KATHOLIEKE UNIVERSITEIT
LEUVEN

**Arenberg Doctoral School of Science, Engineering & Technology**
Faculty of Engineering
Department of Computer Science

# Inference and Learning for Directed Probabilistic Logic Models

Wannes MEERT

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

March 2011

# Inference and Learning for Directed Probabilistic Logic Models

**Wannes MEERT**

Jury:
Prof. Dr. ir. J. Berlamont, chair
Prof. Dr. ir. H. Blockeel, promotor
Prof. Dr. ir. M. Bruynooghe
Prof. Dr. ir. H. Bruyninckx
Prof. Dr. P. Lucas
(Radboud University Nijmegen)
Dr. T. Gärtner
(University of Bonn/Fraunhofer IAIS)
Dr. ir. K. Driessens
(Maastricht University)

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

March 2011

© Katholieke Universiteit Leuven – Faculty of Engineering
Kasteelpark Arenberg 1, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2011/7515/25
ISBN 978-94-6018-323-2

# Preface

“We rarely recognize how wonderful it is that a person can traverse an entire lifetime without making a single really serious mistake – like putting a fork in one’s eye or using a window instead of a door.

Marvin Minsky, The Society of Mind (1988)

Having the text of this thesis finished it is time to thank some people, without whom I never would have gotten this far.

First and foremost, I would like to express my gratitude towards my promoter, Hendrik Blockeel, for taking the time to discuss ideas and problems, for writing numerous comments on my drafts, and for spotting plenty of new challenges I (c | sh)ould tackle. But I am especially thankful for giving me the freedom to also follow my own ideas, good or bad. His persistent enthusiasm was a great motivation during my PhD.

I thank all the people in my jury, Maurice Bruynooghe, Herman Bruyninckx and Kurt Driessens for carefully reading my text and for their suggestions, and Peter Lucas and Thomas Gärtner, not only for their appreciated feedback but also for their hospitality. The corrections and suggestions by the jury greatly improved the content of this thesis. Lastly, I thank Jean Berlamont for chairing the jury.

Research is a group effort and I had the pleasure to collaborate with a number of interesting colleagues. My gratitude goes towards the entire DTAI research group for creating an enjoyable environment. I specifically thank the people who made my time at the department a lot of fun by sharing an office, by writing papers together, by TA-ing, by doing coffee runs, by fighting the rock-scissors-paper machine, and by organizing events: Albrecht, Anton, Beau, Bernd, Björn, Celine, Daan, Eduardo, Gitte, Guy, Hanne, Ingo, Jesse, Joaquin, Jon, Joost, Kurt, Leander, Luc, Marc, Nick, Nima, Paolo, Raquel, Stefan, Theofrastos, Tias, Vinicius, Wim, and many more people. I thank the system administrators and secretaries for all their help. Lastly, I especially would like to thank Jan Struyf. I had the chance to not only having a promoter but also an experienced researcher pushing me forward. I remember endless discussions about one single word that was not perfect, coding revelations, and one late-night writing session that started with pizza and ended in the emergency room.

The research which led to this thesis was supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) and the K.U.Leuven.

I had the luck of starting a PhD in (almost) the same building where two of my friends from back home achieved a PhD. Thanks, Raf and Steven, for the breaks and the wildly varying discussion topics. Also, thanks to all my friends for all the moments we shared while camping, climbing, skiing, bouldering, traveling, discovering Leuven, and sitting around campfires.

Finally, I sincerely thank my parents for supporting me throughout the years and for creating an environment that was an awful lot of fun to grew up in, my grandparents for frangipanes and showing me the world of engineering, and my brother and sister for letting me win in one sport. A special word of thanks goes to Bieke. She, patiently, brings structure in my chaotic mind and our otherwise chaotic home. She also showed me how to get a PhD (Broek 2010) and she helped me to make this text more readable. Biekje, ook dank je wel! Voor alles!

