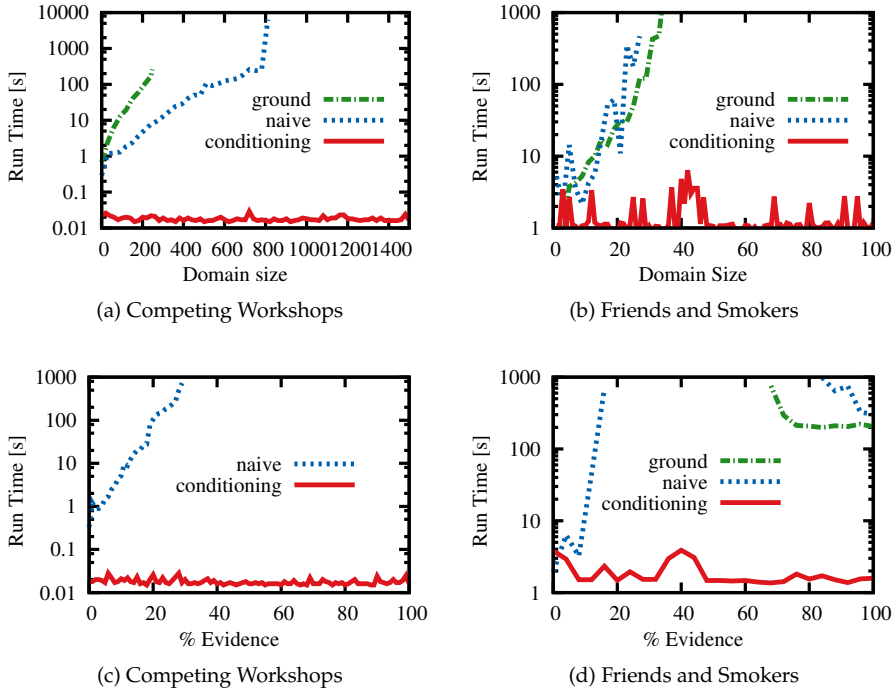


Figure 5.6: Plots (a) and (b) show run time with varying domain size and fixed percentage of evidence (50%). Plots (c) and (d) show run time with varying percentage of evidence and fixed domain size ((c) 1000 people, (d) 50 people).

Results

The experimental results are shown in Figure 5.6. The **conditioning** curve shows the sum of the transformation, the compilation, and the circuit evaluation (i.e., we are not amortizing the fixed costs over all answered queries).

Figures 5.6a and 5.6b illustrate the results for the first experiment where we vary the number of objects in the domain while holding the proportion of observed unary literals at 50%. Both the naive and ground approaches are

always slower than conditioning, whose curve stays relatively flat. When the domain size reaches 20 objects, conditioning is between one and two orders of magnitude faster than the baseline algorithms. It quickly becomes impossible for the baseline algorithms to compile the theories, whereas the conditioning approach has no such difficulty.

Figures 5.6c and 5.6d illustrate the results for the second experiment. Here, we hold the domain size constant (1000 and 50 people) and vary the proportion of observed unary literals. Conditioning works for any amount of evidence with relatively consistent run times. The naive approach works for low levels of evidence, which is the ideal case of lifted inference, as the lack of evidence preserves the symmetries encoded by the model (i.e., most objects are indistinguishable). Additionally, the compilation does not have the overhead of supporting conditioning. However, adding evidence quickly breaks the symmetries in the model, causing the theory to become mostly ground after shattering and the run time to dramatically increase, eventually making it impossible to compile the theory. Once evidence reaches 10%, conditioning is one to three orders of magnitude faster than the naive approach. Ground compilation fails entirely for the competing workshops domain. In the friends and smokers domain, both the naive and ground strategies work for high evidence levels because the evidence sufficiently simplifies the theory. Still, both are at least two orders of magnitude slower than conditioning.

Both experiments clearly demonstrate that the benefit of supporting conditioning on unary relations far outweighs the cost of applying the transformation from Proposition 5.6 and compiling a more complex circuit. This answers question **(Q5)**.

5.8 Conclusions and Future Work

This chapter started by giving an overview of statistical relational languages. Insights from lifted algorithms for other tasks, and the observation that performing inference in statistical relational models is hard, motivated the need for lifted probabilistic inference algorithms. We proposed a definition of lifted inference, called domain-lifted inference, and a new approach to exact lifted probabilistic inference. The proposed approach reduces probabilistic inference problems to weighted first-order model counting problems. It decouples

the logical, relational part of statistical relational models from their specific probabilistic parametrization.

We showed how to solve the weighted first-order model counting problem efficiently by first-order knowledge compilation to `FO-sda-DNNF` circuits. Furthermore, we presented an algorithm to efficiently condition on propositional and unary evidence, by using domain variables in first-order knowledge compilation. Our experiments showed that this approach provides benefits over propositional inference algorithms and advances the state of the art in lifted probabilistic inference.

For future work, we want to investigate new notions of lifted inference, other than domain-lifted inference. More specifically, we are still lacking a formal definition of what approximate lifted inference means. Alternative notions of exact lifted inference ideally can also distinguish between partially lifted algorithms, that at some point have the ground the model, and classical non-lifted inference algorithms. It should also be possible to define notions of lifted inference for cases where non-lifted algorithms are already polynomial in the domain size.

The work needed to increase the class of models for which we can perform lifted inference is closely tied with extensions of our first-order knowledge compilation algorithm presented in Chapter 4. For example, adding support for existential quantifiers in compilation would allow us to perform lifted inference for a larger class of probabilistic logic programs. A more challenging problem for future work is to develop lifted inference algorithms for probabilistic logic programs that are not tight, and cannot be represented in first-order logic, or for open-universe probabilistic models. These problems are still completely open.

6

Completeness and Liftability

This chapter introduces a general framework for defining classes of statistical relational models and associated classes of inference problems. Within this framework, we investigate the complexity of inference in terms of the size of logical variable domains and the size of the query and evidence, corresponding to different notions of lifted inference. We propose the notion of *completeness* of a lifted inference algorithm for a class of inference tasks, indicating that it can solve all such tasks in a lifted manner. The related notion of *liftability* of a class of inference tasks is concerned with the existence of a complete algorithm for this class.

Experimental and theoretical analyses of existing lifted inference techniques show that they provide (domain-)lifted inference in some cases where basic propositional inference techniques would exhibit exponential complexity. However, until recently, these positive results were mostly limited to examples of individual models, and little was known about the feasibility of lifted inference for certain well-defined classes of models.

The main contribution of this chapter is the first positive result that shows the feasibility of lifted inference for a non-trivial class of models and inference tasks. This result states that the problem of computing single marginal probabilities in the *two-variable* fragment of *universally quantified* FOL-DC is amenable to lifted

inference. More specifically, the first-order knowledge compilation approach to lifted inference is complete for this class of inference problems. This class corresponds to the class of quantifier-free Markov logic networks with up to two logical variables per formula.

In addition, we have several other contributions. We prove that the problem of computing *conditional probabilities* is not liftable by any inference algorithm, unless the evidence consists solely of *unary atoms* and *propositions*. We also show that inference in *monadic* FOL-DC or Markov logic is liftable. By combining these results with existing complexity results, we present an initial liftability map for inference in statistical relational models.

We introduce our formal framework for analyzing the complexity of inference in statistical relational models in **Section 6.1**. This includes the definition of classes of models and inference tasks and definitions of lifted inference. **Section 6.2** introduces the notion of completeness and proves several related results. The notion of liftability is introduced in **Section 6.3**, which indicates which classes of inference tasks are amenable to lifted inference. We show several positive and negative liftability results and give an overview of related work on lower complexity bounds for inference in statistical relational models. Finally, we give some related work and challenges for future work in **Section 6.4**.

This chapter is based on the following publications:

> G. Van den Broeck (2011b). "On the completeness of first-order knowledge compilation for lifted probabilistic inference". In: *Advances in Neural Information Processing Systems 24 (NIPS)*, pp. 1386–1394
>
> M. Jaeger and G. Van den Broeck (2012). "Liftability of probabilistic inference: Upper and lower bounds". In: *Proceedings of the 2nd International Workshop on Statistical Relational AI,*

6.1 Liftability Framework

This section introduces the classes of models, classes of inference problems and definitions of lifted inference used in this chapter.

6.1.1 Classes of Inference Tasks

A *probabilistic inference problem* $\mathrm{PI}(\Delta, w_T, w_F, q, e)$ for a weighted first-order theory is given by FOL-DC sentence Δ, two weight functions w_T,w_F, and two first-order sentences q, e. The solution to the inference problem is the conditional probability

$$\Pr(q|e) = \frac{\Pr(q \wedge e)}{\Pr(e)} = \frac{\mathrm{WFOMC}(\Delta \wedge q \wedge e, w_T, w_F)}{\mathrm{WFOMC}(\Delta \wedge e, w_T, w_F)}. \tag{5.8′}$$

A *class* of inference problems is defined by allowing arguments Δ, ϕ, and ψ only from some restricted classes $\mathcal{S}$ (the class of sentences), $\mathcal{Q}$ (the query class), and $\mathcal{E}$ (the evidence class), respectively. We use the notation

$$\mathrm{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E}) = \bigcup_{w_T, w_F} \{\mathrm{PI}(\Delta, w_T, w_F, q, e) \mid \Delta \in \mathcal{S}, q \in \mathcal{Q}, e \in \mathcal{E}\} \tag{6.1}$$

for classes of inference problems. Note that our definition does not depend on the model parameters expressed by the functions w_T, w_F.

For queries $\mathcal{Q}$ and evidence $\mathcal{E}$, we will use the following classes

- $\mathcal{A}$ for single ground atoms,
- $\mathcal{T}$ for terms (conjunctions) of ground literals,
- $\mathcal{T}_{0,1}$ for terms of ground literals with arity 0 or 1,
- $\emptyset$ for empty sets of evidence.

For example, $\mathcal{Q} = \mathcal{A}$ and $\mathcal{E} = \emptyset$ denotes queries for single marginals, without evidence.

Classes $\mathcal{S}$ are defined by various syntactic restrictions on the sentences in the model. All considered classes are sentences in function-free first-order logic with domain constraints (FOL-DC, see Section 3.1), with constraint sets as defined in Section 4.1.1.

We will additionally consider two fragments of FOL-DC:

- MonL, for *monadic* FOL-DC, that is, FOL-DC with no predicates of arity higher than one,[1] and
- RFOL, for *relational* FOL, that is, FOL-DC with a single unique domain term (set of constants) and no constants appearing in the logical atoms.

The RFOL class corresponds to formulas in classical first-order logic with a single finite domain of discourse and no constants.

We further distinguish between classes based on their support for *quantifiers* and the *equality predicate*. Sentences with arbitrary quantifiers are denoted by $(\forall\exists)$. Otherwise, sentences are assumed to be in Skolem normal form (Definition 4.1). Support for the equality predicate is indicated with $(=)$. For example, FOL-DC$^{(\forall\exists,=)}$ is FOL-DC with quantifiers and equality and FOL-DC$^{(=)}$ is FOL-DC in Skolem normal form with equality. A last type of subclass limits the number of bound logical variables per formula. For example, k-FOL-DC is FOL-DC with k bound logical variables per formula.

An algorithm *solves* a class $\text{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E})$, if it computes $\Pr(q|e)$ for all instances $\text{PI}(\Delta, \text{w}_\text{T}, \text{w}_\text{F}, q, e)$ in the class.

6.1.2 Definitions of Lifted Probabilistic Inference

Definition 5.2 in Section 5.3.4 proposed the following definition of lifted inference, in terms of the computational complexity of inference w.r.t. the domains of logical variables.

Definition (Domain-Lifted Inference)**.** A probabilistic inference procedure is *domain-lifted* for a model Δ, query q and evidence e iff it computes $\Pr(q|e)$ in time *polynomial* in each $|D_1|, \ldots, |D_k|$, where D_i is the domain of the logical variable X_i appearing in Δ, q or e.

Domain-lifted inference does not prohibit the algorithm to have an *exponential* complexity in the number of formulas, predicates and logical terms in Δ, ϕ and ψ. The definition was motivated by the observation that first-order theories are often concise, but the presence of large domains causes inference to become intractable, when done at the propositional level.

[1] Monadic first-order logic only permits predicates with arity exactly one. We will here also permit propositions in the sentence.

In this chapter we will also consider an alternative definition, which takes into account the query and evidence.

Definition 6.1 (DQE-Lifted Inference)**.** A probabilistic inference procedure is *domain-, query- and evidence-lifted* (DQE-lifted) for a model Δ, query q and evidence e iff it is domain-lifted and it computes $\Pr(q|e)$ in time polynomial in the size of q and e.

This definition is motivated by the fact that domain-lifted inference algorithms tend to perform poorly in the presence of evidence, which breaks the symmetries in the first-order model (cf., Section 5.7.3). When sufficient amounts of evidence are present, lifted inference algorithms will start to behave identically to their propositional counterparts, that is, first-order variable elimination (Poole, 2003) will perform variable elimination (Zhang and Poole, 1994), first-order knowledge compilation to `FO-sda-DNNF` will perform propositional knowledge compilation to `d-DNNF`, etc.

6.2 Completeness

This section proves completeness results for lifted inference by weighted first-order model counting and first-order knowledge compilation (WFOMC).

Definition 6.2 (Completeness)**.** An algorithm is called a *complete* domain-lifted (DQE-lifted) inference algorithm for the class $\mathrm{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E})$ iff it is domain-lifted (DQE-lifted) and solves this class.

Theorem 5.5 already established that WFOMC is a domain-lifted inference algorithm. Hence, it is a complete domain-lifted algorithm for the class of problems it can solve. Unfortunately, this class cannot easily be captured in a single syntactic definition. We will now show several syntactically defined classes that WFOMC can solve, leading to our completeness results.

6.2.1 Completeness for Monadic Logic

To prove that WFOMC can solve the class `MonL`, we first prove the following lemmas, related to the first-order knowledge compilation algorithm into `FO-da-DNNF` (Algorithm 1, p. 84), as presented in Chapter 4.

Lemma 6.1. *Algorithm 1 removes all atoms without bound logical variable arguments (including propositions) from a* FO-CNF.

Proof. Since all atoms in the sentence contain no bound variables, the ShannonDecomposition rule (Section 4.2.3) can be applied as long as the sentence contains atoms. A subsequent application of the UnitPropagate rule (Section 4.2.1) removes one atom from the sentence. This continues until all atoms are removed. □

Lemma 6.2. *Algorithm 1 can compile any* FO-CNF *with one bound variable per atom into* FO-da-DNNF.

Proof. We can assume that all atoms without bound variables have been removed from the theory (Lemma 6.1). What is left are all atoms with one bound variable argument. As long as there are atoms left in the theory, the AtomCounting rule (Section 4.4.6) applies. Its preconditions can always be met by applying the ShatteredCompilation rule (Section 4.3.4). A subsequent application of the UnitPropagate rule (Section 4.2.1) removes a non-empty set of atoms from the theory. When all atoms are removed, the base case of Algorithm 1 is met. □

This shows that we can always compile the circuits for $\Delta \wedge q \wedge e$ and $\Delta \wedge e$ in Equation 5.8′, given that these are all monadic.[2] Therefore, we have the following result.

Theorem 6.3. *WFOMC is a complete domain-lifted inference algorithm for the class* $\mathrm{PI}(\mathtt{MonL}(=), \mathtt{MonL}(=), \mathtt{MonL}(=))$.

Let us now look at completeness for the definition of DQE-lifted inference. This requires that inference is also polynomial in the size of the query and evidence.

Proposition 6.4. *WFOMC is* not *a complete DQE-lifted inference algorithm for the class* $\mathrm{PI}(\mathtt{MonL}(=), \mathtt{MonL}(=), \mathtt{MonL}(=))$.

Proof. Converting an arbitrary query into FO-CNF, which is the input of the compilation algorithm, can cause an exponential increase in the size of the

[2] Note that the input to Algorithm 1 is a FO-CNF in Skolem normal form, where the FOL part of each formula is a clause. We can convert the FOL part of an arbitrary formula in Skolem normal form into CNF and thereby convert any theory in Skolem normal form into a FO-CNF.

theory, making inference exponential in the size of the query. This follows from the succinctness relation between CNF and DNF (Gogic et al., 1995). □

Theorem 6.5. *WFOMC is a complete DQE-lifted inference algorithm for the class* $\text{PI}(\texttt{MonL}_{(=)}, \mathcal{T}, \mathcal{T})$.

Proof. Section 5.5.3 outlines an approach to compile FO-sda-DNNF circuits that can be conditioned on any proposition or unary literal. It involves a transformation of the theory Δ (Proposition 5.6). The set of monadic theories MonL is closed under this transformation. In case the query or evidence term contains higher-arity literals, these are independent of the theory Δ, and can simply be conjoined with its circuit. Hence, it follows from Lemma 6.2 that the circuits built by this technique can be compiled. Subsequently conditioning the circuit on the propositional and unary evidence and evaluating its weighted model counts is polynomial in the node labels and therefore polynomial in the domain size, query size and evidence size. □

6.2.2 Completeness for the Two-Variable Fragment

Monadic logic has a very limited expressivity. It can only represent knowledge about properties of objects, not about relations between objects. In this section, we analyze the completeness of WFOMC for a more powerful class of models, namely the two-variable fragment of FOL-DC.

We first prove that first-order knowledge compilation can compile any theory with up to two bound logical variables per clause.

Theorem 6.6. *Algorithm 1 can compile any* FO-CNF *with up to two bound logical variables per formula into* FO-da-DNNF.

Proof. Algorithm 1 compiles these formulas by using three sets of compilation rules in sequence:

(i) All atoms with zero or one logical variables are removed from the theory by the ShannonDecomposition, AtomCounting and UnitPropagation rules (cf., proof of Lemma 6.2). In a theory with up to two bound logical variables per formula, this leaves us only with atoms that contain both variables. Hence, every variable is root and every class of unifying variables is a root unifying class (Definition 4.3).

(ii) Lemma 4.7 states that when a theory is first shattered and independent subtheories are decomposed, any root unifying class contains the same number of variables from each atom in the theory. By applying the IndependenceRule (Section 4.2.2) and ShatteredCompilationRule rules, we can always satisfy the conditions stated in Lemma 4.7.

(iii) Step (i) guarantees that the theory is either empty or contains a root unifying class. Step (ii) guarantees that this root unifying class either contains

- one logical variable per atom in the theory, which means the IndependentSingleGroundingsRule (Section 4.4.3) rule applies, or
- two logical variables per atom in the theory, which means the IndependentPairedGroundingsRule (Section 4.4.4) rule applies.

These rules recursively call Algorithm 1 to compile a theory with at most one bound logical variable per atom, which Lemma 6.2 states can always be compiled.

□

We can now use this result to establish two completeness results for the two-variable fragment. According to Equation 5.8′, we need to compile the formulas $\Delta \wedge q \wedge e$ and $\Delta \wedge e$ into $\texttt{FO-da-DNNF}$. For the class $\mathrm{PI}(\texttt{2-FOL-DC}(=), \texttt{2-FOL-DC}(=), \texttt{2-FOL-DC}(=))$, these formulas can be converted into $\texttt{FO-CNF}$s that have up to two bound logical variables per formula, which can be compiled (Theorem 6.6). Because compilation has a complexity that is independent of the domain sizes and weighted first-order model counting on these circuits is polynomial in the domain sizes, we have the following main result.

Theorem 6.7. *WFOMC is a complete domain-lifted inference algorithm for the class* $\mathrm{PI}(\texttt{2-FOL-DC}(=), \texttt{2-FOL-DC}(=), \texttt{2-FOL-DC}(=))$.

Again, by applying the conditioning technique outlined in Section 5.5.3, we obtain a corresponding result for the notion of DQE-lifted inference.

Theorem 6.8. *WFOMC is a complete DQE-lifted inference algorithm for the class* $\mathrm{PI}(\texttt{2-FOL-DC}(=), \mathcal{T}_{0,1}, \mathcal{T}_{0,1})$.

Proof. Analogous to the proof of Theorem 6.5, only using Theorem 6.6 instead of Lemma 6.2. □

This means that lifted inference is possible when we have (partial) type information about every object in the world. When we have evidence on properties (which have arity one), this induces a type for each object, which is the set of properties of that object. For example, when there are k properties, each object can belong to one of 2^k types, and we can have partial evidence about the type of each object, which induces 3^k equivalence classes. Theorem 6.8 shows that probabilistic reasoning with such information is polynomial in the number of objects in the world.

6.2.3 Completeness for Markov Logic Networks, Parfactor Graphs and ProbLog Programs

To put these completeness results into perspective, we will now investigate their implications on popular statistical relational modeling languages. Section 5.4.2 outlines the reduction from *Markov logic networks*, *parfactor graphs* and *ProbLog programs* to WFOMC problems. Next, we discuss the syntactic properties of the generated WFOMC problems and how our completeness results carry over.

All completeness results pertain to sentences in *Skolem normal form*. The reduction from MLNs is in this form when the MLN is quantifier-free. When the MLN contains a universal or existential quantifier, the reduction will contain an existential quantifier and will not be in Skolem normal form.[3] The reduction from parfactor graphs is in Skolem normal form when it contains no counting formulas. For ProbLog programs, the conditions are more strict. To obtain a Skolem normal form theory, all the logical variables that appear in the body of a Prolog clause must also appear in its head.

Under these conditions, our completeness results for *monadic* WFOMC models (Theorems 6.3 and 6.5) also apply to monadic MLNs, monadic parfactor graphs and ProbLog programs with monadic bodies. This is not immediate, because these models can reduce to WFOMC problems that are not monadic. However, the reduction has at most one non-monadic atom per clause, and we can extend Lemma 6.2 to show that Algorithm 1 can compile any such `FO-CNF`.

Under the above restrictions to obtain Skolem normal form reductions, the completeness results for the *two-variable fragment* of WFOMC do carry over directly. WFOMC is a complete domain-lifted inference algorithm for MLNs

[3] One can always replace an existential quantifier in the MLN by a finite disjunction, at the cost of introducing additional explicit constants in the model and increasing its size.

with up to two logical variables per formula, parfactor graphs with up to two variables per parfactor, and ProbLog programs with up to two variables per clause. For MLNs, this covers many of the models in use today. As we will see in Chapter 8, many MLNs generated by structure learning algorithms are in this class. In contrast, the condition for ProbLog that all variables that appear in the body must also appear in the head is very restrictive.