Wannes Meert
Leuven, March 2011

# Abstract

We are confronted with a growing amount of available data which are not only noisy but also have an increasingly complex structure. The field of machine learning, a subfield of artificial intelligence, focuses on algorithms that deduce useful knowledge from data. Our goal is to represent knowledge using probabilistic logic models and to reason with these models in an automated and efficient manner. Such models bring the expressive power of first-order logic to probabilistic models, enabling them to capture both the relational structure and the uncertainty present in such data.

In this dissertation we focus on directed probabilistic logic models and more specifically on CP-logic. The aim of CP-logic is to model causal knowledge that explicitly incorporates dynamic concepts such as events and processes. The fundamental building block is the knowledge why events occur and what the effects of these events are. Efficient inference, however, is a bottleneck in CP-logic and in probabilistic logic models in general, affecting also the cost of learning. We have contributed two methods to improve the efficiency of inference and one method for learning.

The first method, first-order Bayes ball, extracts the minimal requisite subtheory of a CP-theory necessary to answer a particular query given evidence. Inference becomes more efficient by restricting computations to the minimal requisite subtheory. Contrary to Bayes ball for Bayesian networks, first-order Bayes ball reasons on the first-order level and it returns the requisite part as a first-order CP-theory. The advantages of working on the first-order level are twofold; first, it is more efficient to find the ground requisite network compared to current methods. Second, the resulting requisite network is first-order, permitting it to be used as input for lifted inference methods which exploit the symmetries present in probabilistic logic models to improve the efficiency of inference with several orders of magnitude. Experiments show that first-order Bayes ball improves existing lifted inference methods by reducing the size of the theory that needs to be analyzed and processed.

The second method to improve the efficiency of inference is contextual variable elimination with overlapping contexts which capitalizes on deterministic dependencies present in probabilistic logic models. Two special cases of combining deterministic and probabilistic relations are contextual and causal independencies, both commonly used structures in probabilistic models. The original contextual variable elimination technique compactly represents contextual independence by representing the probabilistic model in terms of confactors but cannot handle causal independence because of some restrictions in these confactors. We lift these restrictions and propose a new algorithm to deal with more general confactors. This allows for a more efficient encoding of confactors and a reduction of the computational cost. Experiments show that our algorithm outperforms contextual variable elimination and variable elimination on multiple problems.

Lastly, we propose SEM-CP-logic, an algorithm for learning ground CP-logic from data by leveraging Bayesian network learning techniques. To this end, certain modifications are required to parameter and structure learning for Bayesian networks. Most importantly, the refinement operator used by the search must take into account the fine granularity of CP-logic. Experiments in a controlled artificial domain show that learning CP-theories with SEM-CP-logic requires fewer training data than Bayesian network learning.

# Samenvatting

We worden overstelpt met een toenemende hoeveelheid aan beschikbare data die niet alleen ruis en fouten bevatten maar ook een steeds complexere structuur. Machine learning, een onderdeel van artificiële intelligentie, concentreert zich op algoritmes die betekenisvolle kennis afleiden uit data. Ons doel is om kennis voor te stellen met behulp van probabilistisch logische modellen en om met deze modellen te redeneren op een geautomatiseerde en efficiënte manier. Dergelijke modellen combineren de expressieve kracht van eerste-orde logica met probabilistische modellen zodat zowel de relationele structuur als de onzekerheid aanwezig in de data aangepakt wordt.

In dit proefschrift concentreren wij ons op gerichte probabilistisch logische modellen en meer specifiek op CP-logic. CP-logic heeft als doel de causale kennis te modelleren die expliciet dynamische concepten bevat zoals gebeurtenissen en processen. De fundamentele bouwsteen is de kennis over waarom gebeurtenissen plaatsvinden en wat de gevolgen van deze gebeurtenissen zijn. Efficiënte inferentie is een knelpunt in CP-logic en in probabilistisch logische modellen in het algemeen en beïnvloedt onrechtstreeks ook de kost van leren. We hebben aan twee methoden gewerkt om de efficiëntie van inferentie te verbeteren en aan één methode voor het verbeteren van leren.