The two-variable fragment is not trivial and surprisingly expressive, containing many models in practical use today. It can contain important concepts such as symmetric, anti-symmetric, total and homophily relations (in MLN syntax):

$$\text{friends}(X, Y) \Rightarrow \text{friends}(Y, X).$$

$$\text{parent}(X, Y) \Rightarrow \neg\, \text{parent}(Y, X).$$

$$\neg \leq (X, Y) \Rightarrow \leq (Y, X).$$

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

Furthermore, all models currently used in the lifted inference literature are in this class. Still, there are useful models that are not in it, most notably models with existential quantifiers and concepts such as transitivity and generalized homophily:

$$\text{friends}(X, Y) \wedge \text{friends}(Y, Z) \Rightarrow \text{friends}(X, Z). \tag{6.2}$$

$$\text{likes}(X, Z) \wedge \text{friends}(X, Y) \Rightarrow \text{likes}(Y, Z). \tag{6.3}$$

which contain three logical variables per formula. For these formulas, compilation fails because after removing all atoms with less than two logical variables (step (i) of Theorem 6.6), not all logical variables are guaranteed to be root, the groundings of the model are still connected and do not decompose. Many models not in the two-variable fragment can still be compiled to `FO-da-DNNF`, but there are no guarantees in the form of a completeness result.

6.3 Liftability

The related notion of liftability talks about the existence of a complete domain-lifted algorithm.

Definition 6.3 (Liftability)**.** A class $\mathrm{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E})$ is domain-liftable (DQE-liftable) iff there exists an algorithm that is complete domain-lifted (DQE-lifted) for this class.

The above definition of liftability pertains only to exact inference. An analogous notion is *approximate liftability*, when there exists an algorithm that ϵ-approximately solves all problems in a class and is domain-lifted. An algorithm ϵ-approximately solves $\mathrm{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E})$, if for any $\mathrm{PI}(\Delta, \mathrm{w_T}, \mathrm{w_F}, q, e)$ in the class it returns a number in $[\Pr(q \mid e) - \epsilon, \Pr(q \mid e) + \epsilon]$.[4]

The following sections will investigate the liftability of different classes of $\mathrm{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E})$. Table 6.1 summarizes the results and indicates several unknown liftability results.

Open Problem 5. *Question marks in Table 6.1.*

Notion	$\mathcal{S}$	$\mathcal{Q}$	$\mathcal{E}$	Exact	Approx.	Theorem
domain	RFOL$(\forall\exists,=)$	$\mathcal{A}$	$\emptyset$	✗	✗	6.14
	RFOL	$\mathcal{A}$	$\emptyset$	✗*	✗*	6.15
	k-RFOL	$\mathcal{A}$	$\emptyset$	?	?	
	3-RFOL	$\mathcal{A}$	$\emptyset$	?	?	
	2-FOL-DC$(=)$	2-FOL-DC$(=)$	2-FOL-DC$(=)$	✓	✓	6.11
	MonL$(=)$	MonL$(=)$	MonL$(=)$	✓	✓	6.9
dqe	2-RFOL	$\mathcal{A}$	$\mathcal{T}$	✗	?	6.17
	2-FOL-DC$(=)$	$\mathcal{T}_{0,1}$	$\mathcal{T}_{0,1}$	✓	✓	6.12
	MonL$(=)$	$\mathcal{T}_{0,1}$	$\mathcal{T}_{0,1}$	✓	✓	6.10

Table 6.1: Liftability results (contingent on complexity assumptions) w.r.t. classes of knowledge bases $\mathcal{S}$, queries $\mathcal{Q}$ and evidence $\mathcal{E}$. For negative/positive results, the most specific/general class is shown. ✗*: subject to an additional condition on polynomial time complexity as a function of parameter complexity.

6.3.1 Positive Liftability Results

A proof that one particular algorithm for lifted probabilistic inference is complete for a class of problems is also a constructive proof that this class

[4] Implicitly, Pr denotes the distribution w.r.t. the given Δ, $\mathrm{w_T}$ and $\mathrm{w_F}$.

of models is liftable. Hence, from Theorems 6.3, 6.5, 6.7 and 6.8, we obtain the following liftability results.

Corollary 6.9. *The class* $\mathrm{PI}(\mathtt{MonL}(=), \mathtt{MonL}(=), \mathtt{MonL}(=))$ *is domain-liftable.*

Corollary 6.10. *The class* $\mathrm{PI}(\mathtt{MonL}(=), \mathcal{T}_{0,1}, \mathcal{T}_{0,1})$ *is DQE-liftable.*

Corollary 6.11. *The class* $\mathrm{PI}(\mathtt{2\text{-}FOL\text{-}DC}(=), \mathtt{2\text{-}FOL\text{-}DC}(=), \mathtt{2\text{-}FOL\text{-}DC}(=))$ *is domain-liftable.*

Corollary 6.12. *The class* $\mathrm{PI}(\mathtt{2\text{-}FOL\text{-}DC}(=), \mathcal{T}_{0,1}, \mathcal{T}_{0,1})$ *is DQE-liftable.*

Because these classes are exactly liftable, they are also trivially approximately liftable.

6.3.2 Negative Domain-Liftability Results

Before the problem of lifted probabilistic inference was posed, Jaeger (2000) showed that under certain assumptions on the expressivity of a statistical relational modeling language, probabilistic inference is not polynomial in the domain size, thereby demonstrating some inherent limitations in terms of worst-case complexity for the goals of lifted inference. The results of Jaeger (2000) essentially assume a directed modeling framework, and the expressivity requirements amount to a probabilistic version of full first-order logic.

Because FOL-DC and many popular statistical relational languages are undirected and, more significantly, because we are interested in the liftability of classes of models without the expressivity of full first-order logic, Jaeger and Van den Broeck (2012); Jaeger (2012) investigate how these earlier intractability results are applicable to ongoing efforts. Extending the general approach taken in Jaeger (2000), this work derives new lower complexity bounds that show that domain-lifted inference still is infeasible when the strong assumptions of Jaeger (2000) are loosened, and only the expressivity of Skolem normal form relational first-order logic without equality (RFOL) is required.

All negative liftability results for domain-lifted inference are obtained by reducing the *spectrum recognition problem* to probabilistic inference problems. We give a brief overview of Jaeger and Van den Broeck (2012); Jaeger (2012) and how these results fit into the liftability framework.

We first briefly review the fundamental concepts about spectra of first-order logic sentences, and the complexity class ETIME.

Definition 6.4. Let ψ be a sentence in first-order logic. The *spectrum* of ψ is the set of integers $n \in \mathbb{N}$ for which ψ is satisfiable by an interpretation of domain size n.

Example 6.1. Let $\psi = \psi_1 \wedge \psi_2 \wedge \psi_3$, where

$$\begin{aligned} \psi_1 &\equiv \forall X, Y : \mathrm{u}(X,Y) \Leftrightarrow \mathrm{u}(Y,X) \\ \psi_2 &\equiv \forall X \, \exists Y : Y \neq X \wedge \mathrm{u}(X,Y) \\ \psi_3 &\equiv \forall X, Y, Y' : (\mathrm{u}(X,Y) \wedge \mathrm{u}(X,Y') \Rightarrow Y = Y') \end{aligned}$$

ψ expresses that the binary relation u defines an undirected graph (ψ_1) in which every node is connected to exactly one other node (ψ_2, ψ_3). Thus, ψ describes a *pairing* relation that is satisfiable exactly over domains of even size: $spec(\psi) = \{n \mid n \text{ even}\}$.

The complexity class ETIME consists of problems solvable on a deterministic Turing machine in time $O(2^{cn})$, for some constant c (Johnson, 1990). The corresponding nondeterministic class is NETIME. Based on a characterization of NETIME in terms of first-order spectra given by Jones and Selman (1972), we obtain the following.

Proposition 6.13. *If NETIME $\neq$ ETIME, then there exists a first-order sentence ϕ, such that $\{n \mid n \in \mathrm{spec}(\phi)\}$ cannot be recognized by a deterministic algorithm in time polynomial in n.*

Thus, by reducing the spectra-recognition problem to a class of inference problems $\mathrm{PI}(\mathcal{S}, \mathcal{Q}, \mathcal{E})$, one establishes that the latter is not polynomial in the domain size (under the assumption ETIME $\neq$ NETIME).

We first state a result for knowledge bases using $\mathtt{RFOL}_{(\forall\exists,=)}$. This is rather straightforward, and (for exact inference) already implied by the results of Jaeger (2000).

Theorem 6.14. *If NETIME $\neq$ ETIME, then there does not exist an algorithm that 0.25-approximately solves $\mathrm{PI}(\mathtt{RFOL}_{(\forall\exists,=)}, \mathcal{A}, \emptyset)$ in time polynomial in the domain size.*

The following theorem extends Theorem 6.14 to knowledge bases with only quantifier-free formulas without equality. At the same time a slight weakening is introduced by imposing an additional condition on the complexity in terms of representation size of the weight parameters $\mathrm{w}_\mathrm{T}, \mathrm{w}_\mathrm{F}$.

Theorem 6.15. *If NETIME $\neq$ ETIME, then there does not exist an algorithm that 0.25-approximately solves* $\mathrm{PI}(\texttt{RFOL}, \mathcal{A}, \emptyset)$ *in time polynomial in n and the representation size $l := \sum_i \log(w_i)$ of the weight parameters in* $\mathrm{w_T}, \mathrm{w_F}$.

The basic strategy for proving this theorem is to replace quantifiers with relational encodings of Skolem functions, and the equality predicate $=$ with an ordinary binary relation $\mathrm{E}(\cdot,\cdot)$. The problem, then, is to enforce that the newly introduced relations behave like functions, respectively like the equality relation. Given the expressive power only of RFOL, this, naturally, is not possible to do exactly. However, by adding predicates and formulas and by using suitable weight functions, one can ensure that interpretations in which the Skolem relations and the E relation do not show the desired properties have a negligible weight. The weights for the formulas constraining the Skolem and E relations, now, have to be calibrated as a function of the domain size n. Thus, for a given first-order formula ψ, the problem $n \in \mathit{spec}(\psi)$ is reduced to an inference problem $\mathrm{PI}(\Delta(n), \mathrm{w_T}(n), \mathrm{w_F}(n), q, \mathrm{T})$, where for different n, we have different $\Delta, \mathrm{w_T}, \mathrm{w_F}$.

6.3.3 Negative DQE-Liftability Result

We will now prove a negative DQE-liftability result for the class of models 2-FOL-DC$(=)$, in spite of its positive domain-liftability result. Section 3.4.4 investigated the problem of conditioning a FO-da-DNNF circuit. Theorem 3.18 established that this is impossible in general, unless P=NP. With a similar proof strategy, we can prove our negative liftability result.

The proof strategy shows that one can solve #2SAT, that is, the model counting problem for propositional CNFs with 2 literals per clause (2CNF), by conditioning the following theory and computing its model count.

$$\begin{aligned}
&\forall X, Y \in \mathsf{Prop} : \neg\, \mathrm{q} \vee \quad \mathrm{p}(X) \vee \quad \mathrm{p}(Y) \vee \neg\, \mathrm{c}_1(X, Y) \\
&\forall X, Y \in \mathsf{Prop} : \neg\, \mathrm{q} \vee \quad \mathrm{p}(X) \vee \neg\, \mathrm{p}(Y) \vee \neg\, \mathrm{c}_2(X, Y) \\
&\forall X, Y \in \mathsf{Prop} : \neg\, \mathrm{q} \vee \neg\, \mathrm{p}(X) \vee \neg\, \mathrm{p}(Y) \vee \neg\, \mathrm{c}_3(X, Y) \qquad (6.4)
\end{aligned}$$

Assuming for now that we have conditioned on q being true, this formula is equivalent to Formula 3.11. Atoms $\mathrm{p}(X)$ represent propositions and atoms

$c_i(X, Y)$ represent clauses. Conditioning on a positive c_i literal includes the clause of type i for the propositions $p(X)$ and $p(Y)$ in the CNF (see Section 3.4.4).

Given that the #2SAT problem is #P-complete (Valiant, 1979), we will now show that computing conditional probabilities is #P-hard, by showing that #2SAT is reducible to it.

Theorem 6.16. *For any $\mathcal{S}$ that can express a uniform distribution over the models of Formula 6.4, solving $\mathrm{PI}(\mathcal{S}, \mathcal{A}, \mathcal{T})$ is #P-hard.*

Proof. Querying for $\Pr(q \mid \psi)$, where ψ assigns a truth value to every c_i atom, returns $c/(c + 2^n)$, where c is the model count of the 2CNF represented by ψ and 2^n the number of possible assignments to the n propositions in the 2CNF. By solving for c, we can answer arbitrary #2SAT problems. □

Because Formula 6.4 is in 2-RFOL, we have the following liftability result.

Corollary 6.17. *$\mathrm{PI}(\text{2-RFOL}, \mathcal{A}, \mathcal{T})$ is not (DQE-)liftable, unless P=NP.*

This result highlights an important limitation of all exact lifted inference methods: computing probabilities with evidence on binary relations cannot be polynomial in the size of the evidence, unless P=NP. This result identifies a sharp contrast between lifted and propositional inference algorithms: whereas evidence in undirected models simplifies inference in the propositional case, it complicates inference in the lifted case.

6.4 Related and Future Work

The IndependentPairedGroundings is an essential tool in establishing the completeness result for the two-variable fragment. The lack of an inference rule that could deal with theories with a root unifying class with two variables per atom, used to be the major limitation of early lifted inference approaches, such as first-order variable elimination. Recently, Taghipour et al. (2012b) have shown that first-order variable elimination with an extension similar to the IndependentPairedGroundings rule is complete domain-lifted.

Domingos and Webb (2012) introduce "Tractable Markov Logic (TML)", which also is a syntactically restricted statistical relational modeling language. TML is not directly defined as a fragment of first-order logic fitting our weighted model

counting framework, and therefore its placement into our complexity map of Table 6.1 is not immediate, but still seems quite feasible based on a precise first-order representation of TML syntax. It is remarkable, though, that TML is related to probabilistic description logics, and that description logics, in turn, are well-known to be representable in the 2-variable fragment of first-order logic (Hustadt, Schmidt, and Georgieva, 2004). Thus, it may well turn out that TML and PI$(2\text{-}\mathtt{FOL}\text{-}\mathtt{DC}(=), \mathcal{T}, \mathcal{T})$ actually exploit the same underlying source of tractability, and that a combination of the two could lead to the identification of a tractable class that has both a natural and concise characterization, and that is more expressive than either of the two classes alone.

Further important open problems include a complexity results for the 3-variable fragment 3-RFOL. Formulas with three variables provide important added expressivity compared to the 2-variable fragment. The transitive clause (Formula 6.2) and generalized homophily (Formula 6.3) are in 3-RFOL. Liftability for at least some inference classes PI$(3\text{-}\mathtt{RFOL}, \mathbf{?}, \mathbf{?})$ would be an important extension. For transitivity, we know precisely which are the symmetries of the model (e.g., by preemptive shattering), but we do not know how to exploit them.

Another open problem is a complexity result for RFOL with bounded weight parameters. Our lower complexity bound of Theorem 6.15 is based on the assumptions that weights can be arbitrarily small or big. For practical modeling tasks it would not be a significant limitation to restrict weights to lie between a certain lower and upper bound (in addition to zero weights for hard constraints). While, on the one hand, our results do not imply non-liftability for such weight-bounded knowledge bases, it is hard to imagine how a concrete exact inference algorithm could exploit weight-boundedness, since such an algorithm would then have to exhibit non-polynomial complexity with respect to the representation size of the weights. However, in the case of approximate inference, it is feasible that disallowing extreme weights improves worst-case approximation quality and therefore also approximate liftability. For different tasks and parameter spaces, it is clear that broader classes of problems are liftable. For example, MPE inference with $0/1$ weights is identical to the SAT problem, for which first-order resolution is complete.

6.5 Conclusions

In this chapter, we have collected existing and new results that are the beginning of a systematic formal analysis of lifted inference. The general factorization of classes of inference problems according to knowledge base, query, and evidence-classes allows us to accommodate a variety of different complexity aspects in a coherent framework, with a particular focus on the question of liftability of inference. Furthermore, our framework aligns the complexity of inference analysis with well-established concepts of syntactic complexity of predicate logic formulas, notably in terms of quantifier complexity and the number of variables.

Within this framework, we obtained several concrete results. We identified of a first non-trivial classes of models and inference tasks that is amenable to lifted inference. On the other hand, we showed that for certain other classes, lifted inference is unlikely to be possible. We expect future work that extends these proofs to larger classes, and to approximate inference, as well as work that introduces additional notions of lifting.

7

From Approximate to Exact Lifted Inference

This chapter proposes an approach to *approximate* lifted inference that is based on performing *exact* lifted inference in a simplified first-order model. Namely, we simplify the structure of a first-order model until it is amenable to exact lifted inference, by relaxing first-order equivalence constraints in the model. Relaxing equivalence constraints ignores (many) dependencies between random variables, so we compensate for this relaxation by restoring a weaker notion of equivalence, in a lifted way. We then incrementally improve this approximation by recovering first-order equivalence constraints back into the model.

In fact, our proposal corresponds to an approach to approximate inference, called Relax, Compensate and then Recover (RCR) for (ground) probabilistic graphical models (see Section 2.5.2).[1] For such models, the RCR framework gives rise to a spectrum of approximations, with iterative belief propagation on one end (when we use the coarsest possible model), and exact inference on the other (when we use the original model). We show how relaxations, compensations and recovery can all be performed at the first-order level, giving rise to a *spectrum of first-order approximations*, with lifted first-order belief propagation on one end,

[1] A solver based on the RCR framework won first place in two categories evaluated at the UAI'10 approximate inference challenge, in the most demanding sub-category of 20-second response time (Elidan and Globerson, 2010).

and exact lifted inference on the other.

The propositional RCR framework operates on probabilistic graphical models. In order to motivate our Lifted RCR algorithm, we first adapt RCR to work with propositional, or ground Markov logic networks in **Section 7.1**. A Lifted RCR algorithm that operates on first-order MLNs is presented in **Section 7.2** An essential component in the Lifted RCR framework is an algorithm that partitions equiprobable equivalences. **Section 7.3** discusses this problem in more detail. **Section 7.4** discusses related work on approximate lifted inference. It identifies propositional inference algorithms whose approximations are in the spectrum of Lifted RCR, as well as existing lifted inference algorithms that fit into this framework. In **Section 7.5**, we evaluate our approach on benchmarks from the lifted inference literature. Experiments indicate that recovering a small number of first-order equivalences can improve on the approximations of lifted belief propagation by several orders of magnitude. We show that, compared to Ground RCR, Lifted RCR can recover many more equivalences in the same amount of time, leading to better approximations.

The work presented in this chapter was previously published as:

> G. Van den Broeck, A. Choi, and A. Darwiche (2012). "Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference". In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*

7.1 RCR for Ground MLNs

Although our main objective is to present a lifted version of the RCR framework, we start by adapting RCR to ground Markov logic networks (MLNs). This is meant to both motivate the specifics of Lifted RCR and to provide a basis for its semantics (i.e., the correctness of Lifted RCR will be against the results of Ground RCR).

The syntax and semantics of MLNs were introduced in Section 5.1.1. In addition, we will here assume that all logical variables in the MLN are free. This can always be achieved by transforming existential quantifiers into disjunctions and universal quantifiers into conjunctions. As a consequence of this assumption, all instances of MLN formulas are ground. The grounding of an entire MLN Δ can simply be obtained by replacing each formula in Δ with all its instances (using

the same weight). The *ground distribution* of MLN Δ is then the probability distribution induced by the grounding of Δ.

Example 7.1. As a running example, we will use the MLN from Example 5.1.

$$1.5 \quad \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y) \tag{5.2'}$$

$$1.3 \quad \text{smokes}(X) \Rightarrow \text{cancer}(X) \tag{5.3'}$$

For the domain $\{alice, bob\}$, this first-order MLN represents the ground MLN

$$1.5 \quad \text{smokes}(alice) \wedge \text{friends}(alice, alice) \Rightarrow \text{smokes}(alice)$$

$$1.5 \quad \text{smokes}(alice) \wedge \text{friends}(alice, bob) \Rightarrow \text{smokes}(bob)$$

$$1.5 \quad \text{smokes}(bob) \wedge \text{friends}(bob, alice) \Rightarrow \text{smokes}(alice)$$

$$1.5 \quad \text{smokes}(bob) \wedge \text{friends}(bob, bob) \Rightarrow \text{smokes}(bob)$$

$$1.3 \quad \text{smokes}(alice) \Rightarrow \text{cancer}(alice) \tag{7.1}$$

$$1.3 \quad \text{smokes}(bob) \Rightarrow \text{cancer}(bob)$$

The RCR algorithm can be understood in terms of three steps: Relaxation (R), Compensation (C), and Recovery (R). Next, we examine each of these steps and how they apply to ground MLNs.

7.1.1 Ground Relaxation

Relaxation is the process of ignoring interactions between the formulas of a ground MLN. An interaction takes place when the same ground atom a_g appears in more than one ground formula in an MLN. We can ignore this interaction via a two step process. First, we rename one occurrence of a_g into, say, a'_g, through a process that we call *cloning*. We then assert an equivalence constraint between the original ground atom a_g and its clone, $a_g \Leftrightarrow a'_g$. At this point, we can ignore the interaction by simply dropping the equivalence constraint, through a process that we call *relaxation*. Bringing back the equivalence is known as *recovery* and will be discussed in more detail later.

Example 7.2. The smokes$(alice)$ atom in Formula 7.1 leads to an interaction between this formula and some of the other five formulas in the ground MLN. To

ignore this interaction, we first rename this atom occurrence into $\text{smokes}_2(alice)$ leading to the modified formula

$$1.3 \quad \text{smokes}_2(alice) \Rightarrow \text{cancer}(alice) \tag{7.2}$$

which replaces Formula 7.1 in the MLN. The corresponding equivalence is

$$\text{smokes}_2(alice) \Leftrightarrow \text{smokes}(alice) \tag{7.3}$$

Dropping this equivalence constraint amounts to removing the interaction between Formula 7.2 and the rest of the MLN.

7.1.2 Ground Compensation

When relaxing an equivalence constraint $a_g \Leftrightarrow a'_g$, the connection between the ground atoms a_g and a'_g is lost. One can *compensate* for this loss by adding two weighted atoms

$$w \quad a_g \qquad \text{and} \qquad w' \quad a'_g$$

If the weights w and w' are chosen carefully, one can reestablish a weaker connection between the ground atoms. For example, one can choose these weights to ensure that the ground atoms have the same probability, establishing a weaker notion of equivalence.

We will now suggest a specific compensation scheme based on a key result from Choi and Darwiche (2006). Suppose that we relax a single equivalence constraint, $a_g \Leftrightarrow a'_g$, which splits the MLN into two disconnected components, one containing atom a_g and another containing atom a'_g. Suppose further that we choose the compensations w and w' such that

$$\Pr(a_g) = \Pr(a'_g) = \frac{e^{w+w'}}{1 + e^{w+w'}}. \tag{7.4}$$

We now have a number of guarantees. First, the resulting MLN will yield exact results when computing the probability of any ground atom. Second, the compensations w and w' can be identified by finding a fixed point for the

following equations:

$$w_{i+1} = \log\left(\Pr_i(a'_g)\right) - \log\left(\Pr_i(\neg a'_g)\right) - w'_i$$
$$w'_{i+1} = \log\left(\Pr_i(a_g)\right) - \log\left(\Pr_i(\neg a_g)\right) - w_i. \quad (7.5)$$

Following Choi and Darwiche (2006), we will seek compensations using these update equations even when the relaxed equivalence constraint does not disconnect the MLN, and even when relaxing multiple equivalence constraints. In this more general case, a fixed-point to the above equations will still guarantee the weak equivalence given in Equation 7.4. However, when computing the probabilities of ground atoms, we will only get approximations instead of exact results.[2]

Searching for compensations using Equations 7.5 will lead to the generation of a sequence of MLNs that differ only on the weights of atoms added during the compensation process. The first MLN in this sequence is obtained by using zero weights for all compensating atoms, leading to an initial ground distribution $\Pr_0$. Each application of Equations 7.5 will then lead to a new MLN (with new compensations) and, hence, a new ground distribution, $\Pr_{i+1}$. Upon convergence, the resulting MLN and its ground distribution will then be used for answering queries. This is typically done using an exact inference algorithm as one usually relaxes enough equivalence constraints to make the ground MLN amenable to exact inference. Note that applying Equations 7.5 also requires exact inference, as one must compute the probabilities $\Pr_i(a_g)$ and $\Pr_i(a'_g)$.

7.1.3 Ground Recovery

Now that we can relax equivalences and compensate for their loss, the remaining question is which equivalences to relax. In general, deciding which equivalences to relax is hard, because it requires inference in the original model, which is intractable. Instead, Choi and Darwiche (2006) take the approach of relaxing every equivalence constraint and then incrementally recovering them as time and exact inference allow.

[2] In this more general case, there is no longer a guarantee that Equations 7.5 will converge to a fixed point. Convergence can be improved by using damping in Equations 7.5. Damping sets the new parameters to a weighted sum of the parameters from the previous iteration, and the updates as defined in Equations 7.5.

It follows from their results that when (i) relaxing all equivalence constraints, (ii) using the above compensation scheme and (iii) doing exact inference in the approximate model, the approximate marginals found correspond to the approximations found by *iterative belief propagation* (IBP, see Section 2.5.1) (Pearl, 1988). The connection to IBP is even stronger: the compensating weights computed in each iteration of Equations 7.5 exactly correspond to the messages passed by IBP.

Several heuristics have been proposed to decide which equivalences to recover, by performing inference in the relaxed model. We will work with the *residual recovery* heuristic (Choi and Darwiche, 2011). It is based on the practical observation that when IBP converges easily, the quality of its approximation is high. The heuristic tries to recover those edges whose convergence was most difficult in previous compensation steps. It recovers the equivalence that least satisfies Equation 7.4 after the compensation algorithm has converged. This is measured by taking the three-way symmetric KL divergence between the three terms of Equation 7.4.

7.2 Lifted RCR

We now introduce a lifted version of the relax, compensate and recover framework, which is meant to operate directly on first-order MLNs without having to ground them. Lifted RCR is based on first-order relaxation, compensation and recovery.

7.2.1 First-Order Relaxation

We begin with a first-order notion of relaxation where the goal is to ignore interactions between ground MLN formulas, yet without having to fully ground the MLN. This requires a first-order version of atom cloning and first-order equivalences.

Definition 7.1 (First-Order Cloning)**.** Cloning an atom occurrence in an MLN formula amounts to renaming the atom by concatenating its predicate with (i) an identifier of the formula, (ii) an identifier of the occurrence of the atom within the formula, and (iii) the logical variables appearing in the atom's formula.

Example 7.3. The first-order cloning of the atom occurrence $\text{smokes}(Y)$ in Formula 5.2′ gives

$$1.5 \quad \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}_{1b<X,Y>}(Y) \tag{7.6}$$

Here, 1 is an identifier of the formula, b is an identifier of the atom occurrence in the formula, and $<X, Y>$ are the logical variables appearing in the formula.

As in the ground case, each first-order cloning is associated with a corresponding equivalence between the original atom and its clone, except that the equivalence is first-order in this case.

Example 7.4. The first-order cloning of atom occurrence $\text{smokes}(Y)$ into $\text{smokes}_{1b<X,Y>}(Y)$ in the example above leads to introducing the following first-order equivalence:

$$\text{smokes}(Y) \Leftrightarrow \text{smokes}_{1b<X,Y>}(Y) \tag{7.7}$$

Let us now consider the groundings of Formulas 7.6 and 7.7, assuming a domain of $\{p, q\}$:

$$1.5 \quad \text{smokes}(p) \wedge \text{friends}(p, p) \Rightarrow \text{smokes}_{1b<p,p>}(p)$$

$$1.5 \quad \text{smokes}(p) \wedge \text{friends}(p, q) \Rightarrow \text{smokes}_{1b<p,q>}(q)$$

$$1.5 \quad \text{smokes}(q) \wedge \text{friends}(q, p) \Rightarrow \text{smokes}_{1b<q,p>}(p)$$

$$1.5 \quad \text{smokes}(q) \wedge \text{friends}(q, q) \Rightarrow \text{smokes}_{1b<q,q>}(b)$$

$$\text{smokes}(p) \Leftrightarrow \text{smokes}_{1b<p,p>}(p)$$

$$\text{smokes}(q) \Leftrightarrow \text{smokes}_{1b<p,q>}(q)$$

$$\text{smokes}(p) \Leftrightarrow \text{smokes}_{1b<q,p>}(p)$$

$$\text{smokes}(q) \Leftrightarrow \text{smokes}_{1b<q,q>}(q)$$

We have a few observations on the proposed cloning and relaxation techniques. First, the four groundings of Formula 7.6 contain distinct groundings of the clone $\text{smokes}_{1b<X,Y>}(Y)$. Second, if we relax the equivalence in Formula 7.7, the ground instances of Formula 7.6 will no longer interact through the

clone $\text{smokes}_{1b<X,Y>}(Y)$. Third, if we did not append the logical variables $<X, Y>$ during the cloning process, the previous statement would no longer be correct. In particular, without appending logical variables, the four groundings of Formula 7.6 would have contained only the two distinct clone groundings, $\text{smokes}_{1b}(p)$ and $\text{smokes}_{1b}(q)$. This would lead to continued interactions between the four groundings of Formula 7.6

Removing all interactions among groundings of the same formula is necessary for the following reasoning. Consider an MLN with the single formula,

$$w \quad \text{friends}(X, Y) \wedge \text{friends}(Y, Z) \Rightarrow \text{friends}(X, Z)$$

If we clone all atom occurrences, yet without appending logical variables, and then relax all equivalences, we would obtain the MLN $(w, \text{friends}_a(X, Y) \wedge \text{friends}_b(Y, Z) \Rightarrow \text{friends}_c(X, Z))$, which still has many interacting random variables at the ground level. As we will see in Chapter 6, there is currently no exact lifted inference algorithm that can handle this MLN without grounding it first. Furthermore, because this ground model has high treewidth for non-trivial domain sizes, it is also not amenable for exact inference by propositional algorithms. However, by cloning atoms in Lifted RCR, we are able to perform approximate lifted inference in this model.

The proposed cloning technique leads to MLNs in which one quantifies over predicate names (as in second-order logic). This can be avoided, but it leads to less transparent semantics. In particular, we can avoid quantifying over predicate names by using ground predicate names with increased arity. For example, $\text{smokes}_{1b<X,Y>}(Y)$ could have been written as $\text{smokes}_{1b}(X, Y)$ where we pushed $<X, Y>$ into the predicate arguments. The disadvantage of this, however, is that the semantics of the individual arguments is lost as the arguments become overloaded.

We now have the following key theorem.

Theorem 7.1. *Let Δ^r be the MLN resulting from cloning all atom occurrences in MLN Δ and then relaxing all introduced equivalences. Let Δ^g be the grounding of Δ^r. The formulas of Δ^g are then fully disconnected (i.e., they share no atoms).*

Proof. First, concatenating each predicate occurrence with an identifier for its formula causes all formulas to become disconnected. Second, cloned atoms contain all logical variables in the formula. For two groundings g_1, g_2 of a

formula, there is a difference in the assignment to at least one logical variable. Hence, there is a difference in the arguments of any pair of atoms a_1 from g_1 and a_2 from g_2. Therefore, a_1 and a_2 are distinct random variables and the groundings g_1 and g_2 are disconnected. □

With this theorem, the proposed first-order cloning and relaxation technique allows one to fully disconnect the grounding of an MLN by simply relaxing first-order equivalences in the first-order MLN.

7.2.2 First-Order Compensation

In principle, one can just clone atom occurrences, relax some equivalence constraints, and then use the resulting MLN as an approximation of the original MLN. By relaxing enough equivalences, the approximate MLN can be made arbitrarily easy for exact inference. Our goal in this section, however, is to improve the quality of approximations by compensating for the relaxed equivalences, yet without making the relaxed MLN any harder for exact inference. This will be done through a notion of first-order compensation.

Equiprobable Equivalences

The proposed technique is similar to the one for ground MLNs, that is, using *weighted atoms* whose weights will allow for compensation. The key, however, is to use first-order weighted atoms instead of ground ones. For this, we need to define the notion of equiprobable equivalences, based on the notion of equiprobable sets of random variables (Definition 5.4).

Definition 7.2 (Equiprobable Equivalence)**.** Let Δ be an MLN from which a first-order equivalence $a \Leftrightarrow a'$ was relaxed. Let $a_1 \Leftrightarrow a'_1, \ldots, a_n \Leftrightarrow a'_n$ be all groundings of $a \Leftrightarrow a'$. The equivalence $a \Leftrightarrow a'$ is equiprobable iff the sets $\{a_1, \ldots, a_n\}$ and $\{a'_1, \ldots, a'_n\}$ are both equiprobable w.r.t the ground distribution of MLN Δ.

The basic idea of first-order compensation is that when relaxing an equiprobable equivalence $a \Leftrightarrow a'$, under certain conditions, one can compensate for its loss using only two weighted first-order atoms of the form:

$$w \quad a \qquad \text{and} \qquad w' \quad a'$$

This follows because if we were to fully ground the equivalence into $a_1 \Leftrightarrow a'_1, \ldots, a_n \Leftrightarrow a'_n$ and then apply ground compensation, the relevant ground atoms will attain the same weights. That is, by the end of ground compensation, the weighted ground atoms,

$$w_i \quad a_i \qquad \text{and} \qquad w'_i \quad a'_i$$

will have $w_i = w_j$ and $w'_i = w'_j$ for all i and j.

Partitioning Equivalences

To realize first-order compensation, one must address two issues. First, a relaxed first-order equivalence may not be equiprobable to start with. Second, even when the equivalence is equiprobable, it may cease to be equiprobable as we adjust the weights during the compensation process. Recall that equiprobability is defined with respect to the ground distribution of an MLN. Yet, this distribution changes during the compensation process, which iteratively changes the weights of compensating atoms and, hence, also iteratively changes the ground distribution.

Example 7.5. Consider the following relaxed equivalences: $\mathrm{p}(X) \Leftrightarrow \mathrm{q}(X)$ and $\mathrm{q}(X) \Leftrightarrow \mathrm{r}(X)$. Suppose the domain is $\{a, b\}$ and the current ground distribution, $\Pr_i$, is such that $\Pr_i(\mathrm{p}(a)) = \Pr_i(\mathrm{p}(b))$, $\Pr_i(\mathrm{q}(a)) = \Pr_i(\mathrm{q}(b))$, and $\Pr_i(\mathrm{r}(a)) \neq \Pr_i(\mathrm{r}(b))$. In this case, the equivalence $\mathrm{p}(X) \Leftrightarrow \mathrm{q}(X)$ is equiprobable, but $\mathrm{q}(X) \Leftrightarrow \mathrm{r}(X)$ is not equiprobable.

If an equivalence constraint is not equiprobable, one can always partition it into a set of equiprobable equivalences — in the worst case, the partition will include all groundings of the equivalence. In the above example, one can partition the equivalence $\mathrm{q}(X) \Leftrightarrow \mathrm{r}(X)$ into the equivalences $\mathrm{q}(a) \Leftrightarrow \mathrm{r}(a)$ and $\mathrm{q}(b) \Leftrightarrow \mathrm{r}(b)$, which are trivially equiprobable.

Given this partitioning, the compensation algorithm will add distinct weights for the compensating atoms $\mathrm{q}(a)$ and $\mathrm{q}(b)$. Therefore, the set $\{\mathrm{q}(a), \mathrm{q}(b)\}$ will no longer be equiprobable in the next ground distribution, $\Pr_{i+1}$. As a result, the equivalence $\mathrm{p}(X) \Leftrightarrow \mathrm{q}(X)$ will no longer be equiprobable w.r.t. the ground distribution $\Pr_{i+1}$, even though it was equiprobable with respect to the previous ground distribution $\Pr_i$.

Strongly Equiprobable Equivalences

To attain the highest degree of lifting during compensation, one needs to dynamically partition equivalences after each iteration of the compensation algorithm, to ensure equiprobability. We defer the discussion on dynamic partitioning to Section 7.3.3, focusing here on a strong version of equiprobability that allows one to circumvent the need for dynamic partitioning.

The mentioned technique is employed by our current implementation of Lifted RCR, which starts with equivalences that are *strongly equiprobable.* An equivalence is strongly equiprobable if it is equiprobable w.r.t all ground distributions induced by the compensation algorithm (i.e., ground distributions that result from only modifying the weights of compensating atoms).

Example 7.6. Consider again Formula 5.2′ where we cloned the atom occurrence $\text{smokes}(Y)$ and relaxed its equivalence, leading to the MLN:

$$1.5 \;\; \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}_{1b<X,Y>}(Y)$$

and relaxed equivalence

$$\text{smokes}(Y) \Leftrightarrow \text{smokes}_{1b<X,Y>}(Y) \tag{7.8}$$

Suppose we partition this equivalence as follows:[3]

$$X = Y : \text{smokes}(Y) \Leftrightarrow \text{smokes}_{1b<X,Y>}(Y)$$

$$X \neq Y : \text{smokes}(Y) \Leftrightarrow \text{smokes}_{1b<X,Y>}(Y)$$

These equivalences are not only equiprobable w.r.t. the relaxed MLN, but also strongly equiprobable. That is, suppose we add to the relaxed model

[3]We are using an extension of MLNs that allows constraints, such as $X \neq Y$, to be associated with logical variables. Our implementation is in terms of parfactor graphs (Section 5.1.2), which do allow for the representation of such constraints. In extended MLNs, we will write $cs : \phi$ to mean that cs is a constraint set that applies to formula ϕ.

the compensating atoms

$$w_1 \quad \text{smokes}(X)$$

$$w_1' \quad X = Y : \ \text{smokes}_{1b<X,Y>}(Y)$$

$$w_2 \quad \text{smokes}(X)$$

$$w_2' \quad X \neq Y : \ \text{smokes}_{1b<X,Y>}(Y)$$

The two equivalences will be equiprobable w.r.t. any ground distribution that results from adjusting the weights of these compensating atoms.

We will present an equivalence partitioning algorithm in Section 7.3 that guarantees strong equiprobability of the partitioned equivalences. This algorithm is employed by our current implementation of Lifted RCR and will be used when reporting experimental results later.

7.2.3 Count-Normalization

We are one step away from presenting our first-order compensation scheme. What is still missing is a discussion of *count-normalized* equivalences, which are also required by our compensation scheme.

Example 7.7. Consider Equivalence 7.8, which has four groundings

$$\text{smokes}(p) \Leftrightarrow \text{smokes}_{1b<p,p>}(p)$$

$$\text{smokes}(q) \Leftrightarrow \text{smokes}_{1b<p,q>}(q)$$

$$\text{smokes}(p) \Leftrightarrow \text{smokes}_{1b<q,p>}(p)$$

$$\text{smokes}(q) \Leftrightarrow \text{smokes}_{1b<q,q>}(q)$$

for the domain $\{p, q\}$. There are two distinct groundings of the original atom $\text{smokes}(Y)$ in this case and each of them appears in two groundings. When each grounding of the original atom appears in exactly the same number of ground equivalences, we say that the first-order equivalence is count-normalized.

Now consider a constrained version of Equivalence 7.8

$$X \neq q \vee Y \neq q : \text{smokes}(Y) \Leftrightarrow \text{smokes}_{1b<X,Y>}(Y)$$

which has the following groundings

$$\text{smokes}(p) \Leftrightarrow \text{smokes}_{1b<p,p>}(p)$$

$$\text{smokes}(q) \Leftrightarrow \text{smokes}_{1b<p,q>}(q)$$

$$\text{smokes}(p) \Leftrightarrow \text{smokes}_{1b<q,p>}(p)$$

This constrained equivalence is not count-normalized since the atom $\text{smokes}(p)$ appears in two ground equivalences while the atom $\text{smokes}(q)$ appears in only one.

Generally, we have the following definition.

Definition 7.3. Let $(cs : a \Leftrightarrow a')$ be a first-order equivalence. Let θ be an instantiation of the variables $\mathbf{X}$ in the original atom a for which the constraint set cs is satisfiable. The equivalence is *count-normalized* iff for each instantiation θ, $cs\,\theta$ has the same number of solutions for the remaining variables $\mathbf{Y}$. More formally, the condition is as follows.

$$\forall \theta_1, \theta_2 \in \text{solutions}(cs, \mathbf{X}) : |\,\text{solutions}(cs\,\theta_1, \mathbf{Y})| = |\,\text{solutions}(cs\,\theta_2, \mathbf{Y})|$$

Moreover, the number of groundings for $cs : a$ is called the *original count* and the number of groundings for $cs : a'$ is called the *clone count*.

Count-normalization can only be violated by constrained equivalences. Moreover, for a certain class of constraints, count-normalization is always preserved. The algorithm we shall present in Section 7.3 for partitioning equivalences takes advantage of this observation. In particular, the algorithm generates constrained equivalences whose constraint structure guarantees count-normalization.

7.2.4 The Compensation Scheme

We now have the following theorem.

Theorem 7.2. *Let Δ_i be an MLN with relaxed equivalences $(cs: a \Leftrightarrow a')$ and, hence, corresponding compensating atoms:*

$$w_i \quad cs: a \qquad \text{and} \qquad w_i' \quad cs: a'$$

Suppose that the equivalences are count-normalized and strongly equiprobable. Let $(a_g \Leftrightarrow a_g')$ be one grounding of equivalence $(cs: a \Leftrightarrow a')$, let n be its original count and n' be its clone count. Consider now the MLN Δ_{i+1} obtained using the following updates:

$$w_{i+1} = \frac{n'}{n}\left(\log\left(\Pr_i(a_g')\right) - \log\left(\Pr_i(\neg a_g')\right) - w_i'\right)$$

$$w_{i+1}' = \log\left(\Pr_i(a_g)\right) - \log\left(\Pr_i(\neg a_g)\right) - w_i \tag{7.9}$$

The ground distribution of MLN Δ_{i+1} equals the one obtained by applying Ground RCR to MLN Δ_i.

Proof. It follows from Equations 7.5 that Ground RCR assigns the same compensating weights to the groundings of $a \Leftrightarrow a'$, because it is equiprobable. Each grounding of a' occurs in a single grounding of $a \Leftrightarrow a'$ and gets the same weight from Lifted or Ground RCR. From Definition 7.3 for count-normalization, each grounding of a occurs in n'/n groundings of $a \Leftrightarrow a'$. Ground RCR would add n'/n compensating weighted atoms which all get the same weight. Lifted RCR aggregates these ground weighted atoms in a single first-order weighted atom, and aggregates the weights for each grounding into a single weight by multiplying it with n'/n. Finally, because of *strong* equiprobability, the equivalences that were equiprobable in Δ_0 will also be equiprobable in Δ_i and the computed weights correspond to the weights of Ground RCR in each iteration i. □

Note that first-order compensation requires exact inference on the MLN Δ_i, which is needed for computing $\Pr_i(a_g)$ and $\Pr_i(a_g')$. Moreover, these computations will need to be repeated until one obtains a fixed point of the update equations given by Theorem 7.2. The key, however, is that one does not need to change the set of compensating atoms during the compensation process, given the strong equiprobability of relaxed equivalences.

7.2.5 First-Order Recovery

Recovering a first-order equivalence $(cs : a \Leftrightarrow a')$ amounts to removing its compensating atoms

$$w_i \quad cs : a \qquad \text{and} \qquad w_i' \quad cs : a'$$

and then adding the equivalence back to the MLN.

Adapting the ground recovery heuristic suggested earlier, one recovers the first-order equivalence that maximizes the symmetric pairwise KL-divergence[4]

$$n' \cdot \mathrm{KLD}\left(\Pr(a_g), \Pr(a_g'), \frac{e^{w_i + w_i'}}{1 + e^{w_i + w_i'}}\right),$$

where n' is the clone count of the equivalence. Note here that n' is also the number of equivalence groundings since, by definition, the clone atom contains all logical variables that appear in the equivalence.

Please note that recovering first-order equivalences may destroy the equiprobability of equivalences that continue to be relaxed. Hence, one generally needs to re-partition these relaxed equivalences.

7.3 Partitioning Equivalences

We will now discuss a method for partitioning first-order equivalences, which guarantees both strong equiprobability and count-normalization. It builds on the preemptive shattering algorithm to partition atoms into equiprobable sets. First, we review these results from earlier chapters. Second, we present the partitioning method that is used by our current implementation of Lifted RCR. It uses the atom partitions to partition first-order equivalences. Third, we discuss the problem of dynamic partitioning of equivalences.