De eerste methode, eerste-orde Bayes ball, leidt de minimaal vereiste subtheorie van een CP-theorie af die noodzakelijk is om vragen over kansen gegeven observaties te beantwoorden. Inferentie wordt efficiënter door berekeningen te limiteren tot de minimaal vereiste subtheorie. In tegenstelling tot Bayes ball voor Bayesiaanse netwerken, redeneert eerste-orde Bayes ball op het eerste-orde niveau en het herleidt het vereiste deel tot een eerste-orde CP-theorie. De voordelen van werken op het het eerste-orde niveau zijn tweeledig; ten eerste is het efficiënter om het basis vereiste netwerk te vinden in vergelijking met huidige methoden. Ten tweede is het resulterend vereiste netwerk van eerste orde, hetgeen toelaat om het te gebruiken als input voor gelifte inferentiemethoden die gebruik maken van symmetrieën aanwezig in probabilistisch logische modellen om de efficiëntie van inferentie met enkele grootte ordes te verbeteren. Experimenten tonen aan dat eerste-orde Bayes ball bestaande gelifte inferentiemethoden verbetert door de grootte van de theorie die geanalyseerd en verwerkt moet worden te reduceren.

De tweede methode om de efficiëntie van inferentie te verbeteren omvat contextuele variabele eliminatie met overlappende contexten dat voornamelijk rekening houdt met deterministische afhankelijkheden aanwezig in probabilistisch logische modellen. Twee speciale gevallen van gecombineerde deterministische en probabilistische relaties zijn contextuele en causale onafhankelijkheden, die beiden veel gebruikt worden in probabilistische modellen. De originele contextuele eliminatietechniek voor variabelen vertegenwoordigt compact contextuele onafhankelijkheid door het probabilistisch model voor te stellen als confactors maar het kan causale onafhankelijkheid niet aan omwille van enkele restricties in deze confactors. We heffen deze restricties op en stellen een nieuw algoritme voor om met algemenere confactors te kunnen werken. Dit laat toe om confactors efficiënter te coderen en om de computationele kost te verminderen. Experimenten tonen aan dat onze algoritmes contextuele variabele eliminatie en variabele eliminatie op meerdere problemen overtreffen.

Tenslotte stellen we SEM-CP-logic voor, een algoritme om CP-logic te leren uit data door Bayesiaanse netwerk leertechnieken toe te passen. Hiertoe zijn een aantal wijzigingen vereist aan parameter- en structuurleren voor Bayesiaanse netwerken. Het is belangrijk dat de verfijningsoperator die gebruikt wordt tijdens het zoeken rekening houdt met de fijnere granulariteit van CP-logic. Experimenten in een gecontroleerd artificieel domein tonen aan dat leren met SEM-CP-logic minder data vereist dan leren met Bayesiaanse netwerken.

# Abbreviations

| | |
|---|---|
| **AC** | Arithmetic Circuit |
| **ADD** | Algebraic Decision Diagram |
| **BB** | Bayes Ball |
| **BDD** | Binary Decision Diagram |
| **BIC** | Bayesian Information Criterion |
| **BLP** | Bayesian Logic Program |
| **BN** | Bayesian network |
| **CHR** | Constraint Handling Rules |
| **CI** | Contextual Independence |
| **CNF** | Conjunctive Normal Form |
| **CP** | Causal Probabilistic |
| **CPD** | Conditional Probability Distribution |
| **CVE** | Contextual Variable Elimination |
| **CVE-OC** | Contextual Variable Elimination with Overlapping Contexts |
| **DAG** | Directed Acyclic Graph |

| | |
|---|---|
| **EBN** | Equivalent Bayesian Network |
| **EM** | Expectation-Maximization |
| **FOBB** | First-order Bayes ball |
| **HMM** | Hidden Markov model |
| **ICI** | Independent Causal Influence |
| **ICL** | Independent Choice Logic |
| **KBMC** | Knowledge Based Model Construction |
| **LBN** | Logical Bayesian Network |
| **MEE** | Mutual-Exclusive and Exhaustive |
| **MF** | Multiplicative Factorization |
| **MLN** | Markov Logic Network |
| **MRN** | Minimal Requisite Network |
| **PBN** | Parameterized Bayesian Network |
| **PLL** | Probabilistic Logic Learning |
| **PLM** | Probabilistic Logic Model |
| **PRISM** | PRogramming In Statistical Modelling |
| **PRM** | Probabilistic Relational Models |
| **SEM** | Structural Expectation-Maximization |
| **SLP** | Stochastic Logic Programs |
| **SRL** | Statistical Relational Learning |
| **VE** | Variable Elimination |