[4] This is the sum of the KL-divergences between all pairs of arguments, in both directions.

7.3.1 Partitioning Atoms by Preemptive Shattering

Section 5.4.3 proposed an algorithm, called preemptive shattering, that identifies equiprobable atoms, on a purely syntactic basis. More precisely, Proposition 5.4 established that the preemptive shattering algorithm (Algorithm 8) of Section 4.3.1 can be used for this purpose.

Preemptive shattering looks at the constants K, domains $\mathcal{D}$ and free variables V that appear in the model. When using MLN models, the set of domains $\mathcal{D}$ and free variables V is always empty.[5] Hence, for the special case of MLN models, the preemptive shattering algorithm adds (in)equality constraints such that each logical variable in the MLN is either equal to exactly one constant in K, or different from all of them. It furthermore adds (in)equality constraints between logical variables that appear in the same literal, so that there is an inequality constraint between each pair of variables.

Example 7.8. Consider the formula $\text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$ and assume we have evidence $\text{smokes}(\mathit{alice})$ and therefore $K = \{\mathit{alice}\}$. Preemptive shattering of the input atom $\text{smokes}(X)$ returns back

$$X = a : \text{smokes}(X)$$

$$X \neq a : \text{smokes}(X)$$

Preemptive shattering of the input atom $\text{smokes}_{<X,Y>}(X)$ returns back

$$X = a \wedge Y = a : \text{smokes}_{<X,Y>}(X)$$

$$X = a \wedge Y \neq a : \text{smokes}_{<X,Y>}(X)$$

$$X \neq a \wedge Y = a : \text{smokes}_{<X,Y>}(X)$$

$$X \neq a \wedge Y \neq a \wedge X = Y : \text{smokes}_{<X,Y>}(X)$$

$$X \neq a \wedge Y \neq a \wedge X \neq Y : \text{smokes}_{<X,Y>}(X)$$

We will next show how this shattering procedure forms the basis of a method for partitioning equivalence constraints, with the aim of ensuring both strong equiprobability and count-normalization.

[5] All variables are quantified when transforming the MLN to a WFOMC representation, on which preemptive shattering is defined.

7.3.2 Partitioning Equivalences by Preemptive Shattering

Consider an MLN which results from cloning some atom occurrences and then adding corresponding equivalence constraints. Let K be all the constants appearing explicitly in the MLN.

To partition a first-order equivalence $a \Leftrightarrow a'$, our method will first apply preemptive shattering to the original atom a and clone atom a', yielding a partition for each. Suppose that $\{(cs_1 : a_1), \ldots, (cs_n : a_n)\}$ is the partition returned for original atom a. Suppose further that $\{(cs'_1 : a'_1), \ldots, (cs'_m : a'_m)\}$ is the partition returned for clone atom a'. By definition of cloning, all variables that appear in original atom a must also appear in clone atom a'. This implies the following property. For every (original) constraint cs_i, there is a corresponding set of (clone) constraints cs'_j that specialize, or partition cs_i. Each pair of constraints cs_i and cs'_j will then generate a member of the equivalence partition: $(cs_i \wedge cs'_j : a_i \Leftrightarrow a'_j)$. Note that $cs_i \wedge cs'_j \equiv cs'_j$ since cs'_j implies cs_i.

Example 7.9. Suppose we are partitioning the equivalence $\text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$. Example 7.8 showed the preemptive shattering of the atoms $\text{smokes}(X)$ and $\text{smokes}_{<X,Y>}(X)$. These give rise to the following equivalence partition:

$$X = a \wedge Y = a : \text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$$

$$X = a \wedge Y \neq a : \text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$$

$$X \neq a \wedge Y = a : \text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$$

$$X \neq a \wedge Y \neq a \wedge X = Y : \text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$$

$$X \neq a \wedge Y \neq a \wedge X \neq Y : \text{smokes}(X) \Leftrightarrow \text{smokes}_{<X,Y>}(X)$$

We now have the following results.

Lemma 7.3. *Partitioning by preemptive shattering returns equiprobable equivalences.*

Proof outline. Let $cs \wedge cs' : a \Leftrightarrow a'$ be an element of the partition of an equivalence constraint found by preemptive shattering. The constrained atoms $(cs : a)$ and $(cs' : a')$ themselves were found by preemptive shattering of the MLN Δ. It follows from Proposition 5.4 that the groundings of $(cs : a)$ and $(cs' : a')$ are equiprobable. □

Lemma 7.4. *Partitioning by preemptive shattering returns count-normalized equivalences.*

Proof outline. Let $cs : a \Leftrightarrow a'$ be an element of the partition of an equivalence constraint found by preemptive shattering. Let $\mathbf{X}$ be the set of logical variable arguments of a and $\mathbf{X} \cup \mathbf{Y}$ be the logical variable arguments of its clone a'. For a each grounding of a, that is, each substitution of the variables $\mathbf{X}$ by constants that satisfy cs, the clone a' has the same number of groundings of the variables $\mathbf{Y}$ that satisfy cs. This follows from the fact that each logical variable that is not bound to a constant has the same set of inequality constraints associated with it. □

Lemma 7.5. *Partitioning by preemptive shattering returns strongly equiprobable equivalences.*

Proof outline. The preemptive shattering procedure neither depends on the weight parameters of the MLN model Δ, nor on the precise formulas in Δ. It only depends on the constants K that appear in it. The partition returned by preemptive shattering does not introduce any constants that were not in the input Δ. Therefore, the MLNs Δ_i that are constructed in each iteration of the compensation algorithm (with compensating weighted atoms) do not introduce additional constants and each call to preemptive shattering returns identical partitions in each iteration of the compensation algorithm. □

Theorem 7.6. *Partitioning by preemptive shattering returns count-normalized, strongly equiprobable equivalences.*

Proof. Follows from Lemmas 7.3, 7.4 and 7.5. □

7.3.3 Dynamic Equivalence Partitioning

To attain the highest degree of lifting, one may need to dynamically partition equivalences after each iteration of the compensation scheme. Moreover, one would need to find the smallest possible partition for each considered equivalence, while guaranteeing both equiprobability and count-normalization.[6]

[6] There is a unique smallest partition satisfying these properties.

Dynamic partitioning leads to a higher degree of lifting as it removes the need for strong equiprobability, which usually leads to larger partitions of equivalences. We do not employ dynamic partitioning in our current implementation of Lifted RCR, leaving this to future work. We point out, however, that dynamic partitioning requires a slight adjustment to the compensation scheme given by Theorem 7.2.

Suppose that the first-order equivalence $cs : a \Leftrightarrow a'$ was equiprobable in MLN Δ_i, but ceases to be equiprobable in MLN Δ_{i+1} due to the adjustment of weights for compensating atoms. A dynamic partitioning scheme will then partition this equivalence into a set of equiprobable and count-normalized equivalences. One implication of this partitioning is that the two compensating atoms associated with the equivalence $cs : a \Leftrightarrow a'$ in MLN Δ_i will now have to be partitioned as well. That is, the compensating atoms in MLN Δ_i

$$w_i : \quad cs : a \qquad \text{and} \qquad w_i' : \quad cs : a'$$

will need to be replaced in MLN Δ_{i+1} by two compensating atoms for each equivalence in the computed partition. Moreover, the initial weights of these new compensating atoms will be precisely w_i (for original atoms) and w_i' for cloned atoms.

7.4 Related and Future Work

In this section, we discuss related propositional and lifted approximate inference algorithms and identify future work on the problem of equivalence shattering.

7.4.1 Relation to Propositional Algorithms

The RCR framework has previously been used to characterize loopy belief propagation and some of its generalizations. In the case that the simplified model is fully disconnected, the approximate marginals of RCR correspond to the approximate marginals provided by *iterative belief propagation* (Pearl, 1988; Choi and Darwiche, 2006). The approximation to the partition function further corresponds to the *Bethe free energy* approximation (Yedidia, Freeman, and Weiss, 2003; Choi and Darwiche, 2008). When equivalence constraints have been recovered, RCR corresponds to a class of *generalized belief propagation* (GBP)

approximations (Yedidia, Freeman, and Weiss, 2003), in particular *iterative joingraph propagation* with the corresponding joingraph free energies (Aji and McEliece, 2001; Dechter, Kask, and Mateescu, 2002). RCR further inspired a system that was successfully employed in a recent approximate inference competition (Elidan and Globerson, 2010; Choi and Darwiche, 2011). *Mini-bucket* approximation can be viewed as an instance of RCR where no compensations are used (Kask and Dechter, 2001; Dechter and Rish, 2003; Choi, Chavira, and Darwiche, 2007). In these cases, RCR is also capable of providing upper bounds on the partition function. Since every approximate MLN found by Lifted RCR corresponds to a solution found by Ground RCR on the ground MLN, the above mentioned results carry over to the lifted setting.

7.4.2 Relation to Lifted Algorithms

The motivation for calling our approach *lifted* is threefold. First, in the compensation phase, we are compensating for many ground equivalences at the same time. Computing compensating weights for all of these requires inferring only a *single* pair of marginal probabilities. Second, computing marginal probabilities is done by an *exact lifted* inference algorithm. Third, we relax and recover first-order equivalence constraints, which correspond to *sets* of ground equivalences.

The work on lifted approximate inference has mainly focused on lifting the IBP algorithm. The correspondence between IBP and RCR carries over to their lifted counterparts: Lifted RCR compensations on a fully relaxed model correspond to *lifted belief propagation* (lifted BP) (Jaimovich, Meshi, and Friedman, 2007; Singla and Domingos, 2008). Starting from a first-order model, Singla and Domingos (2008) proposed *lifted network construction* (LNC), which partitions the random variables and factors of a factor graph into so-called *supernodes* and *superfeatures*. The ground atoms represented by these supernodes send and receive the same messages when running IBP. Kersting, Ahmadi, and Natarajan (2009) proposed a *color-passing* (CP) algorithm that achieves similar results as LNC, only starting from a ground model, where the first-order structure is not apparent.

Bisimulation-based lifted inference (Sen, Deshpande, and Getoor, 2009) uses a mini-bucket approximation on a model that was compressed by detecting symmetries. Because of the correspondence between Ground RCR and mini-buckets mentioned above, also this approach can be seen as an instance of Lifted RCR with the compensation phase removed.

Lifted BP was the first approximate lifted inference approach and dominated the field for some time. In the last year, however, several alternatives have been proposed. One line of work performs lifted *importance sampling* on the search space of the probabilistic theorem proving algorithm (Gogate and Domingos, 2011; Gogate, Jha, and Venugopal, 2012). This search space is closely related to a FO-da-DNNF circuit, as discussed in Section 4.7.2. Two lifted *Markov chain Monte Carlo* algorithms were recently proposed, one starting from propositional models (Niepert, 2012c; Niepert, 2012a) and another from first-order models (Venugopal and Gogate, 2012). Finally, there has been recent work on lifting *variational* inference algorithms (Choi and Amir, 2012; Bui, Huynh, and Riedel, 2012).

7.4.3 Opportunities for Equivalence Partitioning

For models that contain few explicit constants (K is small), preemptive shattering will find partitions of equivalence constraints that are close to minimal. When K is large, however, it will create large partitions, defeating the purpose of lifted inference.

The work on lifted BP provides us with alternative partitioning algorithms that work correctly on a fully relaxed model. These algorithms construct lifted networks whose nodes send and receive the same messages when running IBP. This means that they partition the atoms into equiprobable sets (w.r.t. the approximate distribution) and that LNC can be used for equivalence partitioning in Lifted RCR, when the model is fully relaxed.

However, preemptive shattering is the only partitioning algorithm to our knowledge that works for any level of relaxation. One way to make the partitions found by preemptive shattering smaller (with fewer sets) is by exploiting the actual structure of the model Δ itself. Knowing that there are certain independences in Δ might remove the need to ground atoms for all constants in K. This requires reasoning on the model, as is done by the vanilla shattering algorithm (see Section 4.3.3). We believe our work can motivate future work on finding more efficient general partitioning algorithms.

Much of the work on lifted BP has looked at more intelligently constructing lifted networks. Given the strong connection between LNC and partitioning of equiprobable equivalences, these high-level ideas can directly be applied to Lifted RCR, as we discuss next.

The problem of dynamic equivalence partitioning is related to *anytime lifted BP* (de Salvo Braz et al., 2009; Freedman et al., 2012), which gradually shatters its partitions of random variables and factors with each iteration of lifted BP. In addition, it makes these partitions specific to a query and propagates bounds on the marginals, in order to stop message passing early, when the desired approximation quality has been reached. This would correspond to a version of RCR where compensating weights are not numbers but intervals that are tightened with each iteration of the compensation algorithm.

Several ways of constructing an *approximate lifted network* were proposed (Singla, Nath, and Domingos, 2010; Kersting et al., 2010). This corresponds to partitioning into approximately equiprobable equivalences in the context of Lifted RCR. Approximating the partition can be done by ignoring long-range dependencies in the model or by looking at the actual parameters of the model, instead of only its logical syntax. Another line of work looks at efficient lifted network construction for multiple queries with *changing evidence* (Nath and Domingos, 2010; Ahmadi, Kersting, and Hadiji, 2010; Hadiji, Ahmadi, and Kersting, 2011; Ahmadi, Kersting, and Sanner, 2011). These techniques also allow for the efficient computation of joint marginals.

7.5 Experiments

In this section, we evaluate the Lifted RCR algorithm on common benchmarks from the lifted inference literature. The experiments were set up to answer the following questions:

(Q1) To what extent does recovering first-order equivalences improve the approximations found by Lifted RCR?

(Q2) Can IBP be improved considerably through the recovery of a small number of equivalences?

(Q3) Is there a significant advantage to using Lifted RCR over Ground RCR?

(Q4) What is the run time behavior of the compensation algorithm?

7.5.1 Implementation

We implemented Lifted RCR in the WFOMC tool.[7] To compute exact marginal probabilities in Equations 7.9, we use exact lifted inference by first-order knowledge compilation (cf., Section 5.5). In combination with preemptive shattering, we compile a first-order circuit once for every equiprobable set of random variables (cf., Section 5.4.3) and re-evaluate it in each iteration of the compensation algorithm. This is possible because the structure of the compensated MLNs does not change between iterations, only their parameters change. Re-evaluating an already compiled first-order circuit can be done very efficiently. In the compensation phase, we use damping with a weight of 50%.

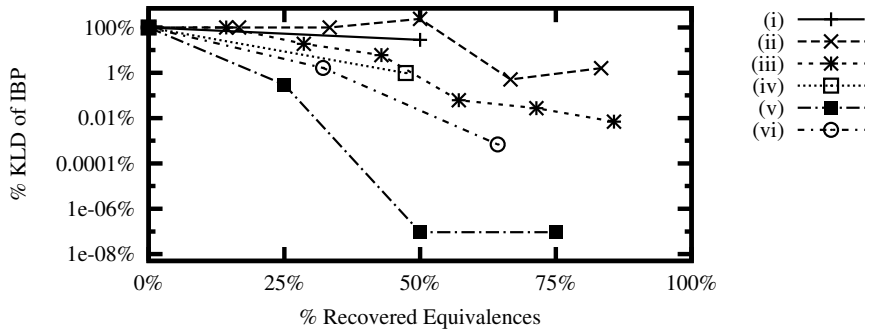
In practice, the Ground RCR algorithm does not start off with a fully relaxed model. Instead, it starts with some equivalences intact, such that the relaxed model forms a spanning tree, and exact inference is still efficient. As long as the relaxed model is a tree, the set of approximate single marginals that can be found with RCR correspond to the set of marginals that the loopy BP algorithm can converge to in the original model. Relaxing equivalences beyond a spanning tree neither makes inference more tractable, nor does it change the approximations made by RCR.

For these reasons, we do not clone all atoms occurrences and atom groundings in the relaxation step of Lifted RCR. Instead, we clone all but one atom per MLN formula. This guarantees that the relaxed model is still a tree (but not necessarily spanning). To select the atom that is not cloned, we choose one with a high number of logical variable arguments, to have as few equivalences relaxed as possible overall. As a consequence of this approach to relaxation, weighted unit clauses are never relaxed.

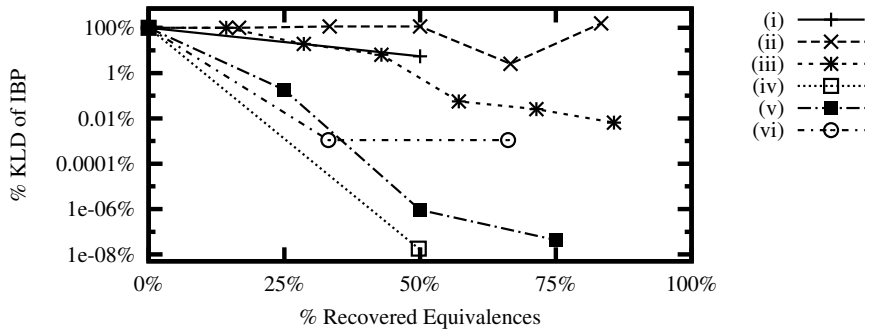
7.5.2 Results

To answer **(Q1-2)** we ran Lifted RCR on MLNs from the exact lifted inference literature, where computing exact marginals is tractable. This allows us to evaluate the approximation quality of Lifted RCR for different degrees of relaxation. We used the models *p-r* and *sick-death* (de Salvo Braz, Amir, and Roth, 2005), *workshop attributes* (Milch et al., 2008), *smokers* (Singla and Domingos, 2008), *smokers and drinkers* (Van den Broeck et al., 2011a) and *symmetric smokers*

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



(a) Small domains - 100 random variables



(b) Large Domains - 10,000 random variables

Figure 7.1: Normalized approximation error of Lifted RCR for different levels of approximation on the models (i) *p-r*, (ii) *sick-death*, (iii) *workshop attributes*, (iv) *smokers*, (v) *smokers and drinkers* and (vi) *symmetric smokers*.

(Van den Broeck, 2011b). Each of these models represents a new advance in exact lifted inference. They are incrementally more complex and challenging for lifted inference.

The results are shown in Figure 7.1, where we ran Lifted RCR on the above models for two sets of domain sizes: a small and a large set, where the number of random variables is on the order of 100 and 10,000 respectively. We plot the symmetric KL divergence between the exact marginals and the approximations found by Lifted RCR, as a percentage of the KL divergence of the fully relaxed approximation (the IBP approximation). The horizontal axis shows the level of relaxation in terms of the percentage of recovered ground equivalences. The 0% recovery point corresponds to the approximations found by (lifted) IBP. The

100% recovery point corresponds to exact inference.[8]

We see that each recovered first-order equivalence tends to improve the approximation quality significantly, often by more than one order of magnitude, answering **(Q1)**. In the case of *smokers* with a large domain size, recovering a single equivalence more than the IBP approximation reduced the KL divergence by 10 orders of magnitude, giving a positive answer to **(Q2)**. The *sick-death* model is the only negative case for **(Q2)**, where recovering equivalences does not lead to approximations that are better than IBP.[9]

To answer **(Q3)**, first note that as argued in Section 7.4, the 0% recovery point of Lifted RCR using LNC or CP to partition equivalences corresponds to lifted BP. For this case, the work of Singla and Domingos (2008) and Kersting, Ahmadi, and Natarajan (2009) has extensively shown that Lifted BP/RCR methods can significantly outperform Ground BP/RCR. Similarly, computational gains for the 100% recovery point were shown in the exact lifted inference literature. For intermediate levels of relaxation, we ran Ground RCR on the above models with large domain sizes. On these, Ground RCR could never recover more than 5% of the relaxed equivalences before exact inference in the relaxed model becomes intractable. This answers **(Q3)** positively.

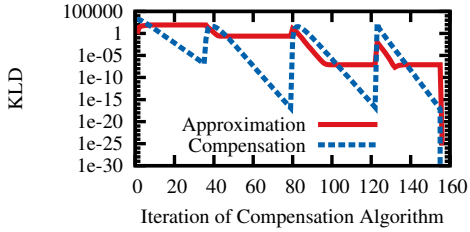
To answer **(Q4)**, we report on the quality of the approximations and the convergence of the compensation algorithm, both as a function of run time and the number of iterations. The results for the four most challenging benchmarks (small and large *smokers and drinkers* and *symmetric smokers*) are shown in Figure 7.2.

The figures on the left show convergence as a function of the number of iterations of the compensation algorithm.

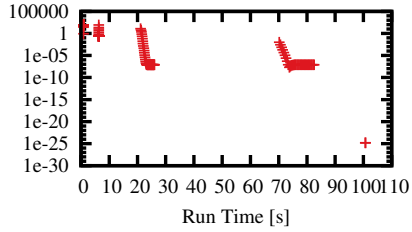
- The solid line shows the KL divergence between the approximate marginals (according to the relaxed MLN and compensating weights of that iteration) and the exact marginals (computed with exact lifted inference).
- The dashed line shows the three-way pairwise symmetric KL divergence between the three terms of Equation 7.4. Our compensation algorithm aims to satisfy this equation, lowering the reported KLD.

[8] All experiments ran up to the 100% point, which is not shown in the plot because it has a KL divergence of 0.

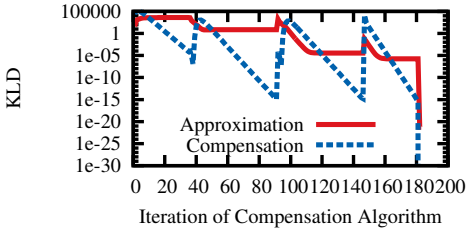
[9] Interestingly, it is also the only example where some compensations failed to converge without using damping.



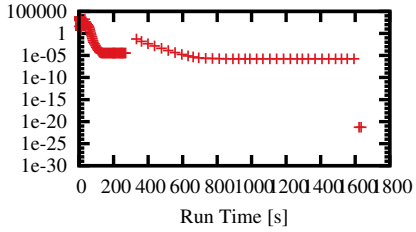
(a) Smokers and Drinkers (Small) - Iterations



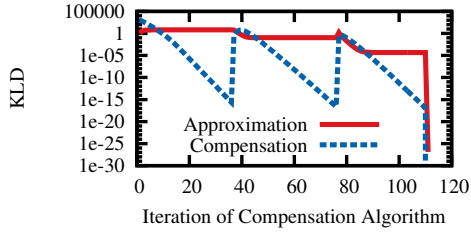
(b) Smokers and Drinkers (Small) - Time



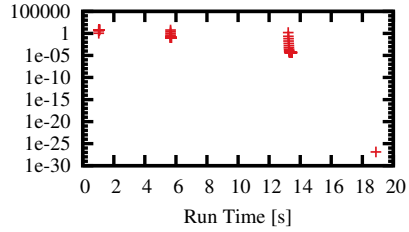
(c) Smokers and Drinkers (Large) - Iterations



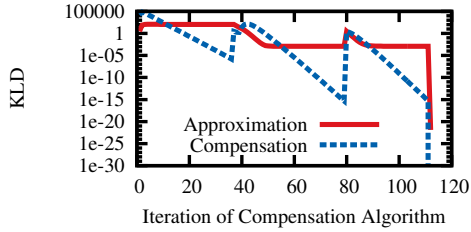
(d) Smokers and Drinkers (Large) - Time



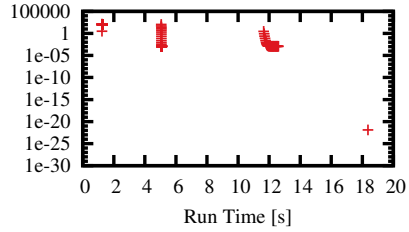
(e) Symmetric Smokers (Small) - Iterations



(f) Symmetric Smokers (Small) - Time



(g) Symmetric Smokers (Large) - Iterations



(h) Symmetric Smokers (Large) - Time

Figure 7.2: Convergence Rate of Lifted RCR

We see that the compensation KLD increases dramatically between certain iterations. This happens when we recover a relaxed first-order equivalence. After a recovery step, the compensating weights still stem from the previous compensation iteration. These weights no longer satisfy Equation 7.4. The compensation KLD is increased and the compensation algorithm starts searching for a better set of compensating weights. In the iterations where recovery happens, we also see a change in the quality of the approximation. It can improve, because of the added equivalence, but it can also decrease, because the compensating weights no longer satisfy Equation 7.4. In either case, we see that running the compensation algorithm after recovering an equivalence improves both the compensation objective and the quality of the approximations. The KLD of the approximation right before each recovery corresponds to the errors reported in Figure 7.1.

The figures on the right show the same quality of approximation (solid line to the left), this time as a function of time. In between reported data points, there are large gaps. These again correspond to recovery steps, where the algorithm needs to compile a set of new circuits to perform exact inference in the new approximate, relaxed model. Evaluating these circuits in the compensation phase (to compute all marginals) is typically fast, leading to a quick succession of iterations. For one experiment, in Figure 7.2d, the compilation is not the bottleneck. This is because evaluating the compiled circuits for its large domain size becomes expensive when most equivalences have been recovered.

As a final observation, the iterations on the left side become increasingly time-consuming. For example, in Figure 7.2a, the first 80 iterations out of 160 take less than 10 seconds out of 100 in Figure 7.2b. Similarly, in Figure 7.2c, the first 150 iterations out of 180 take 300 seconds out of 1800 in Figure 7.2d. To complement our earlier claim that it suffices to recover a small number of equivalences in order to significantly improve upon IBP approximations, this also shows that recovering these first few equivalences can be done very efficiently.

7.6 Conclusions

This chapter presented Lifted RCR, a lifted approximate inference algorithm that performs exact inference in a simplified model. We showed how to obtain a simplified model by relaxing first-order equivalences, compensating for their loss, and recovering them as long as exact inference remains tractable. The

algorithm can traverse an entire spectrum of approximations, from lifted iterative belief propagation to exact lifted inference. Inside this spectrum is a family of lifted joingraph propagation (and GBP) approximations. We empirically showed that recovering first-order equivalences in a relaxed model can substantially improve the quality of an approximation. We also remark that Lifted RCR relies on an exact lifted inference engine as a black box, and that with Lifted RCR, any future advances in exact lifted inference have immediate impact in lifted approximate inference.

8

Lifted Learning

This chapter addresses the problem of learning the parameters, or weights associated with each formula in a MLN model from data. Lifted inference algorithms improve the efficiency of inference by exploiting symmetries in statistical relational models and reasoning about groups of objects. However, when learning the parameters of these models, one often resorts to reasoning over individual objects. We propose to harness the full power of relational representations in the learning phase by using lifted inference.

Ideally, generative weight learning of undirected models is performed by convex optimization of the likelihood of the parameters. However, computing this likelihood and its gradient is intractable for most models. Therefore, the standard approach is to optimize an approximate objective such as *pseudolikelihood*. Instead, this chapter investigates using lifted inference in order to efficiently learn weights that maximize the *actual likelihood*, in a lifted manner.

We start by reviewing existing weight learning algorithms for Markov logic in **Section 8.1**. The next two sections put forward our lifted learning approach. First, **Section 8.2** provides a generic overview of how lifted inference can be integrated into weight learning. A key insight from lifting is the possibility to identify indistinguishable groups of objects, which can be reasoned about as a whole and significantly reduce the number of inferences needed to compute

gradients. Furthermore, lifting can compute gradients and likelihoods in time polynomial in the size of the data, as opposed to exponential when using classical methods. Second, **Section 8.3** describes a concrete weight learning algorithm for Markov logic based on weighted first-order model counting (WFOMC). Its most appealing property for weight learning is that compilation only needs to be performed once. On each iteration of convex optimization, the circuit can be reparametrized with updated weights to compute a new likelihood and its gradient.

Finally, **Section 8.4** evaluates our approach on three standard real-world SRL data sets. We find that our algorithm learns models with better test-set likelihood than two competing approaches. We can employ *exact* lifted inference, because the generative setting does not require conditioning on evidence, which is #P-hard as it breaks the symmetries in the model. This is a new application of exact lifted inference algorithms to *real-world* data.

The work in this chapter is under review as

> G. Van den Broeck, W. Meert, and J. Davis (2012). *Lifted parameter learning for Markov logic*. (submitted)

8.1 Weight Learning for Markov Logic

Several algorithms have been proposed for learning the weights associated with each formula in Markov logic (Singla and Domingos, 2005; Richardson and Domingos, 2006; Lowd and Domingos, 2007), as well as for learning the formulas themselves (Kok and Domingos, 2005; Mihalkova and Mooney, 2007; Huynh and Mooney, 2009). The input to these algorithms is a set of training examples DB. Each training example is a database that assigns a truth value to each ground atom. This corresponds to the *learning from interpretations* setting in inductive logic programming (De Raedt and Džeroski, 1994). This chapter focuses on the weight learning task. Weight learning uses data to automatically learn the weight associated with each feature (formula) by optimizing a given objective function.

Ideally, each candidate weight vector w would be scored by its training set log-likelihood

$$\log \Pr_w(DB) = \sum_{i=1}^{N} \log \Pr_w(DB_i), \tag{2.6'}$$

where N is the number of examples, DB_i is the ith database in DB and $\Pr_w(DB_i)$ is defined by the MLN semantics (see Section 5.1.1):

$$\Pr_w(db) = \frac{1}{Z} \exp\left(\sum_{i}^{|\Psi|} w_i n_i(db)\right). \tag{5.1'}$$

For MLNs, the log-likelihood is a convex function of the weights and learning can be solved via convex optimization. The derivative of the log-likelihood (Richardson and Domingos, 2006) with respect to the jth feature is

$$\frac{\partial}{\partial w_j} \log \Pr_w(db) = n_j(db) - \mathbb{E}_w[n_j], \tag{8.1}$$

where $n_j(db)$ is the number of true groundings of formula ψ_j in the training data and $\mathbb{E}_w[n_j]$ is computed using the current weight vector. The jth component of the gradient is simply the difference between the empirical counts of the jth feature in the database and its expectation according to the current model.

Thus, each iteration of weight learning must perform inference on the current model to compute the expectations. This is often computationally infeasible. As for propositional undirected graphical models, optimizing pseudo-log-likelihood (Equation 2.7) is a tractable alternative to optimizing the likelihood of a MLN. This is currently the standard approach for generative weight learning for Markov logic.

In the context of discriminative learning, there has been work on optimizing the *conditional* log-likelihood. The most advanced work for this setting is by Lowd and Domingos (2007). They propose several approaches, the best of which is a second-order method called pre-conditioned scaled conjugate gradient (PCSG). They use MC-SAT (Poon and Domingos, 2006), which is a slice-sampling Markov chain Monte Carlo method, to approximate the expected counts. In principle, this approach could be used to learn maximum likelihood weights by setting the query set to include all the variables in the domain. To the best of our

knowledge, this has yet to be attempted until now. In Section 8.4, we compare against this approach.

Independent of our work, Ahmadi, Kersting, and Natarajan (2012) recently proposed the use of *approximate* lifted inference, namely lifted belief propagation (Jaimovich, Meshi, and Friedman, 2007; Singla and Domingos, 2008; Kersting, Ahmadi, and Natarajan, 2009), in a stochastic gradient optimization approach to piecewise *discriminative* weight learning. Together with this work, our approach is the first application of lifted inference to learning, opening up new possibilities for lifted techniques.

8.2 Lifted Generative Weight Learning

In this section, we provide a general overview of how lifted inference can be used in the context of weight learning. This approach yields *two benefits*:

First Benefit Leveraging insights from the lifted inference literature allows weight learning to compute a small number of marginals to compute the gradient.

Second Benefit Each query is computed more efficiently using lifted inference. Namely, it is polynomial in the size of the databases, that is, in the number of objects in the databases, whereas propositional inference is in general exponential in this size.

To illustrate the intuition behind the approach, assume that we want to learn the weight for a MLN that contains a single formula $w : \psi(X_1, \ldots, X_n)$. Assume we have access to a lifted inference oracle that can efficiently compute the marginal probability of any random variable (ground atom) in the MLN, even for large domain sizes. Learning a weight that maximizes the probability of the data is challenging because it requires computing $\mathbb{E}_w[n_\psi]$ at each iteration of an optimization algorithm, by summing over the probability of each possible grounding of ψ:

$$\mathbb{E}_w[n_\psi] = \Pr(\psi(X_1, \ldots, X_n)\theta_1) + \cdots + \Pr(\psi(X_1, \ldots, X_n)\theta_m) \tag{8.2}$$

where θ_i is a grounding substitution which replaces all the logical variables (X_i) in the atom by constants.

We now first review some lifted probabilistic inference techniques. Second, we will show how to use these techniques to efficiently compute Equation 8.2.

8.2.1 Equiprobable Random Variables

In the absence of evidence, many of the queries in an MLN will be equiprobable, because of the symmetries imposed by the model. Lifted inference excels at answering these types of queries. In fact, one of the key insights from lifted inference is that we can partition the set of random variables into equiprobable sets (Definition 5.4) by purely syntactic operations on the first-order model (see Section 5.4.3). The procedure is based on the preemptive shattering algorithm (Algorithm 8) of Section 4.3.1. This algorithm can also be used for finding equiprobable instances of a quantifier-free MLN formula.

Example 8.1. To review this point, consider again the model

$$w \quad \text{smokes}(X) \wedge \text{friends}(X, Y) \Rightarrow \text{smokes}(Y) \tag{5.2''}$$

which states that smokers are more likely to be friends with other smokers. Assuming a domain of three constants, *alice*, *bob*, and *charlie*, the atoms $\text{friends}(\textit{alice}, \textit{bob})$ and $\text{friends}(\textit{bob}, \textit{charlie})$ are equiprobable, but the atoms $\text{friends}(\textit{alice}, \textit{alice})$ and $\text{friends}(\textit{bob}, \textit{charlie})$ are not. The first two have identical probabilities because they are indistinguishable w.r.t. the MLN. Intuitively, this can be seen by looking at a permutation of the constants, mapping *alice* into *bob* and *bob* into *charlie*. This permutation turns $\Pr(\text{friends}(\textit{alice}, \textit{bob}))$ into $\Pr(\text{friends}(\textit{bob}, \textit{charlie}))$, whereas the MLN model is invariant under this permutation (it does not even explicitly mention the constants). Therefore, these atoms are indistinguishable and they must have the same marginal probability.

The same reasoning applies to instances of the entire formula: $\text{smokes}(\textit{alice}) \wedge \text{friends}(\textit{alice}, \textit{bob}) \Rightarrow \text{smokes}(\textit{bob})$ is equiprobable with

$$\text{smokes}(\textit{bob}) \wedge \text{friends}(\textit{bob}, \textit{charlie}) \Rightarrow \text{smokes}(\textit{charlie})$$

but not with

$$\text{smokes}(\textit{alice}) \wedge \text{friends}(\textit{alice}, \textit{alice}) \Rightarrow \text{smokes}(\textit{alice})$$

The latter happens to be a tautology and thus has probability 1.

This technique can be applied to a model with an arbitrary number of formulas. If the model does not mention specific constants, the algorithm enumerates all ways in which the logical variables in the same atom can be equal or different. More formally, $\Pr(\psi(X_1,\ldots,X_n)\theta_k) = \Pr(\psi(X_1,\ldots,X_n)\theta_l)$ when two arguments $X_i\theta_k = X_j\theta_k$ iff $X_i\theta_l = X_j\theta_l$. If the model itself mentions certain unique information about specific constants, we can still partition atoms into equiprobable sets, but finding such sets gets slightly more complicated. Section 5.4.3 contains the full details on this procedure.

8.2.2 Evaluating Expected Counts

We will now show how to use lifted inference techniques to compute the gradient.

First Benefit

To achieve the *first benefit*, we identify equiprobable sets. This allows us to reduce the number of queries (Equation 8.2) that need to be answered during weight learning. For each set of equiprobable queries, only one representative query needs to be answered as each other query in the group will have the same marginal probability. Answering these queries simply requires calculating the probability that each formula is true according to the model, and does not involve conditioning on the data. The only way in which the probabilities depend on the data is through the domain size of each variable, that is, the number of objects in the world.

Let $\mathcal{P} = \{E_1,\ldots E_q\}$ denote the equiprobable partition found for formula $\psi(X_1,\ldots,X_n)$ by preemptive shattering and let $\psi(X_1,\ldots,X_n)\theta_{E_i}$ be some element of E_i. We can then reduce the computation of Equation 8.2 to computing

$$\mathbb{E}_w[n_\psi] = |E_1| \cdot \Pr(\psi(X_1,\ldots,X_n)\theta_{E_1}) + \cdots + |E_q| \cdot \Pr(\psi(X_1,\ldots,X_n)\theta_{E_q}), \tag{8.3}$$

involving as many queries as there are elements in the partition (i.e., $|\mathcal{P}| = q$).

In the multiple database setting (e.g., when performing cross-validation), it is necessary to compute the gradient for each database, as the domain size and therefore the size of each E_i can vary according to the fold. The size of

the partition $|\mathcal{P}|$, however, does not depend on the domain size, only on the structure of the MLN formulas.

Theorem 8.1. *Given b databases and an equiprobable partition $\mathcal{P}$, evaluating the gradient of the likelihood (Equation 8.1) requires computing $b \cdot |\mathcal{P}|$ marginal probabilities.*

In the special case of a single formula with n logical variables, the number of queries we need to pose is the Bell number B_n (Bell, 1934; Rota, 1964).

Definition 8.1 (Bell Number)**.** Bell numbers are recursively defined as

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \quad \text{with} \quad B_0 = B_1 = 1.$$

The Bell number B_n represents the number of partitions of n elements into non-empty sets. In our case, it is the number of equivalence relations on the n logical variables in the formula, which does not depend on the size of the domains or database. Assuming a domain size of D, that same formula will have D^n groundings and computing Equation 8.1 without using these insights from lifted inference would require answering D^n queries.

When more generally, we have multiple formulas that do not explicitly mention any constants from the database, the analysis is also easy, based on properties of the preemptive shattering algorithm.

Proposition 8.2. *A Markov logic network with k formulas $(w_1, \psi_1(X_1^1, \ldots, X_1^{n_1}))$ to $(w_k, \psi_k(X_k^1, \ldots, X_k^{n_k}))$ without constants has an equiprobable partition $\mathcal{P}$ such that $|\mathcal{P}| = \sum_{i=1}^{k} B_{n_i}$.*

For this case, Equation 8.2 requires computing $\sum_{i=1}^{k} D^{n_i}$ marginals per database, whereas lifted learning requires computing $\sum_{i=1}^{k} B_{n_i}$, essentially removing the dependency on the size of the database from Equation 8.2. This difference can be significant. Typically, formulas have a bounded number of variables n_i, on the order of between two and four, which gives Bell numbers $B_2 = 2$, $B_3 = 5$ and $B_4 = 15$. Databases in SRL, on the other hand, typically describe relations between hundreds of objects (D), resulting in models with tens of thousands of random variables. This is also true for the databases used in Section 8.4. In the most general case where the MLN being learned explicitly mentions certain constants, the analysis becomes more complex. However, the size of the

partition found by preemptive shattering will still be polynomial in the number of such constants and independent of the number of constants in the entire domain and in the databases used for learning.

Second Benefit

The *second benefit* of lifted learning over its propositional counterpart is the *complexity* of inference for each query. When using a lifted inference oracle that is *domain-lifted* (Definition 5.2), computing the probabilities in Equation 8.3 will be polynomial in the domain size, and therefore polynomial in the size of the databases. On the other hand, when doing propositional inference to compute the same numbers, inference is in general exponential in the domain size. For most non-trivial MLNs, the treewidth (Robertson and Seymour, 1983; Robertson and Seymour, 1986) of its corresponding probabilistic graphical models is a polynomial of the domain size and propositional inference is exponential in this treewidth.

8.3 Lifted Learning by Knowledge Compilation

The previous section assumed the presence of a lifted inference oracle. We will now look at the implications of choosing one particular algorithm, namely lifted probabilistic inference by *first-order knowledge compilation* (see Section 5.5), which reduces the probabilistic inference task to a *weighted first-order model counting* (WFOMC) problem on a corresponding theory Δ with weight functions w_T and w_F.

To illustrate the benefits of using a knowledge compilation approach, we first consider computing the gradient of the likelihood for a single formula. Then we present our full lifted weight learning algorithm.

In the single formula case, computing the expected number of groundings of $\psi(X_1, \ldots, X_n)$ requires estimating $\Pr(\psi(X_1, \ldots, X_n)\theta_{E_i})$ once for each equiprobable partition. Following Equation 5.8, WFOMC solves this by computing the ratio

$$\frac{\text{WFOMC}(\psi(X_1, \ldots, X_n)\theta_{E_i} \wedge \Delta, w_T, w_F)}{\text{WFOMC}(\Delta, w_T, w_F)}$$

Notice that each partition has the same denominator $\text{WFOMC}(\Delta, w_T, w_F)$, which corresponds to the partition function of the MLN. If we have q equiprobable partitions, evaluating the weighted model counts requires compiling $q + 1$ circuits: one for each equiprobable partition, and one for the partition function. These circuits are *independent* of the weights and domains of the first-order model and can therefore be used to compute Equation 8.3 for any database size and weight vector. Thus, each circuit can be *reused on each iteration of weight learning*. This exemplifies the idea behind the knowledge compilation approach to inference: transform the model once from a representation where a certain task is hard to a representation where the task is easy, and reuse that representation to solve multiple problems.

Algorithm 17 outlines our Lifted Weight Learning (LWL) approach. It takes a MLN Δ and a set of databases DB as inputs and returns a weight vector $\bar{w}$. The algorithm works as follows. First, it builds all the circuits needed to compute the likelihood and its gradient. It compiles one circuit for Δ to compute the partition function. Then it preemptively shatters each weighted formula ψ in Δ to identify its equiprobable partition. It compiles one circuit for every equiprobable partition of the formula. Second, it runs an iterative procedure to learn the weights. During each iteration i of the convex optimization, it computes the gradient of the likelihood given the current weights $\bar{w}_i$. First it computes the partition function. Then, for each of the b databases, the expected counts for each formula are calculated by reevaluating the compiled circuit associated with every one of the formula's equiprobable partitions. Traditionally, this is the most challenging step in computing Equation 8.1 (the gradient). The algorithm terminates when a stop condition is met (e.g., after a predefined number of iterations).

The computational saving of employing first-order knowledge compilation during inference can be characterized as follows. Over t iterations of the convex optimization algorithm, instead of answering $t \cdot b \cdot |\mathcal{P}|$ hard inference queries, as done by a vanilla lifted learning algorithm, the knowledge compilation approach performs $1 + |\mathcal{P}|$ hard compilation steps and reuses each circuit $t \cdot b$ times by reevaluating it for different weight vectors w_i and domain sizes D.

For the special case of a single formula with n logical variables, by using both knowledge compilation and lifted inference, we went from answering $t \cdot b \cdot D^n$ queries that are exponential in the size of the databases to $1 + B_n$ compilations that are independent of the training examples and $t \cdot b$ circuit evaluations that are polynomial in the size of the databases.

Algorithm 17 LIFTEDWEIGHTLEARNING(Δ, DB)

Input.

Δ: A set of MLN formulas with initial weights
DB: A set of databases.

Supporting Functions.

COMPILE	Compile to FO-sda-DNNF circuit (Algorithm 1, p. 84)
SHATTER	Partition into equiprobable sets (Algorithm 8, p. 96)
WFOMC	Compute weighted first-order model count (Def. 5.5, p. 153)
LBFGS	Convex optimization algorithm (Liu and Nocedal, 1989)

Function.

let D^{db} be the domain sizes in database db
let n_ψ^{db} be the number of true groundings of formula ψ in database db
let $\bar{w}$ be the initial weight vector of Δ
$C_Z \leftarrow \text{COMPILE}(\Delta)$ *// Compile circuits*
for each $\psi \in \Delta$ **do**
  $\mathcal{P}_\psi \leftarrow \text{SHATTER}(\Delta, \psi)$ *// Partition into equiprobable sets*
  for each $E \in \mathcal{P}_\psi$ **do**
    for some $\alpha \in E$ **do**
      $C_E \leftarrow \text{COMPILE}(\Delta \wedge \alpha)$
repeat *// Optimize weights*
  $\mathcal{L} \leftarrow 0$ *// Log-likelihood*
  $\nabla\mathcal{L} \leftarrow \bar{0}$ *// Log-likelihood gradient vector*
  for each $db \in DB$ **do**
    $Z \leftarrow \text{WFOMC}(C_Z, \bar{w}, D^{db})$
    $\mathcal{L} \leftarrow \mathcal{L} - \log(Z)$
    for each $\psi_i \in \Delta$ **do**
      $\mathcal{L} \leftarrow \mathcal{L} + \bar{w}_i \cdot n_{\psi_i}^{db}$
      $\nabla\mathcal{L}_i \leftarrow \nabla\mathcal{L}_i + n_{\psi_i}^{db}$
      for each $E \in \mathcal{P}_\psi$ **do**
        $p \leftarrow \text{WFOMC}(C_E, \bar{w}, D^{db})/Z$
        $\nabla\mathcal{L}_i \leftarrow \nabla\mathcal{L}_i - |E| \cdot p$
  $\bar{w} \leftarrow \text{LBFGS}(\mathcal{L}, \bar{w}, \nabla\mathcal{L})$
until convergence
return $\bar{w}$

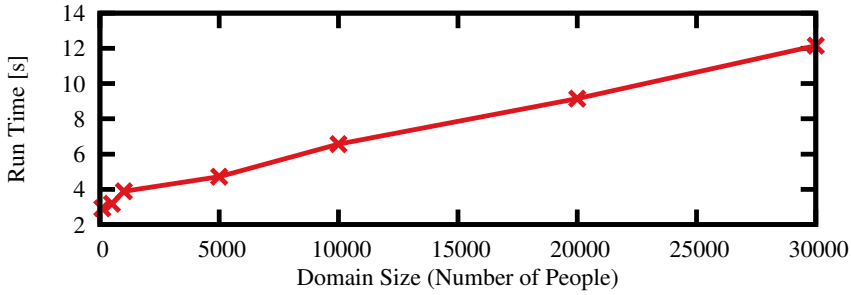


Figure 8.1: Run time of learning weights for the friends and smokers model as a function of the domain size of the training database.