# List of Symbols

| | |
|---|---|
| $\Diamond$ | End of example |
| $\mathbb{G}$ | Graph |
| $\perp\!\!\!\perp$ | Conditional independence |
| $\mathrm{Nd}(X)$ | Nondescendents of $X$ |
| $\mathrm{Pa}(X)$ | Parents of $X$ |
| $\Pr(\cdot \mid \cdot)$ | Conditional probability |
| $\Pr(\cdot)$ | Probability |
| $\square$ | End of proof |

# Contents

# List of Figures

# List of Algorithms

# 1 Introduction

Intelligence is the ability to see abstract relations between objects and events, the know how to express these relations, communicate them and reason about new objects and events. Artificial intelligence is the discipline of science that tries to create machines that exhibit intelligent behaviour. The subfield of artificial intelligence that concentrates on the algorithms to extract abstract relations from data, like for example observations, is machine learning. Our goal is to represent knowledge using probabilistic logic models and to reason with these models in an automated and efficient manner. Probabilistic logic models bring the expressive power of first-order logic to probabilistic models, enabling them to capture both the relational structure and the uncertainty present in the complex data we are confronted with nowadays.

## 1.1 Context

This dissertation is situated on the intersection of three fields: probability theory, logic, and machine learning. We will explain each of these fields in more detail followed by an introduction to the intersection of these fields: probabilistic logic learning.

### 1.1.1 Probability theory

Probability theory (Kolmogorov 1933; Shafer and Vladimir 2006) allows us to reason about a world that exhibits non-deterministic events. Outcomes of such events are captured in *random variables* and interactions between such random variables are expressed by the laws of probability. Non-determinism appears if we have only partial knowledge of the world. This is the case in complex systems where we cannot observe all possible events, or when external events can disturb the events we observe and thus cause noise on our observations.

**Example 1.1.** An example of an event is throwing a coin. The outcome of that event will be heads or tails. This can be modelled with a random variable we will call *coin* and can take one of two values: *heads* or *tails*. For a fair coin, we can state that the outcome of the event will be heads in half the cases and tails in the other half. $\diamondsuit$

**Example 1.2.** Another example of an event is a signal transmitted in a computer network, the outcome of this event is whether the signal arrives at its destination or not. This event can be constructed from a set of events describing the successful passing of the signal from node to node in the network. Such a combination of events can be complex, as we will see later. $\diamondsuit$

A variety of formalisms has been developed to express these random variables and the relations between them. Well known formalisms are *Bayesian networks* (Pearl 1988), *Markov random fields* (Kindermann and Snell 1980), and *stochastic grammars*. These models represent a subset of possible conditional independence relations on which you can do more efficient inference. In Chapter 2, we introduce Bayesian networks more formally.

Although these probabilistic formalisms are very robust with respect to noise, a disadvantage is their 'stiff' structure. Probabilistic relations are defined over specific random variables and cannot adapt easily to a changing world. This makes it difficult to model a world with a variable number of objects and relations between these objects. For example, modelling a computer network as in Example 1.2 is difficult because of the complex and dynamic structure of such a computer network. The relations between different components change as computers are added to the network. Thus, although these components are identical at the abstract level, e.g., every computer has parts like a CPU and a hard disk, in the probabilistic model, every computer has to be modelled explicitly. There is no way of expressing probabilistic rules that are in true for computer networks in general.

## 1.1.2 Logic

First-order logic (Hilbert and Ackermann 1950) and other relational formalisms such as logic programming (Lloyd 1987) can model complex situations and relations elegantly. It allows us to easily express knowledge about a variable number of objects and the relations between them.