8.4 Empirical Evaluation

We evaluate our approach on one synthetic domain and three real-world datasets.

8.4.1 Synethetic Data: Scaling Behavior

We use the friends and smokers model from Example 8.1 (Singla and Domingos, 2005) as a controlled enviroment to explore how our algorithm scales with respect to domain size. We vary the number of people in the domain from 100 to 30,000 and randomly generate a training database for each size. The results are shown in Figure 8.1. This model is fully liftable and training time is polynomial in the domain size. The largest database, for domain size $30,000$, assigns truth values to $30,000^2 + 30,000$ or approximately 900 million ground atoms.

8.4.2 Real-World Data: Test-Set Likelihood

We use three real-world datasets to compare the following three MLN weight learning algorithms:

PLL optimizes the pseudolikelihood of the data (Equation 2.7) via the LBFGS algorithm (Liu and Nocedal, 1989) and is the default generative weight

learning algorithm in the Alchemy package (Kok et al., 2008).

PSCG is a discriminative weight learning algorithm (Lowd and Domingos, 2007) in Alchemy. However, it generatively optimizes the likelihood of the data when all predicates are treated as query atoms (i.e., the evidence set is empty).

LWL is our proposed approach. It uses our implementation of lifted probabilistic inference by first-order knowledge compilation to compute the likelihood and its gradient and the limited-memory BFGS optimizer in the Factorie system (McCallum, Schultz, and Singh, 2009). Our implementation of LWL is available as open-source software.

The goal of this empirical evaluation is to compare whether exactly optimizing the likelihood is better than optimizing the approximated likelihood. Additionally, we are interested in whether learned theories, and not just hand-crafted ones, can be compiled.

Datasets and Methodology

We first briefly describe the datasets we use.[1] The **IMDB** data comes from the IMDB.com website. The data set contains information about attributes (e.g., gender) and relationships among actors, directors, and movies. The data is divided into five different folds. The **UWCSE** data contains information about the University of Washington CSE Department. The data contains information about students, professors, and classes and models relationships (e.g., TAs and Advisor) among these entities. The data consists of five folds, each one corresponding to a different group in the CSE Department. The **WebKB** data consists of Web pages from the computer science departments of four universities (Craven and Slattery, 2001). The data has information about words that appear on pages, labels of pages and links between Web pages. There are four folds, one for each university. We limit the number of words considered on each fold to the ten with the highest information gain with respect to a page's class.

[1] All datasets are available on http://alchemy.cs.washington.edu

UWCSE			
	PSCG	PLL	LWL
BUSL	-1671	-2564	**-1523**
	-684	-909	**-541**
	-1702	-2871	**-1283**
	-3291	-2660	**-2565**
	-3399	-5280	**-2362**
MSL	-25745	-1519	**-1497**
	-1070	-535	**-524**
	-21936	-1213	**-1209**
	-2756	-2472	**-2471**
	-51903	**-2261**	-2274

IMDB			
	PSCG	PLL	LWL
BUSL	-379	-1190	**-377**
	-580	-1241	**-558**
	-907	-1581	-968
	-291	-660	**-284**
	-608	-539	**-267**
MSL	-1641792	-9672	**-831**
	-1690616	-12471	**-766**
	-32286	-18242	**-1307**
	-229063	-1354	**-698**
	-16250	-982	**-700**

WebKB			
	PSCG	PLL	LWL
MSL	-9453	-5860	**-5655**
	-27628	-5129	**-5047**
	-26548	-4135	**-3917**
	-18052	-4367	**-4280**

Table 8.1: Log-likelihoods for models learned in the UWCSE, IMDB and WebKB domains.

To generate a set of models, we use two standard MLN structure learning algorithms: BUSL (Mihalkova and Mooney, 2007), which works bottom-up, and MSL (Kok and Domingos, 2005), which works top-down. During structure learning, we limit each clause to contain at most four literals and three variables.[2] In all domains, we perform cross-validation and hold out one fold as the test set, and use the remaining folds to learn the model. Each fold serves as a test set once. Given the learned structure, each weight learning algorithm uses the same data that produced the structure to learn weights. Then we compute the test-set likelihood for each learned model. We always use WFOMC to compute the test-set likelihood so the only difference among the three approaches is *how* the weights were learned.

Results and Discussion

Table 1 reports results for all learned models. LWL consistenly outperforms both PLL and PSCG in terms of test-set likelihood. It loses once to PLL and once to PSCG. The magnitude of differences varies. LWL does better than PLL if the pseudolikelihood assumption is violated (e.g., long chains of interactions). For some of the learned theories this results in large differences between LWL and PLL. For models that contain mostly nearly deterministic formulas, LWL and PLL tend to have similar test-set likelihoods. For example, MSL learns a number of such formulas for the UWCSE domain, which express statements like 'all people are either students or professors'. PSCG performance is quite variable. PSCG performs very poorly for some models as MC-SAT can sometimes give very bad estimates of the gradient. This can occur if MC-SAT fails to converge (e.g., because it gets stuck in a single mode of the distribution).

LWL is able to compile all the learned theories, except for those learned by BUSL on WebKB. These theories, while theoretically compilable, are very large as they contain more than 50 learned complex formulas. The compilation ran out of 25 GB of memory. In terms of run time, on average PLL takes 1 second, LWL takes 2 minutes, and PSCG takes 40 minutes. For LWL, about 25% of the time is spent on compilation.

8.5 Conclusions

We investigated the effect of lifted inference for parameter learning in the statistical relational learning setting. Specifically, we proposed a weight learning algorithm for Markov logic based on weighted first-order model counting. Calculating the gradient of the likelihood is the crucial step in parameter learning. Applying insights from lifted inference allows us to compute the gradient exactly while querying fewer marginals and answering each query more efficiently. Lifted weight learning yields a further benefit in that its compiled circuits are independent of the database and can be reused over all databases and iterations during optimization. Our proposed approach was evaluated on learned models from three real-world data sets. Our approach consistently resulted in better test-set likelihoods than two approximate weight learning algorithms.

[2]The objective function for structure learning is pseudo-likelihood.

Conclusions

We conclude by summarizing the presented work, restating its main contributions, and providing an outlook on future research.

Thesis Summary

The success of fields such as statistical relational learning and probabilistic logic learning depends on efficient algorithms to reason about highly structured and uncertain information. The expressivity of the statistical relational models used in these fields comes at a significant cost: inference is often intractable, especially when using traditional automated reasoning techniques. The recently emerged area of automated reasoning that deals with this type of problems is called *lifted probabilistic inference*. The algorithms developed in that area draw upon techniques developed in two disparate fields: first-order knowledge representation and reasoning, and probabilistic graphical models.

Contributions

This thesis contributes several algorithms, techniques and theoretical results in first-order logical reasoning and lifted probabilistic inference. Furthermore, it investigates the use of these techniques for machine learning. We identify five main contributions:

1. First-order knowledge compilation into negation normal form
2. Lifted probabilistic inference by weighted first-order model counting and first-order knowledge compilation
3. A formal framework for lifted probabilistic inference as a well-defined problem
4. A lifted relax, compensate & recover framework for approximate lifted inference
5. The application of exact lifted inference to lifted learning

In logical reasoning, our main contribution was a new data structure to represent first-order logical theories, called first-order negation normal form circuits, raising the problem of *first-order knowledge compilation* into this new language. We distinguished subclasses of circuits that permit tractable model counting inference and identified theoretical properties of these new languages, including a classification of supported polytime queries and transformations. In order to use the new data structure, we proposed an algorithm that compiles logical theories into these tractable circuit forms.

In lifted probabilistic inference, our most important contribution was a formal definition of lifted inference in terms of complexity considerations, called domain-lifted inference, and a first completeness result that identifies a non-trivial class of inference problems where lifted inference is guaranteed to work. This was formulated as the following claim.

Claim. Computing single marginal probabilities in quantifier-free models with *up to two logical variables per formula* is amenable to lifted inference.

These theoretical results were supported by a new lifted algorithm that reduces probabilistic inference in several statistical relational languages to a *weighted first-order model counting* problem on logical theories. This algorithm made use

of the first-order knowledge compilation techniques that we developed for logical reasoning.

The above claim about the feasibility of exact lifted inference was part of a more general formal framework for lifted inference, which includes two alternative definitions of lifted inference, the notion of completeness of a lifted inference algorithm w.r.t. a class of inference problems and the notion of liftability of these classes. With these concepts, we drew an initial liftability map for inference in statistical relational models, establishing several complexity bounds.

Even when using lifted inference techniques, performing exact inference is an unattainable goal in many statistical relational models. We therefore proposed a new approximate lifted inference algorithm called *lifted relax, compensate & recover*. It has the advantage that it allowed us to use exact lifted inference algorithms to perform approximate inference in a relaxed, simplified model. This algorithm led to a spectrum of approximations, corresponding to lifted versions of several propositional inference algorithms. A theoretical contribution of this work was a unifying semantics for different existing approximate lifted inference approaches.

Finally, we looked at the implications that our new results in automated reasoning have on the problem of *learning* the parameters of statistical relational models from data. Here, we identified several opportunities to make learning algorithms scale better with the size of the training data. Because of this increased efficiency, we were able to optimize the exact likelihood of learned models, where this was previously impossible.

Evidence – A Recurring Theme

The problem of efficiently dealing with *evidence* was a recurring theme throughout this dissertation, as this has always been a major issue for lifted inference. It occurred for the first time in the analysis of first-order circuits, where we looked at the conditioning transformation, which incorporates new evidence into a logical knowledge base. The problem came back in the description of our lifted probabilistic inference algorithm, where we introduced a technique to efficiently compute posterior probabilities for certain types of evidence. The theme was most central in our liftability framework, where we defined DQE-lifted inference in terms of the complexity of inference as a function of, among others, the size of the evidence, where we classified

inference problems based on classes of evidence, and analyzed which classes allow for DQE-lifted inference. Among other results, this permitted us to make the following claim.

Claim. The problem of computing *conditional probabilities* in general is not liftable by any inference algorithm. It becomes liftable when the evidence consists solely of *unary atoms* and *propositions*.

Despite these clear limitation, we identified a powerful positive interplay between lifted inference techniques and evidence in the work on lifted learning. In that context, we were able to learn statistical relational models from large, real-world data sets.

Discussion, Perspectives and Future Work

The work presented in this thesis has significantly advanced the area of lifted inference, practically and theoretically. Practically, it proposed several new efficient algorithms, to compile and perform inference in first-order circuits, for exact and approximate lifted inference while exploiting local structure, for conditioning on certain types of evidence, and to learn the parameters of statistical relational models. We hope that these practical advances will contribute to lifted inference techniques being used in more real-world applications in the future. Theoretically, the work has advanced our understanding of lifted inference and first-order knowledge compilation, their algorithms and the fundamental strengths and limitations of these techniques. This thesis has opened up new avenues, but closed others. We will now discuss long-term perspectives and concrete topics of future work.

First-Order Knowledge Compilation

The avenues that were opened include the problem of first-order knowledge compilation to negation normal form circuits and its possible applications. Our development of first-order knowledge compilation had a single purpose in mind, namely weighted first-order model counting. Consequently, the set of circuit languages we proposed is not nearly as rich as the circuits used for propositional knowledge compilation. For example, it is not yet clear how binary and sentential decision diagrams, DNNF circuits, Horn approximations and other

standard propositional techniques fit into our view on first-order knowledge compilation. Our initial characterization of the properties of first-order negation normal form is still far away from a first-order knowledge compilation map.

We believe that in the next decade, a much broader first-order knowledge map will be developed in the NNF tradition. We expect to see first-order knowledge compilation being applied to a much richer set of first-order languages, including description logics, logic programs and modal logics. Our hope is that first-order circuits can play a meaningful role in application areas of knowledge compilation other than probabilistic inference, such as verification and planning.

As a first step towards this goal, we propose the following extensions of our work. A major limitation of our proposed compilation algorithm is that it does not support existentially quantified input. For precisely this reason, our proposed probabilistic inference algorithm cannot deal with most directed statistical relational models, whose weighted first-order model counting formulation uses existential quantifiers. It is feasible to extend at least some compilation rules to also support existential quantification in a lifted manner, although this would raise new questions of completeness and liftability. Both in the definition of our circuit languages and the compilation algorithm, there are many assumptions that can be relaxed in order to pose new problems. For example, what happens if we allow for function symbols and infinite interpretations? What if we extend the expressivity of the constraint language?

A Theory of Lifted Inference

Around the time our work on lifted inference started, there was considerable skepticism about lifted probabilistic inference. For many it was a hollow word, and anyone could in principle call his algorithm "lifted" to make it more appealing. We believe this was largely due to a lack of formal grounds for lifted inference and its algorithms. In part by the work presented here, on domain-lifted inference and the liftability framework, we believe that such skepticism is no longer justified. Whether the notion of domain-lifted inference will be found useful and stand the test of time is unsure. In the next decade, it may be refined or replaced. However, there will from now on always be a formal definition of lifted inference to work with. The problem has been stated, and the discussion can move forward.

Another avenue that was closed in this thesis is the overly optimistic idea that in the future, lifted inference algorithms will be able to efficiently deal with any type of query, in any type of statistical relational model. The liftability framework and its initial results on the feasibility of lifted inference provide a rich stream of new research questions. We expect that it will always be possible to more precisely define which classes of inference problems are liftable and to design new tractable languages. The most pressing open question in this regard is whether the three-variable fragment of statistical relational models is liftable, or any other class that includes transitive relations.

The theoretical limitations that were identified here allow us to put forth concrete new challenges. To overcome the problem of efficiently dealing with evidence on binary relations, we want to look at techniques to condition on more specific types of relations, such as functions. Our negative theoretical results do not necessarily apply to these more restricted cases. Results in this direction would vastly increase the applicability of lifted inference to real-world problems. It is clear that certain types of binary evidence do not form a problem for lifted inference. For example, conditioning on all ground atoms for one relation being true does not pose any challenge, even though this is evidence about a large number of binary atoms. We want to investigate the complexity of conditioning on evidence in terms of other properties of the evidence, leading to stricter complexity bounds. A related opportunity appears in the lifted relax compensate & recover algorithm, where the largest remaining challenge is to develop algorithms that efficiently partitioning equiprobable equivalences in the presence of evidence. Here, we can seek inspiration in the work on lifted belief propagation.

Lifted Inference and Learning Algorithms

A first breakthrough for automated probabilistic reasoning came at the end of the 1980s, with the advent of inference algorithms for probabilistic graphical models. These algorithms have in common that they *exploit conditional independencies* in the model to speed up inference. In terms of the number of random variables, this speedup can be exponential, when comparing to inference by naively enumerating possible worlds. A second breakthrough came at the end of the 1990s, when it was realized that *exploiting context-specific independencies*, determinism, and local structure in general can speed up probabilistic inference significantly. Again, this speedup can be exponential. We

believe that before long, it will be generally accepted that *exploiting symmetries* in the model is the next major advance in speeding up automated probabilistic reasoning, not only in statistical relational models, but also in standard probabilistic graphical models. As we also showed here, these speedups can again be exponential in terms of the domain size of logical variables, and hence in terms of the number of random variables.

Through the weighted first-order model counting framework that we proposed for lifted probabilistic inference, advances that will be made in first-order knowledge compilation will directly lead to improvements in exact lifted inference. Through our relax, compensate and recover framework, lifted approximate inference can in turn directly benefit from such progress.

Finally, we have only scratched the surface of using lifted techniques for machine learning. It is our hope that symmetries can play a similar role to sparsity in machine learning, as a means of regularization in very large hypothesis spaces. The problem of *learning structures* that are liftable is completely unexplored. For lifted learning of statistical relational models, possible approaches would be to learn theories in the two-variable fragment, or to learn first-order negation normal form circuits directly. In probabilistic graphical models, a lifted learner could give preference to network structures that are more symmetric. Furthermore, there are opportunities in using approximate lifted inference, such as lifted relax, compensate & recover, for the parameter learning task.

Bibliography

B. Ahmadi, K. Kersting, and F. Hadiji (2010). "Lifted belief propagation: pairwise marginals and beyond". In: *Proceedings of the 5th European Workshop on Probabilistic Graphical Models (PGM–10)*. Helsinki, Finland (cit. on p. 208).

B. Ahmadi, K. Kersting, and S. Natarajan (2012). "Lifted online training of relational models with stochastic gradient methods". In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)* (cit. on p. 218).

B. Ahmadi, K. Kersting, and S. Sanner (2011). "Multi-evidence lifted message passing, with application to pagerank and the Kalman filter". In: *Proceedings of the twenty-second international joint conference on Artificial Intelligence (IJCAI)*. AAAI Press, pp. 1152–1158 (cit. on p. 208).

S. M. Aji and R. J. McEliece (2001). "The generalized distributive law and free energy minimization". In: *Proceedings of the 39th Allerton Conference on Communication, Control and Computing*, pp. 672–681 (cit. on p. 206).

U. Apsel and R. I. Brafman (2011). "Extended lifted inference with joint formulas". In: *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 11–18 (cit. on pp. 133, 144, 159).

U. Apsel and R. I. Brafman (2012b). "Lifted MEU by weighted model counting". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Palo Alto, California, USA (cit. on p. 136).

F. Bacchus (1991). "Representing and reasoning with probabilistic knowledge: a logical approach to probabilities". In: (cit. on p. 135).

F. Bacchus, S. Dalmao, and T. Pitassi (2009). "Solving #SAT and Bayesian inference with backtracking search". In: *Journal of Artificial Intelligence Research* 34(1), pp. 391–442 (cit. on p. 31).

F. Bacchus, A. Grove, J. Halpern, and D. Koller (1996). "From statistical knowledge bases to degrees of belief". In: *Artificial Intelligence* 87(1), pp. 75–143 (cit. on p. 135).

R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi (1997). "Algebraic decision diagrams and their applications". In: *Formal methods in system design* 10(2), pp. 171–206 (cit. on p. 78).

J. S. Baras and G. Theodorakopoulos (2010). *Path Problems in Networks*. Vol. 3. Synthesis Lectures on Communication Networks. Morgan & Claypool Publishers (cit. on p. 31).

P. Baumgartner (2000). "FDPLL – a first-order Davis-Putnam-Logeman-Loveland procedure". In: *Automated Deduction-CADE-17*, pp. 200–219 (cit. on pp. 75, 122).

P. Baumgartner and C. Tinelli (2008). "The model evolution calculus as a first-order dpll method". In: *Artificial Intelligence* 172(4), pp. 591–632 (cit. on pp. 75, 122).

P. Beame, H. Kautz, and A. Sabharwal (2004). "Towards understanding and harnessing the potential of clause learning". In: *Journal of Artificial Intelligence Research* 22, pp. 319–351 (cit. on pp. 74, 75).

E. Bell (1934). "Exponential numbers". In: *American Mathematical Monthly*, pp. 411–419 (cit. on p. 221).

R. Bellman (1957). *Dynamic programming*. Princeton University Press (cit. on p. 4).

J. Besag (1975). "Statistical Analysis of Non-Lattice Data". In: *The Statistician* 24, pp. 179–195 (cit. on p. 36).

A. Biere (2009). "Bounded Model Checking". In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. Chap. 14 (cit. on p. 17).

H. Blockeel and L. De Raedt (1998). "Top-down induction of first-order logical decision trees". In: *Artificial Intelligence* 101(1-2), pp. 285–297 (cit. on p. 76).

E. Börger, E. Grädel, and Y. Gurevich (2001). *The classical decision problem*. Springer Verlag (cit. on p. 41).

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller (1996). "Context-specific independence in Bayesian networks". In: *Proceedings of the twelfth international conference on uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 115–123 (cit. on p. 26).

C. Boutilier, R. Reiter, M. Soutchanski, S. Thrun, et al. (2000). "Decision-theoretic, high-level agent programming in the situation calculus". In: *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, pp. 355–362 (cit. on p. 4).

R. Brachman and H. Levesque (2004). *Knowledge representation and reasoning*. Morgan Kaufmann (cit. on pp. 40, 46).

R. Bryant (1986). "Graph-based algorithms for Boolean function manipulation". In: *IEEE Transactions on Computers* 100(8), pp. 677–691 (cit. on pp. 18, 92).

H. Bui, T. Huynh, and S. Riedel (2012). "Automorphism groups of graphical models and lifted variational inference". In: (cit. on p. 207).

W. Buntine (1994). "Operations for learning with graphical m odels". In: *Journal of Artificial Intelligence Research* 2, pp. 159–225 (cit. on p. 135).

M. Cadoli and F. Donini (1997). "A survey on knowledge compilation". In: *AI Communications* 10(3-4), pp. 137–150 (cit. on p. 17).

M. Campbell, A. Hoane, and F. Hsu (2002). "Deep blue". In: *Artificial intelligence* 134(1), pp. 57–83 (cit. on p. 2).

M. Chavira and A. Darwiche (2005). "Compiling Bayesian networks with local structure". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 19, p. 1306 (cit. on p. 26).