**Example 1.3.** For a computer network we can state that a computer $c_1$ has a CPU $p_1$ with the relation $cpu(c_1, p_1)$. This relation is simply stating a fact, the knowledge that we have about computer $c_1$. Similarly, we can express that $c_1$ also has a second CPU $p_2$: $cpu(c_1, p_2)$. Next to these basic facts we can define abstract rules to define general knowledge about computers in general. The formula

$$\forall C\ multiprocessor(C) \leftrightarrow \exists P_a P_b\ cpu(P_a, C) \wedge cpu(P_b, C) \wedge P_a \neq P_b$$

states the abstract concept that a multi-processor computer is a computer with at least two processors. The identifiers $P_a$, $P_b$ and $C$ in the formula are logic variables representing a set of individual objects. Modelling this kind of knowledge in a probabilistic model would require an individual relation for every object representing a computer. ♢

While relational formalisms offer a general and expressive language, the disadvantage is their deterministic character. It is difficult to express non-deterministic relations in such a relational formalism where the amount of non-determinism is precisely quantified.

## 1.1.3 Machine Learning

Machine learning is concerned with developing algorithms that not only take decisions given a situation, but also learn from previous experiences to make smarter decisions in the future. We will refer to 'experiences' as *data* and the knowledge distilled from the data as the *model*. This can be expressed more formally as follows (Mitchell 1997):

**Definition 1.1** (Learning)**.** A computer program is said to *learn* from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.

### 1.1.4 Probabilistic Logic Learning

At the intersection of probability theory, logic theory and machine learning, the field of *probabilistic logic learning* (PLL) appears, also called *statistical relational learning* (SRL). The aim of probabilistic logic learning is to combine the strong characteristics from probability theory, logic and machine learning into a new whole that is stronger than its parts. Probabilistic logic learning started to gain traction in the beginning of the nineties (Haddawy 1994; Wellman, Breese, and Goldman 1992) and is by now a well known field (Getoor and Taskar 2007; Kersting and De Raedt 2007) with a lot of activity.

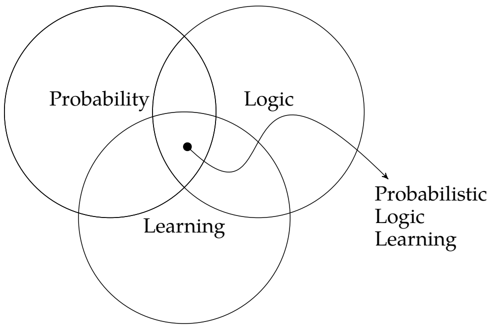


Figure 1.1: Probabilistic logic learning is the intersection of probability theory, logic theory and machine learing. (Taken from De Raedt and Kersting (2003))

**Example 1.4.** A computer network in which messages are transmitted can be elegantly modelled by means of probabilistic logic models. Such a network typically contains a variety of devices like computers, routers, and switches that are connected to each other by means of wires or wireless connections. The connections between these multiple identical components can be elegantly modelled with logic. On top of that, devices can malfunction and cables can be noisy causing messages to get lost. There is no deterministic rule to describe this, therefore, also probabilities are a necessary aspect of the model. $\Diamond$

## 1.2 Motivation, Goal and Contributions

### 1.2.1 Motivation and Goal

Learning of probabilistic logic models is an active research topic and has many promising applications (Bancilhon et al. 1986; Kok and Domingos 2009; Kersting and De Raedt 2008; Getoor and Taskar 2007). One of the key steps for learning, *inference*, however, is still a bottleneck. The expressivity we gain by combining probability theory and logic theory brings along some new challenges with respect to inference. Therefore, efficient inference is key to the development of the field of probabilistic logic learning. Concretely, our goal was *to improve the inference and learning algorithms for directed probabilistic logic models*.

### 1.2.2 Application

Mechatronics is the synergistic integration of mechanics, electronics, control theory and computer science within product design and manufacturing, in order to improve and/or optimize its functionality (AFNOR 2008). The field of mechatronics is a good candidate to apply the powerful methods of probabilistic logic learning. In this dissertation we show methods that can be applied to create and reason with a diagnosis system for a badminton playing robot. Such a badminton robot is part of a project of the Flanders Mechatronics Technology Center (FMTC) at Leuven, Belgium (see Fig. 1.2). The robot is using non-modified shuttles and rackets, which are detected and localized using purely visual information.

The robot subsystems include mechanical design, visual detection of the shuttle, shuttle trajectory estimation and interception, actuation, sensors, control hardware and software. All these subsystems are connected in a complex network in which messages are relayed from one node to another. If the robot fails, it is non-trivial to find the reason of the failure. A possible solution is to, first, model (parts of) this system as a probabilistic logic model by means of expert knowledge and machine learning, and, second, use inference to compute the most likely reason of failure given sensor observations. For a more complete and detailed overview of the robot, see Stoev et al. (2010).

For the badminton playing robot, we can use probabilistic logic models to:

- Model the software and hardware components the robot is composed of in a modular way. The interactions between software and hardware components can be elegantly expressed in a relational way.
- Reason with this model to find the failures that are most likely given some observations.
- Extract the minimal requisite model that describes a particular subsystem. This allows a design engineer to focus on the part for which he or she is responsible without the need to know details about the other subsystems.

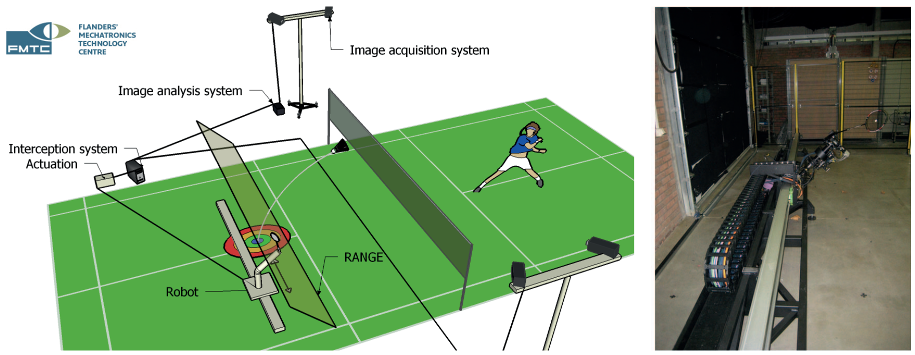


Figure 1.2: Badminton playing robot developed by the Flanders Mechatronics Technology Center (FMTC).

## 1.2.3 Contributions

This dissertation contains four main contributions.

**Extend the knowledge about CP-logic in the field of machine learning.** The probabilistic logical formalism we use is CP-logic (causal probabilistic logic), a formalism that is based on causal processes. Our work can be seen as an extension of the original CP-logic research that is situated in the field of knowledge representation towards the field of machine learning. We do this by comparing CP-logic to probabilistic formalisms like Bayesian networks and other probabilistic logic models.

**A preprocessing method improving inference for directed probabilistic logic models.** The Bayes-ball algorithm can identify the requisite variables in a propositional Bayesian network and thus ignore irrelevant variables that make inference unnecessarily more expensive. In this dissertation we present a lifted version of Bayes-ball, which works directly on the first-order level, and shows how this algorithm applies to (lifted) inference in directed first-order probabilistic models.

**Efficient inference methods for probabilistic graphical models containing local structure.** Bayesian networks extracted from probabilistic logic formalisms often include relations between the variables that cannot be captured exactly with conditional independence. In such cases, naively representing conditional probability distributions as tables and using a general purpose inference algorithm such as variable elimination results in redundant and unnecessary computations. This happens for example when the conditional probability distribution can be represented more compactly by a decision tree or a noisy-or instead of a table. Thus, the term *local structure* refers to the presence of structure in the table representing the conditional probability distribution. Contextual variable elimination partly addresses this problem by representing the Bayesian network in terms of smaller units. This leads to a more compact representation and faster inference. We propose a new approach for contextual variable elimination that lifts some of the restrictions in the original method. This seemingly simple step shows to be powerful and allows for an even more efficient encoding and a reduction of the computational cost.