M. Chavira and A. Darwiche (Apr. 2008). "On probabilistic inference by weighted model counting". In: *Artificial Intelligence* 172(6-7), pp. 772–799. (Cit. on pp. 19, 25, 26, 28, 158).

M. Chavira, A. Darwiche, and M. Jaeger (May 2006). "Compiling relational Bayesian networks for exact inference". In: *International Journal of Approximate Reasoning* 42(1-2), pp. 4–20. (Cit. on pp. 19, 139, 158).

J. Chen and S. Muggleton (2009). "Decision-theoretic logic programs". In: *Proceedings of ILP*. (Cit. on p. 136).

A. Choi, M. Chavira, and A. Darwiche (2007). "Node splitting: a scheme for generating upper bounds in Bayesian networks". In: *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 57–66 (cit. on p. 206).

A. Choi and A. Darwiche (2006). "An edge deletion semantics for belief propagation and its practical impact on approximation quality". In: *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pp. 1107–1114 (cit. on pp. 34, 190, 191, 205).

A. Choi and A. Darwiche (2008). "Approximating the partition function by deleting and then correcting for model edges". In: *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 79–87 (cit. on p. 205).

A. Choi and A. Darwiche (2011). "Relax, Compensate and then Recover". In: *New Frontiers in Artificial Intelligence*. Vol. 6797. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 167–180 (cit. on pp. 33, 192, 206).

J. Choi and E. Amir (2012). "Lifted relational variational inference". In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on p. 207).

J. Choi, E. Amir, and D. Hill (2010). "Lifted inference for relational continuous models". In: *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on p. 159).

J. Choi, R. de Salvo Braz, and H. Bui (2011). "Efficient methods for lifted inference with aggregate factors". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pp. 1030–1036 (cit. on p. 159).

K. Clark (1978). "Negation as failure". In: *Logic and data bases* 1, pp. 293–322 (cit. on pp. 21, 149).

D. Cohen, P. Jeavons, C. Jefferson, K. Petrie, and B. Smith (2005). "Symmetry definitions for constraint satisfaction problems". In: *Principles and Practice of Constraint Programming (CP)*, pp. 17–31 (cit. on p. 144).

G. Cooper and E. Herskovits (1992). "A Bayesian method for the induction of probabilistic networks from data". In: *Machine learning* 9(4), pp. 309–347 (cit. on p. 35).

M. Craven and S. Slattery (2001). "Relational learning with statistical predicate invention: better models for hypertext". In: *Machine Learning Journal* 43(1/2), pp. 97–119 (cit. on p. 226).

J. Crawford, M. Ginsberg, E. Luks, and A. Roy (1996). "Symmetry-breaking predicates for search problems". In: pp. 148–159 (cit. on p. 144).

J. M. Crawford and A. B. Baker (1994). "Experimental results on the application of satisfiability algorithms to scheduling problems". In: *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 1092–1097 (cit. on p. 17).

N. Dalvi, K. Schnaitter, and D. Suciu (2010). "Computing query probability with incidence algebras". In: *Proceedings of ACM SIGMOD/PODS Conference*, pp. 203–214 (cit. on p. 19).

A. Darwiche (1999). "Compiling knowledge into decomposable negation normal form". In: *Proceedings of IJCAI*. Vol. 16, pp. 284–289 (cit. on p. 49).

A. Darwiche (2001a). "Decomposable negation normal form". In: *Journal of the ACM (JACM)* 48(4), pp. 608–647 (cit. on p. 49).

A. Darwiche (2001b). "On the tractability of counting theory models and its application to belief revision and truth maintenance". In: *Journal of Applied Non-Classical Logics* 11(1-2), pp. 11–34 (cit. on pp. 18, 19, 30, 119).

A. Darwiche (2001c). "Recursive conditioning". In: *Artificial Intelligence* 126(1), pp. 5–41 (cit. on pp. 25, 160).

A. Darwiche (2003). "A differential approach to inference in Bayesian networks". In: *Journal of the ACM (JACM)* 50(3), pp. 280–305 (cit. on p. 30).

A. Darwiche (2004). "New advances in compiling CNF to decomposable negation normal form". In: *Proceedings of ECAI*, pp. 328–332 (cit. on pp. 121, 125, 164).

A. Darwiche (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press. (Cit. on pp. 21, 32, 35).

A. Darwiche (2011). "SDD: a new canonical representation of propositional knowledge bases". In: *Proceedings of the twenty-Second international joint conference on artificial Intelligence (IJCAI)*. AAAI Press, pp. 819–826 (cit. on pp. 18, 77).

A. Darwiche and P. Marquis (2002). "A knowledge compilation map". In: *Journal of Artificial Intelligence Research* 17, pp. 229–264 (cit. on pp. 17, 18, 20, 50–56, 63, 64, 70, 72, 119).

A. Darwiche (2000). "Model-based diagnosis under real-world constraints". In: *AI Magazine, Summer* 21, pp. 57–73 (cit. on p. 17).

M. Davis, G. Logemann, and D. Loveland (1962). "A machine program for theorem-proving". In: *Communications of the ACM* 5(7), pp. 394–397 (cit. on pp. 74, 92, 122).

M. Davis and H. Putnam (1960). "A computing procedure for quantification theory". In: *Journal of the ACM (JACM)* 7(3), pp. 201–215 (cit. on pp. 74, 122).

J. De Kleer, A. Mackworth, and R. Reiter (1992). "Characterizing diagnoses and systems". In: *Artificial Intelligence* 56(2), pp. 197–222 (cit. on p. 17).

L. De Raedt (2008). *Logical and relational learning*. Springer (cit. on p. 4).

L. De Raedt and S. Džeroski (1994). "First-order *jk*-clausal theories are PAC-learnable". In: *Artificial Intelligence* 70(1), pp. 375–392 (cit. on p. 216).

L. De Raedt, A. Kimmig, and H. Toivonen (2007). "Problog: a probabilistic prolog and its application in link discovery". In: *Proceedings of the 20th international joint conference on Artifical intelligence*, pp. 2468–2473 (cit. on pp. 133, 139).

L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, eds. (2008). *Probabilistic inductive logic programming: theory and applications*. Berlin, Heidelberg: Springer-Verlag. (Cit. on pp. 4, 129).

R. de Salvo Braz, S. Natarajan, H. Bui, J. Shavlik, and S. Russell (2009). "Anytime lifted belief propagation". In: *Proceedings of the 6th International Workshop on Statistical Relational Learning* (cit. on p. 208).

R. de Salvo Braz, E. Amir, and D. Roth (2005). "Lifted first-order probabilistic inference". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1319–1325 (cit. on pp. 94, 101, 144, 159, 161, 209).

R. de Salvo Braz, E. Amir, and D. Roth (2006). "MPE and partial inversion in lifted probabilistic variable elimination". In: *Proceedings of the 21st national conference on Artificial intelligence (AAAI)*. Boston, Massachusetts, pp. 1123–1130 (cit. on pp. 124, 159).

R. Dechter (1996). "Bucket elimination: a unifying framework for probabilistic inference". In: *Proceedings of the twelfth international conference on uncertainty in artificial intelligence (UAI)*. Morgan Kaufmann Publishers Inc., pp. 211–219 (cit. on pp. 25, 161).

R. Dechter and R. Mateescu (2007). "And/or search spaces for graphical models". In: *Artificial intelligence* 171(2), pp. 73–106 (cit. on p. 122).

R. Dechter, K. Kask, and R. Mateescu (2002). "Iterative join-graph propagation". In: *Proceedings of the 18th conference on uncertainty in artificial intelligence (UAI)*, pp. 128–136 (cit. on pp. 34, 206).

R. Dechter and I. Rish (2003). "Mini-buckets: a general scheme for bounded inference". In: *Journal of the ACM (JACM)* 50(2), pp. 107–153 (cit. on p. 206).

D. Déharbe and S. Ranise (2002). *BDD-driven first-order satisfiability procedures*. Tech. rep. (cit. on p. 77).

A. Del Val (1996). "Approximate knowledge compilation: the first order case". In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 498–503 (cit. on p. 75).

A. Del Val (2005). "First order LUB approximations: characterization and algorithms". In: *Artificial Intelligence* 162(1), pp. 7–48 (cit. on p. 75).

S. Della Pietra, V. Della Pietra, and J. Lafferty (1997). "Inducing features of random fields". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19(4), pp. 380–393 (cit. on p. 35).

A. Dempster, N. Laird, and D. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38 (cit. on p. 35).

M. Denecker, M. Bruynooghe, and V. W. Marek (2001). "Logic programming revisited: logic programs as inductive definitions". In: *ACM Transactions on Computational Logic* 2(4), pp. 623–654 (cit. on p. 135).

P. Domingos and D. Lowd (2009). "Markov logic: an interface layer for artificial intelligence". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3(1), pp. 1–155 (cit. on p. 131).

P. Domingos and W. Webb (2012). "A tractable first-order probabilistic logic". In: *Proceedings of the Twenty-Sixth National Conference on Artificial Intelligence* (cit. on p. 183).

J. Eisner, E. Goldlust, and N. Smith (2005). "Compiling Comp Ling: weighted dynamic programming and the Dyna language". In: *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pp. 281–290 (cit. on p. 31).

G. Elidan and A. Globerson (2010). "Summary of the 2010 UAI approximate inference challenge". http://www.cs.huji.ac.il/project/UAI10/ (cit. on pp. 187, 206).

P. Elliott and B. Williams (2006). "DNNF-based belief state estimation". In: *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)* (cit. on p. 19).

F. Fages (1994). "Consistency of Clark's completion and existence of stable models". In: *Journal of Methods of logic in computer science* 1(1), pp. 51–60 (cit. on p. 149).

T. Fahle, S. Schamberger, and M. Sellmann (2001). "Symmetry breaking". In: *Principles and Practice of Constraint Programming*. Springer, pp. 93–107 (cit. on p. 144).

A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner (2004). "Consistency-based diagnosis of configuration knowledge bases". In: *Artificial Intelligence* 152(2), pp. 213–234 (cit. on p. 17).

D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty (2010). "Building watson: an overview of the deepqa project". In: *AI Magazine* 31(3), pp. 59–79 (cit. on p. 2).

D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon (2005). "Logical Bayesian networks and their relation to other probabilistic logical models". In: *Inductive Logic Programming*, pp. 121–135 (cit. on p. 137).

D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt (2011a). "Inference in probabilistic logic programs using weighted CNF's". In: *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 211–220 (cit. on pp. 12, 19, 139, 151, 158).

D. Fierens, G. Van den Broeck, M. Bruynooghe, and L. De Raedt (Dec. 2012a). "Constraints for probabilistic logic programming". In: *Proceedings of the NIPS Probabilistic Programming Workshop*, (cit. on pp. 135, 151).

D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt (2012b). "Inference and learning in probabilistic logic programs using weighted Boolean formulas". In: (submitted) (cit. on pp. 12, 41, 151).

R. Fikes and N. Nilsson (1972). "STRIPS: a new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2(3), pp. 189–208 (cit. on p. 4).

R. Freedman, R. de Salvo Braz, H. Bui, and S. Natarajan (2012). "Initial empirical evaluation of anytime lifted belief propagation". In: *Proceedings of StaRAI* (cit. on p. 208).

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer (1999). "Learning probabilistic relational models". In: *Proceedings of IJCAI*, pp. 1300–1309 (cit. on pp. 41, 137).

M. Gelfond and V. Lifschitz (1988). "The stable model semantics for logic programming". In: *Proceedings of the 5th International Conference on Logic programming*. Vol. 161 (cit. on p. 21).

I. Gent and B. Smith (2000). "Symmetry breaking during search in constraint programming". In: *Proceedings ECAI*. Vol. 2000. Citeseer (cit. on p. 144).

L. Getoor and B. Taskar, eds. (2007). *An Introduction to Statistical Relational Learning*. MIT Press (cit. on pp. 4, 129).

J. Gillis and J. Van den Bussche (2012). "Expressive power of safe first-order logical decision trees". In: *Inductive Logic Programming*, pp. 160–172 (cit. on p. 76).

V. Gogate and R. Dechter (2011). "Samplesearch: importance sampling in presence of determinism". In: *Artificial Intelligence* 175(2), pp. 694–729 (cit. on p. 28).

V. Gogate and P. Domingos (2010). "Exploiting logical structure in lifted probabilistic inference". In: *Proceedings of StaRAI* (cit. on pp. 122, 159, 160).

V. Gogate and P. Domingos (2011). "Probabilistic theorem proving". In: *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 256–265 (cit. on pp. 75, 122, 124, 158–160, 207).

V. Gogate, A. Jha, and D. Venugopal (2012). "Advances in lifted importance sampling". In: *Proceedings of the AAAI Conference on Artificial Intelligence* (cit. on p. 207).

G. Gogic, H. Kautz, C. Papadimitriou, and B. Selman (1995). "The comparative linguistics of knowledge representation". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. Vol. 14, pp. 862–869 (cit. on p. 175).

C. P. Gomes, A. Sabharwal, and B. Selman (2009). "Model Counting". In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. Chap. 20 (cit. on p. 41).

N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum (2008). "Church: a language for generative models". In: *Proceedings of UAI*, pp. 220–229 (cit. on p. 137).

J. Goubault (1995). "A BDD-based simplification and Skolemization procedure". In: *Logic Journal of IGPL* 3(6), pp. 827–855 (cit. on pp. 77, 78).

J. F. Groote and O. Tveretina (2003). "Binary decision diagrams for first-order predicate logic". In: *Journal of Logic and Algebraic Programming* 57(1-2), pp. 1–22 (cit. on p. 78).

P. Haddawy (1994). "Generating Bayesian networks from probability logic knowledge bases". In: *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 262–269 (cit. on p. 137).

F. Hadiji, B. Ahmadi, and K. Kersting (2011). "Efficient sequential clamping for lifted message passing". In: *KI 2011: Advances in Artificial Intelligence*, pp. 122–133 (cit. on p. 208).

J. Halpern (1990). "An analysis of first-order logics of probability". In: *Artificial Intelligence* 46(3), pp. 311–350 (cit. on p. 135).

T. Hinrichs and M. Genesereth (2006). *Herbrand Logic*. Tech. rep. LG-2006-02. http://logic.stanford.edu/reports/LG-2006-02.pdf. Stanford, CA: Stanford University (cit. on p. 46).

J. Huang and A. Darwiche (2005). "DPLL with a trace: from SAT to knowledge compilation". In: *Proceedings of the International Joint Conference On Artificial Intelligence*. Vol. 19, p. 156 (cit. on pp. 74, 75, 122).

U. Hustadt, R. Schmidt, and L. Georgieva (2004). "A survey of decidable first-order fragments and description logics". In: *Journal of Relational Methods in Computer Science* 1, pp. 251–276 (cit. on p. 184).

T. N. Huynh and R. J. Mooney (2009). "Max-margin weight learning for Markov logic networks". In: *European Conference pm Machine Learning and Knowledge Discovery in Databases*, pp. 564–579 (cit. on p. 216).

M. Jaeger (1997). "Relational Bayesian networks". In: *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence (UAI)*. Morgan Kaufmann Publishers Inc., pp. 266–273 (cit. on p. 137).

M. Jaeger (2000). "On the complexity of inference about probabilistic relational models". In: *Artificial Intelligence* 117, pp. 297–308 (cit. on pp. 180, 181).

M. Jaeger (2012). "Lower complexity bounds for lifted inference". In: *CoRR* abs/1204.3255 (cit. on p. 180).

M. Jaeger and G. Van den Broeck (2012). "Liftability of probabilistic inference: Upper and lower bounds". In: *Proceedings of the 2nd International Workshop on Statistical Relational AI,* (cit. on pp. 11, 170, 180).

A. Jaimovich, O. Meshi, and N. Friedman (2007). "Template based inference in symmetric relational Markov random fields". In: *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on pp. 158, 206, 218).

T. Janhunen (2004). "Representing normal programs with clauses". In: *ECAI*. Vol. 16, p. 358 (cit. on p. 151).

M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon (2012). "The international SAT solver competitions". In: *AI Magazine* 33(1), p. 89 (cit. on p. 41).

R. Jensen (1968). "On the consistency of a slight (?) modification of Quine's new foundations". In: *Synthese* 19(1), pp. 250–263 (cit. on p. 47).

A. Jha, V. Gogate, A. Meliou, and D. Suciu (2010). "Lifted inference seen from the other side: the tractable features". In: *Proceedings of the 24th Conference on Neural Information Processing Systems (NIPS)* (cit. on pp. 105, 124, 159).

D. Johnson (1990). "A catalog of complexity classes". In: *Handbook of theoretical computer science* 1, pp. 67–161 (cit. on p. 181).

N. D. Jones and A. L. Selman (1972). "Turing machines and the spectra of first-order formulas with equality". In: *Proceedings of the Fourth ACM Symposium on Theory of Computing*, pp. 157–167 (cit. on p. 181).

M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul (1999). "An introduction to variational methods for graphical models". In: *Machine learning* 37(2), pp. 183–233 (cit. on p. 30).

S. Joshi and R. Khardon (2011). "Probabilistic relational planning with first order decision diagrams". In: *Journal of Artificial Intelligence Research* 41(2), pp. 231–266 (cit. on p. 78).

K. Kask and R. Dechter (2001). "A general scheme for automatic generation of search heuristics from specification dependencies". In: *Artificial Intelligence* 129(1-2), pp. 91–131 (cit. on p. 206).

H. Kautz, B. Selman, et al. (1992). "Planning as satisfiability". In: *Proceedings of the 10th European conference on Artificial intelligence*, pp. 359–363 (cit. on p. 17).

K. Kersting (2012). "Lifted probabilistic inference". In: *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, pp. 33–38 (cit. on p. 158).

K. Kersting, B. Ahmadi, and S. Natarajan (2009). "Counting belief propagation". In: *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press, pp. 277–284 (cit. on pp. 158, 206, 211, 218).

K. Kersting and L. De Raedt (2001a). "Bayesian logic programs". In: *arXiv preprint cs/0111058* (cit. on p. 137).

K. Kersting and L. De Raedt (2001b). "Towards combining inductive logic programming with Bayesian networks". In: *Inductive Logic Programming*, pp. 118–131 (cit. on p. 41).

K. Kersting, Y. El Massaoudi, B. Ahmadi, and F. Hadiji (2010). "Informed lifting for message-passing". In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, (cit. on p. 208).

A. Kimmig, G. Van den Broeck, and L. De Raedt (2011). "An algebraic Prolog for reasoning about possible worlds". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pp. 209–214 (cit. on pp. 12, 31).

A. Kimmig, G. Van den Broeck, and L. De Raedt (Nov. 2012a). "Algebraic Model Counting". In: arXiv:1211.4475. (Cit. on pp. 12, 31).

J. Kisyński and D. Poole (2009a). "Constraint processing in lifted probabilistic inference". In: *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press, pp. 293–302 (cit. on pp. 103, 132, 144).

J. Kisyński and D. Poole (2009b). "Lifted aggregation in directed first-order probabilistic models". In: *Proc. IJCAI*. Vol. 9, pp. 1922–1929 (cit. on p. 159).

S. Kok and P. Domingos (2005). "Learning the structure of Markov logic networks". In: *Proceedings of the International Conference on Machine Learning*, pp. 441–448 (cit. on pp. 216, 227).

S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos (2008). *The Alchemy System for Statistical Relational AI*. Tech. rep. `http://alchemy.cs.washington.edu`. Seattle, WA: Department of Computer Science and Engineering, University of Washington (cit. on p. 226).

D. Koller and N. Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press (cit. on pp. 21, 32, 35, 36).

D. Kroening (2009). "Software Verification". In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. Chap. 16 (cit. on p. 17).

F. Kschischang, B. Frey, and H. Loeliger (2001). "Factor graphs and the sum-product algorithm". In: *Information Theory, IEEE Transactions on* 47(2), pp. 498–519 (cit. on p. 24).

K. Kunen (1980). *Set theory*. Elsevier (cit. on p. 46).

N. Kushmerick, S. Hanks, and D. Weld (1995). "An algorithm for probabilistic planning". In: *Artificial Intelligence* 76(1), pp. 239–286 (cit. on p. 4).

L. Liberti (2012). "Symmetry in mathematical programming". In: *Mixed Integer Nonlinear Programming*, pp. 263–283 (cit. on p. 144).

F. Lin and Y. Zhao (2004). "ASSAT: computing answer sets of a logic program by SAT solvers". In: *Artificial Intelligence* 157(1), pp. 115–137 (cit. on p. 17).

D. C. Liu and J. Nocedal (1989). "On the Limited Memory BFGS Method for Large Scale Optimization". In: *Mathematical Programming* 45(3), pp. 503–528 (cit. on pp. 224, 225).

D. Lowd and P. Domingos (2007). "Efficient weight learning for Markov logic networks". In: *Proceedings of PKDD*, pp. 200–211 (cit. on pp. 216, 217, 226).

F. Margot (2010). "Symmetry in integer linear programming". In: *50 Years of Integer Programming 1958-2008*, pp. 647–686 (cit. on p. 144).

P. Marquis (1995). "Knowledge compilation using theory prime implicates". In: *Proceedings of the International Joint Conference On Artificial Intelligence*. Vol. 14, pp. 837–845 (cit. on p. 17).

A. McCallum, K. Schultz, and S. Singh (2009). "Factorie: probabilistic programming via imperatively defined factor graphs". In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 137, 226).

W. Meert, N. Taghipour, and H. Blockeel (2010). "First-order Bayes-ball". In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010*, pp. 369–384. (Cit. on p. 138).

C. Meinel and T. Theobald (1998). *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Verlag (cit. on pp. 17, 19).

E. Mendelson (1997). *Introduction to mathematical logic*. Chapman & Hall/CRC (cit. on p. 47).

P. Meseguer, F. Rossi, and T. Schiex (2006). "Soft Constraints". In: *Handbook of constraint programming*. Elsevier Science, pp. 281–328 (cit. on p. 31).

L. Mihalkova and R. J. Mooney (2007). "Bottom-up learning of Markov logic network structure". In: *Proceedings of the 24th International Conference on Machine Learning*, pp. 625–632 (cit. on pp. 216, 227).

B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov (2007). "BLOG: probabilistic models with unknown objects". In: *Introduction to statistical relational learning*, p. 373 (cit. on p. 137).

B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Kaelbling (2008). "Lifted probabilistic inference with counting formulas". In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 1062–1068 (cit. on pp. 132, 133, 159, 161, 209).