**Learning directed probabilistic logic models.** We investigate the learning of CP-logic theories by leveraging Bayesian network techniques. We study the relation between CP-logic theories and Bayesian networks, and show that simple CP-logic theories can be represented with Bayesian networks consisting of noisy-or nodes, while more complex theories require close to fully connected networks, unless additional unobserved nodes are introduced in the network. We introduce an algorithm that learns CP-theories from training data and is based on a transformation between CP-theories and Bayesian networks with unobserved nodes. That is, the method applies Bayesian network learning techniques to learn a CP-logic theory in the form of an equivalent Bayesian network. To this end, certain modifications are required to the Bayesian network parameter learning and structure search, the most important one being that the refinement operator used by the search must be able to introduce the required unobserved nodes, and guarantee that the refined networks are valid CP-logic theories.

## 1.3 Structure of the Text

The remainder of this text consists of seven chapters.

- In Chapter 2 we introduce some background to the fields of probability theory and logic needed for the other chapters.
- In Chapter 3 we focus on CP-logic, a directed probabilistic logic model we have chosen to use throughout the dissertation as the probabilistic logic formalism to explain the developed methods.
- In Chapter 4 we compare CP-logic to other probabilistic logic models. Although we explain the methods developed in this dissertation by means of CP-logic, the proposed methods are in general applicable to directed logic probabilistic models. This chapter shows the connection with some other formalisms.
- In Chapter 5, we present First-order Bayes-ball, a preprocessing method to improve the efficiency of inference for directed probabilistic logic models. More specifically, it improves the applicability of lifted inference methods.
- In Chapter 6, we present the contextual variable elimination algorithm for confactors with overlapping contexts. This algorithm is an inference method that efficiently handles local structure in probabilistic graphical models.
- In Chapter 7, we focus on learning directed probabilistic logic models.
- In Chapter 8, we summarize our work, present the main conclusions, and propose some possible directions for future work.

## 1.4 Implementation

The algorithms for inference proposed in this dissertation are implemented as proof of concepts and combined in the corporal system[1]. The implementation is available at `http://dtai.cs.kuleuven.be/corporal/`. The algorithm for learning is not yet part of corporal but implemented in Python as a prototype.

[1]The abbreviation of the military rank of corporal is *cpl*, and this is also the abbreviation used for CP-logic.

# 2 Background

In this chapter we introduce concepts from the domains of probability theory and logic that will be used throughout this dissertations. In Section 2.1 we discuss the basic concepts and terminology of probability theory. This includes some well known formalisms such as Bayesian networks. In Section 2.2 we introduce propositional logic, first-order logic and logic programming as representation languages for complex models.

## 2.1 Probability Theory

In this dissertation, when we are talking about probability we use the commonly used definition as laid out by A. N. Kolmogorov who developed a set-theoretic definition of probability (Kolmogorov 1933). In the following sections, we will give an overview of the concepts needed for this dissertation. For a more detailed view on probability theory and Bayesian networks (and the proofs of the theorems) see Neapolitan (2003) or Pearl (1988).

### 2.1.1 Sample Space and Random Variables

Probability theory deals with experiments that have a set of distinct outcomes. The set combining all these outcomes is the *sample space*. Every subset of the sample space is called an *event*. A subset containing exactly one element is called an *elementary event*. Once the sample space is identified, a probability function is defined as follows:

Suppose we have a sample space $\Omega$ containing $n$ distinct elements.

$$\Omega = \{e_1, e_2, \ldots, e_n\}$$

A function that assigns a real number $\Pr(E)$ to each event $E \subseteq \Omega$ is called a *probability function* on the set of subsets of $\Omega$ if it satisfies the following conditions:

1. $0 \leq \Pr(\{e_i\}) \leq 1 \quad \text{for } 1 \leq i \leq n$
2. $\Pr(\{e_1\}) + \Pr(\{e_2\}) + \ldots + \Pr(\{e_n\}) = 1$
3. For each event $E = \{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ that is not an elementary event, $\Pr(E) = \Pr(\{e_{i_1}\}) + \Pr(\{e_{i_2}\}) + \ldots + \Pr(\{e_{i_k}\})$.

The pair $(\Omega, \Pr)$ is called a *probability space*.