M. Mladenov, B. Ahmadi, and K. Kersting (2012). "Lifted linear programming". In: *15th International Conference on Artificial Intelligence and Statistics (AISTATS 2012)*. Vol. 22 (cit. on p. 144).

R. Moore (1985). "Semantical considerations on nonmonotonic logic". In: *Artificial intelligence* 25(1), pp. 75–94 (cit. on p. 21).

S. Muggleton (1996). "Stochastic logic programs". In: *Advances in inductive logic programming* 32, pp. 254–264 (cit. on p. 133).

S. Muggleton and L. De Raedt (1994). "Inductive logic programming: theory and methods". In: *The Journal of Logic Programming* 19, pp. 629–679 (cit. on pp. 4, 129).

C. Muise, S. McIlraith, J. Beck, and E. Hsu (2010). "Fast d-DNNF compilation with sharpSAT". In: *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence* (cit. on pp. 121, 125).

C. Muise, S. McIlraith, J. Beck, and E. Hsu (2012). "D sharp: fast d-DNNF compilation with sharpSAT". In: *Advances in Artificial Intelligence*, pp. 356–361 (cit. on p. 121).

K. Murphy, Y. Weiss, and M. Jordan (1999). "Loopy belief propagation for approximate inference: an empirical study". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 467–475 (cit. on p. 32).

A. Nath and P. Domingos (2009). "A language for relational decision theory". In: *Proceedings of the International Workshop on Statistical Relational Learning* (cit. on p. 136).

A. Nath and P. Domingos (2010). "Efficient lifting for online probabilistic inference". In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence* (cit. on p. 208).

M. Niepert (2012a). "Lifted probabilistic inference: an MCMC perspective". In: *Proceedings of the Second International Workshop on Statistical Relational AI (StaRAI)* (cit. on p. 207).

M. Niepert (2012b). Personal communication (cit. on p. 152).

M. Niepert (2012c). "Markov chains on orbits of permutation groups". In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on p. 207).

N. Nilsson (1986). "Probabilistic logic". In: *Artificial intelligence* 28(1), pp. 71–87 (cit. on pp. 21, 135).

H. Palacios, B. Bonet, A. Darwiche, and H. Geffner (2005). "Pruning conformant plans by counting models on compiled d-DNNF representations". In: *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pp. 141–150 (cit. on p. 19).

M. Paskin (2002). *Maximum entropy probabilistic logic*. Tech. rep. UCB/CSD-01-1161. Computer Science Division, University of California, Berkeley (cit. on p. 135).

J. Pearl (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann (cit. on pp. 21, 25, 30, 32, 192, 205).

A. Pfeffer (2001). "IBAL: a probabilistic rational programming language". In: *International Joint Conference on Artificial Intelligence*. Vol. 17. 1, pp. 733–740 (cit. on pp. 136, 137).

A. Pfeffer (2009). "Figaro: an object-oriented probabilistic programming language". In: *Charles River Analytics Technical Report* (cit. on p. 137).

A. Pfeffer, D. Koller, B. Milch, and K. Takusagawa (1999). "Spook: a system for probabilistic object-oriented knowledge representation". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 541–550 (cit. on p. 159).

A. Pfeffer (1999). "Probabilistic reasoning for complex systems". PhD thesis. Stanford University (cit. on p. 159).

K. Pipatsrisawat and A. Darwiche (2011). "On the power of clause-learning SAT solvers as resolution engines". In: *Artificial Intelligence* 175(2), pp. 512–525 (cit. on p. 74).

D. Poole (1993). "Probabilistic horn abduction and Bayesian networks". In: *Artificial Intelligence* 64, pp. 81–129 (cit. on p. 133).

D. Poole (1997). "The independent choice logic for modelling multiple agents under uncertainty". In: *Artificial Intelligence* 94(1), pp. 7–56 (cit. on pp. 4, 133, 136).

D. Poole (2003). "First-order probabilistic inference." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 985–991 (cit. on pp. 5, 88, 94, 101, 131, 140, 159, 173).

D. Poole, F. Bacchus, and J. Kisyński (2011). "Towards completely lifted search-based probabilistic inference". In: *CoRR* abs/1107.4035 (cit. on pp. 94, 146, 159, 160).

H. Poon and P. Domingos (2006). "Sound and efficient inference with probabilistic and deterministic dependencies". In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 21. 1, p. 458 (cit. on pp. 151, 217).

J. Posegga (1993). *Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe*. Sankt Augustin: Infix Verlag (cit. on p. 77).

M. Puterman (1994). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc. (cit. on p. 4).

R. Reiter (1980). "A logic for default reasoning". In: *Artificial intelligence* 13(1), pp. 81–132 (cit. on p. 21).

M. Richardson and P. Domingos (2006). "Markov logic networks". In: *Machine learning* 62(1), pp. 107–136 (cit. on pp. 41, 129, 216, 217).

J. Rintanen (2009). "Planning and SAT". In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. Chap. 15 (cit. on p. 17).

N. Robertson and P. Seymour (1983). "Graph minors. I. Excluding a forest". In: *Journal of Combinatorial Theory, Series B* 35(1), pp. 39–61 (cit. on pp. 30, 222).

N. Robertson and P. Seymour (1986). "Graph minors. II. Algorithmic aspects of tree-width". In: *Journal of algorithms* 7(3), pp. 309–322 (cit. on pp. 30, 222).

J. Robinson (1965). "A machine-oriented logic based on the resolution principle". In: *Journal of the ACM (JACM)* 12(1), pp. 23–41 (cit. on pp. 73, 75, 85, 90, 142, 143).

G. Rota (1964). "The number of partitions of a set". In: *The American Mathematical Monthly* 71(5), pp. 498–504 (cit. on p. 221).

S. J. Russell and P. Norvig (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education (cit. on p. 2).

T. Sang, P. Beame, and H. Kautz (2005). "Solving Bayesian networks by weighted model counting". In: *Proceedings of the Twentieth National Conference on Artificial Intelligence*. Vol. 1, pp. 475–482 (cit. on pp. 25, 26, 28, 158).

V. Santos Costa, D. Page, M. Qazi, and J. Cussens (2003). "CLP(BN): Constraint logic programming for probabilistic knowledge". In: *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 517–524 (cit. on p. 137).

T. Sato and Y. Kameya (1997). "PRISM: a language for symbolic-statistical modeling". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. Vol. 15, pp. 1330–1339 (cit. on p. 133).

T. Sato (1995). "A statistical learning method for logic programs with distribution semantics". In: *Proceedings of the 12th International Conference on Logic Programming (ICLP)*, pp. 715–729 (cit. on pp. 133, 134).

K. Schneider, R. Kumar, and T. Kropf (1993). "Hardware verification using first order BDDs". In: *Proceedings of IFIP Conference on Hardware Description Languages and their Applications CHDL* (cit. on p. 77).

R. Sebastiani and A. Tacchella (2009). "SAT Techniques for Modal and Description Logics". In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. Chap. 25 (cit. on p. 17).

B. Selman, H. Kautz, et al. (1991). "Knowledge compilation using Horn approximations". In: *Proceedings of the ninth national conference on artificial intelligence (AAAI)*, pp. 904–909 (cit. on p. 17).

B. Selman, H. Kautz, et al. (1996). "Knowledge compilation and theory approximation". In: *Journal of the ACM* 43(2), pp. 193–224 (cit. on p. 75).

P. Sen, A. Deshpande, and L. Getoor (2009). "Bisimulation-based approximate lifted inference". In: *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press, pp. 496–505 (cit. on p. 206).

R. Shachter (1986). "Evaluating influence diagrams". In: *Operations research* 34(6), pp. 871–882 (cit. on p. 136).

C. Shannon (1949). "The synthesis of two-terminal switching circuits". In: *Bell System Technical Journal* 28(1), pp. 59–98 (cit. on p. 91).

J. Shavlik and S. Natarajan (2009). "Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network". In: *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)* (cit. on p. 138).

P. Singla and P. Domingos (2005). "Discriminative training of Markov logic networks". In: *Twentieth National Conference on Artificial Intelligence*, pp. 868–873 (cit. on pp. 216, 225).

P. Singla and P. Domingos (2008). "Lifted first-order belief propagation". In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 1094–1099 (cit. on pp. 130, 158, 161, 206, 209, 211, 218).

P. Singla, A. Nath, and P. Domingos (2010). "Approximate lifted belief propagation". In: *Proceedings of StaRAI* (cit. on pp. 144, 208).

N. Taghipour and J. Davis (2012). "Generalized counting for lifted variable elimination". In: *Proceedings of the Second International Workshop on Statistical Relational AI (StaRAI)* (cit. on pp. 133, 159).

N. Taghipour, D. Fierens, G. Van den Broeck, J. Davis, and H. Blockeel (Aug. 2012b). "Lifted variable elimination: a novel operator and completeness results". In: *ArXiv e-prints*. arXiv:1208.3809 [cs.AI] (cit. on pp. 115, 183).

N. Taghipour, D. Fierens, J. Davis, and H. Blockeel (2012). "Lifted variable elimination with arbitrary constraints". In: *Proceedings of the fifteenth international conference on artificial intelligence and statistics* (cit. on pp. 132, 159).

H. Tamaki and T. Sato (1986). "OLD resolution with tabulation". In: *Third International Conference on Logic Programming*. Springer, pp. 84–98 (cit. on p. 85).

W. Thompson and E. Schumann (1987). "Interpretation of statistical evidence in criminal trials: the prosecutor's fallacy and the defense attorney's fallacy." In: *Law and Human Behavior; Law and Human Behavior* 11(3), p. 167 (cit. on p. 6).

A. Turing (1950). "Computing machinery and intelligence". In: *Mind* 59(236), pp. 433–460 (cit. on p. 1).

L. Valiant (1979). "The complexity of enumeration and reliability problems". In: *SIAM Journal on Computing* 8(3), pp. 410–421 (cit. on pp. 68, 183).

G. Van den Broeck (2011b). "On the completeness of first-order knowledge compilation for lifted probabilistic inference". In: *Advances in Neural Information Processing Systems 24 (NIPS)*, pp. 1386–1394 (cit. on pp. 11, 50, 82, 105, 111, 115, 129, 163, 170, 210).

G. Van den Broeck, A. Choi, and A. Darwiche (2012). "Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference". In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on pp. 12, 82, 188).

G. Van den Broeck and J. Davis (2012). "Conditioning in first-order knowledge compilation and lifted probabilistic inference". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, Palo Alto, California, USA (cit. on pp. 11, 40, 128).

G. Van den Broeck, W. Meert, and J. Davis (2012). *Lifted parameter learning for Markov logic*. (submitted) (cit. on pp. 12, 216).

G. Van den Broeck, I. Thon, M. van Otterlo, and L. De Raedt (2010). "DTProbLog: A decision-theoretic probabilistic Prolog". In: *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence*, Menlo Park, California, pp. 1217–1222 (cit. on pp. 12, 136).

G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011a). "Lifted probabilistic inference by first-order knowledge compilation". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. Menlo Park, California, pp. 2178–2185 (cit. on pp. 11, 40, 49, 50, 82, 92, 107, 128, 161, 209).

A. Van Gelder, K. Ross, and J. Schlipf (1991). "The well-founded semantics for general logic programs". In: *Journal of the ACM (JACM)* 38(3), pp. 619–649 (cit. on pp. 149, 151).

J. Vennekens, M. Denecker, and M. Bruynooghe (2009). "CP-logic: a language of causal probabilistic events and its relation to logic programming". In: *Theory and Practice of Logic Programming* 9(3), pp. 245–308 (cit. on p. 133).

J. Vennekens, S. Verbaeten, and M. Bruynooghe (2004). "Logic programs with annotated disjunctions". In: *Logic Programming*, pp. 95–119 (cit. on p. 133).

D. Venugopal and V. Gogate (2012). "On lifting the Gibbs sampling algorithm". In: (cit. on p. 207).

M. Wachter and R. Haenni (2006). "Propositional DAGs: a new graph-based language for representing Boolean functions". In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning* (cit. on p. 77).

C. Wang, S. Joshi, and R. Khardon (2008). "First order decision diagrams for relational MDPs". In: *Journal of Artificial Intelligence Research* 31(1), pp. 431–472 (cit. on p. 78).

W. Wei and B. Selman (2005). "A new approach to model counting". In: *Theory and Applications of Satisfiability Testing*. Springer, pp. 96–97 (cit. on p. 28).

M. Wellman, J. Breese, and R. Goldman (1992). "From knowledge bases to decision models". In: *The Knowledge Engineering Review* 7(01), pp. 35–53 (cit. on p. 41).

J. S. Yedidia, W. T. Freeman, and Y. Weiss (2003). "Understanding belief propagation and its generalizations". In: *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann. Chap. 8, pp. 239–269 (cit. on pp. 34, 205, 206).

H. Younes, M. Littman, D. Weissman, and J. Asmuth (2005). "The first probabilistic track of the international planning competition". In: *Journal of Artificial Intelligence Research* 24(1), pp. 851–887 (cit. on p. 4).

L. Zhang, C. Madigan, M. Moskewicz, and S. Malik (2001). "Efficient conflict driven learning in a boolean satisfiability solver". In: *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*. IEEE Press, pp. 279–285 (cit. on p. 125).

N. Zhang and D. Poole (1994). "A simple approach to Bayesian network computations". In: *Proceedings of the 10th Canadian conference on artificial intelligence* (cit. on pp. 25, 161, 173).

N. Zhang and D. Poole (1999). "On the role of context-specific independence in probabilistic inference". In: *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*. Morgan Kaufmann Publishers Inc., pp. 1288–1293 (cit. on p. 159).

List of Publications

Journal Articles

D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt (2012b). "Inference and learning in probabilistic logic programs using weighted Boolean formulas". In: (submitted) (cit. on pp. 12, 41, 151).

A. Kimmig, G. Van den Broeck, and L. De Raedt (2012b). "Algebraic Model Counting". In: (submitted).

J. Renkens, G. Van den Broeck, and S. Nijssen (July 2012). "K-optimal: A novel approximate inference algorithm for ProbLog". In: *Machine Learning* 89(3), pp. 215–231.

Winner of the "Turing Theory Prize" (best student paper award) at the 21st International Conference on Inductive Logic Programming (ILP 2011), funded by Machine Learning Journal.

Highly Selective Conferences

D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt (2011a). "Inference in probabilistic logic programs using weighted CNF's". In: *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 211–220 (cit. on pp. 12, 19, 139, 151, 158).

A. Kimmig, G. Van den Broeck, and L. De Raedt (2011). "An algebraic Prolog for reasoning about possible worlds". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pp. 209–214 (cit. on pp. 12, 31).

N. Taghipour, D. Fierens, G. Van den Broeck, J. Davis, and H. Blockeel (2012a). "Completeness results for lifted variable elimination". In: (submitted).

G. Van den Broeck (2011b). "On the completeness of first-order knowledge compilation for lifted probabilistic inference". In: *Advances in Neural Information Processing Systems 24 (NIPS)*, pp. 1386–1394 (cit. on pp. 11, 50, 82, 105, 111, 115, 129, 163, 170, 210).

G. Van den Broeck, A. Choi, and A. Darwiche (2012). "Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference". In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on pp. 12, 82, 188).

G. Van den Broeck and J. Davis (2012). "Conditioning in first-order knowledge compilation and lifted probabilistic inference". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, Palo Alto, California, USA (cit. on pp. 11, 40, 128).

G. Van den Broeck, K. Driessens, and J. Ramon (2009a). "Monte-Carlo tree search in poker using expected reward distributions". In: *Proceedings of the 1st Asian Conference on Machine Learning (ACML), Lecture Notes in Computer Science*, pp. 367–381.

G. Van den Broeck, W. Meert, and J. Davis (2012). *Lifted parameter learning for Markov logic*. (submitted) (cit. on pp. 12, 216).

G. Van den Broeck, I. Thon, M. van Otterlo, and L. De Raedt (2010). "DTProbLog: A decision-theoretic probabilistic Prolog". In: *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence*, Menlo Park, California, pp. 1217–1222 (cit. on pp. 12, 136).

G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011a). "Lifted probabilistic inference by first-order knowledge compilation". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial*

Intelligence (IJCAI). Menlo Park, California, pp. 2178–2185 (cit. on pp. 11, 40, 49, 50, 82, 92, 107, 128, 161, 209).

Other Conferences

D. Nitti, G. Van den Broeck, and L. De Raedt (2011). "Probabilistic logic in dynamic domains: Particle filter with distributional clauses". In: *Preliminary Papers ILP 2011,* pp. 1–6.

J. Renkens, G. Van den Broeck, and S. Nijssen (2011). "K-Optimal: A novel approximate inference algorithm for ProbLog". In: *Preliminary Papers ILP 2011,* pp. 1–6.

J. Van Haaren and G. Van den Broeck (2011). "Relational learning for football-related predictions". In: *Preliminary Papers ILP 2011,* pp. 1–6.

Workshop Papers

M. Bruynooghe, B. De Cat, J. Drijkoningen, D. Fierens, J. Goos, B. Gutmann, A. Kimmig, W. Labeeuw, S. Langenaken, N. Landwehr, W. Meert, E. Nuyts, R. Pellegrims, R. Rymenants, S. Segers, I. Thon, J. Van Eyck, G. Van den Broeck, T. Vangansewinkel, L. Van Hove, J. Vennekens, T. Weytjens, and L. De Raedt (2009). "An exercise with statistical relational learning systems". In: *International Workshop on Statistical Relational Learning, Leuven, Belgium, 2-4 July 2009*.

D. Fierens, G. Van den Broeck, M. Bruynooghe, and L. De Raedt (Dec. 2012a). "Constraints for probabilistic logic programming". In: *Proceedings of the NIPS Probabilistic Programming Workshop,* (cit. on pp. 135, 151).

M. Jaeger and G. Van den Broeck (2012). "Liftability of probabilistic inference: Upper and lower bounds". In: *Proceedings of the 2nd International Workshop on Statistical Relational AI,* (cit. on pp. 11, 170, 180).

W. Meert, G. Van den Broeck, N. Taghipour, D. Fierens, H. Blockeel, J. Davis, and L. De Raedt (Dec. 2012). "Lifted inference for probabilistic programming". In: *Proceedings of the NIPS Probabilistic Programming Workshop,*

J. Renkens, D. Shterionov, G. Van den Broeck, J. Vlasselaer, D. Fierens, W. Meert, G. Janssens, and L. De Raedt (Dec. 2012). "ProbLog2: From probabilistic

programming to statistical relational learning". In: *Proceedings of the NIPS Probabilistic Programming Workshop*, Accepted.

I. Thon, B. Gutmann, and G. Van den Broeck (2010). "Probabilistic programming for planning problems". Statistical Relational AI workshop, Atlanta, USA, 12. July 2010.

G. Van den Broeck and K. Driessens (2011). "Automatic discretization of actions and states in Monte-Carlo tree search". In: *Proceedings of the ECML/PKDD 2011 Workshop on Machine Learning and Data Mining in and around Games*, pp. 1–12.

Technical Reports

D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt (2011b). *Inference in probabilistic logic programs using weighted CNF's*. CW Reports CW607. Leuven, Belgium: Department of Computer Science, K.U.Leuven.

A. Kimmig, G. Van den Broeck, and L. De Raedt (Nov. 2012a). "Algebraic Model Counting". In: arXiv:1211.4475. (Cit. on pp. 12, 31).

N. Taghipour, D. Fierens, G. Van den Broeck, J. Davis, and H. Blockeel (Aug. 2012b). "Lifted variable elimination: a novel operator and completeness results". In: *ArXiv e-prints*. arXiv:1208.3809 [cs.AI] (cit. on pp. 115, 183).

Abstracts

H. Blockeel, J. Davis, L. De Raedt, D. Fierens, W. Meert, N. Taghipour, and G. Van den Broeck (2012). "Recent advances in lifted inference at Leuven". Spring Workshop on Mining and Learning, Bad Neuenahr, Germany, 18-20 April 2012.

A. Kimmig, B. Gutmann, T. Mantadelis, G. Van den Broeck, V. Santos Costa, G. Janssens, and L. De Raedt (2011). *ProbLog*. ALP Newsletter.

G. Van den Broeck (2011a). "Monte-Carlo tree search for multi-player, no-limit Texas hold'em poker". SIKS Symposium on Strategic Decision-Making in Complex Games, Maastricht, the Netherlands, 15 June 2011.

G. Van den Broeck (2011c). *Probabilistic programming in Scala*. BeScala Meet-up, 13 September 2011.

G. Van den Broeck, K. Driessens, and J. Ramon (2009b). "Monte-Carlo tree search in poker using expected reward distributions". Benelux Conference on Artificial Intelligence (BNAIC), Eindhoven, the Netherlands, 29-30 October 2009.

G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011b). "Lifted probabilistic inference by first-order knowledge compilation". IJCAI Tutorial on Lifted Inference in Probabilistic Logical Models, Barcelona, Spain, 18 July 2011.

J. Van Haaren and G. Van den Broeck (2012). "Relational learning for football-related predictions". BeneLearn, Gent, Belgium, 24-25 May 2012.

Curriculum Vitae

Guy Van den Broeck was born in Leuven, Belgium on June 5th 1986. He went to school at the Sint-Jozefcollege in Turnhout and started higher education in 2004, at the KU Leuven. In 2007, he obtained a Bachelor of Science degree in engineering, specialized in computer science and electrical engineering. In 2009, he finished his Master studies at the same university, now specialized in computer science, and graduated summa cum laude with congratulations of the Board of Examiners. His Master thesis, titled "Algorithms and assessment in no-limit computer poker", was awarded the Alcatel-Lucent Innovation Award.

Supported by a four-year PhD fellowship (Aspirant) from the Research Foundation-Flanders (FWO-Vlaanderen), he began doctoral studies in September 2009, under the auspices of Prof. Luc De Raedt at the DTAI lab (Declarative Languages and Artificial Intelligence) of the KU Leuven. From February until June 2012, he was a visiting student at the University of California, Los Angeles, in the Automated Reasoning lab of Prof. Adnan Darwiche. In January 2013 he will defend his doctoral thesis, titled "Lifted inference and learning in statistical relational models".

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE
DECLARATIVE LANGUAGES AND ARTIFICIAL INTELLIGENCE
Celestijnenlaan 200A box 2402
B-3001 Heverlee
guy.vandenbroeck@cs.kuleuven.be
http://people.cs.kuleuven.be/~guy.vandenbroeck/