Instead of saying that Pr is a probability function on the set of subsets of $\Omega$, we often shorten it to "Pr is a probability function on $\Omega$".

**Example 2.1.** Suppose you are playing a game of badminton. Your opponent launches the shuttle and you have to intercept and return it. You are not familiar with the game nor your opponent and thus have no prior knowledge to how your opponent will shoot the shuttle. We can draw an imaginary ten by four grid above the net (see Fig. 2.1) and simplify the situation to a two dimensional problem by assuming that speed is irrelevant. If you know where the shuttle appears above the net, you know where to position your racket to return the shuttle.

In this example, $\Omega$ contains all 40 possible locations where the shuttle can appear, and we consider all locations or elementary events equiprobable. We assign $\Pr(\{e\}) = 1/40$ for each $e \in \Omega$. If we let $(x, y)$ represent the event that the shuttle appears in the square on the $x^{\text{th}}$ row and the $y^{\text{th}}$ column, then

$\Pr(\{(3,1)\}) = 1/40$, and $\Pr(\{(3,1),(6,3)\}) = \Pr(\{(3,1)\}) + \Pr(\{(6,3)\}) = 1/40 + 1/40 = 1/20$. ♢

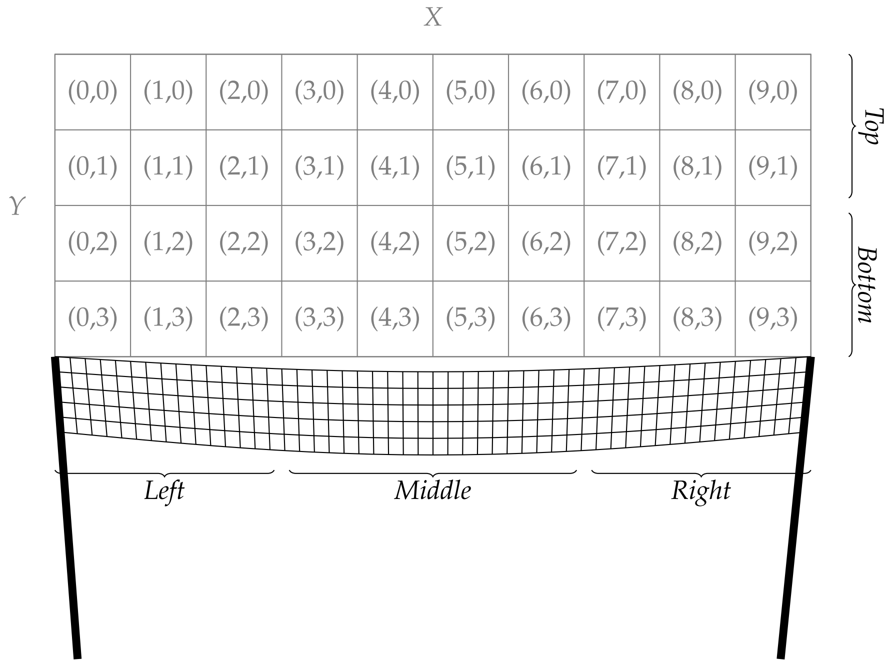


Figure 2.1: Sample space for the position of the incoming shuttle in a game of badminton.

In general it is cumbersome to talk about (elementary) events. It is more intuitive to talk about states of particular objects in the world we are considering. Therefore, a concept that is often used for representing models, is the *random variable*: Given a probability space $(\Omega, \Pr)$, a *random variable* $x$ is a function on $\Omega$ that assigns a unique value to each element in the sample space[1]. The set of values a random variable called $x$ can take is the *space* of $x$ or the *domain* of $x$ ($\text{dom}(x)$). When we say $x = x_1$, we refer to all elementary events for which $x$ maps to $x_1$ and $x_1$ is an element in the space of $x$.

[1] In this dissertation we follow the convention that a lowercase letter is used for a random variable. This is a common convention in PLL and differs from pure probability theory where capitals are used for random variables. In PLL, capitals are used for logic variables which are introduced in Section 2.2

The conditions for a sample space can be restated for random variables and are often referred to as the *axioms of probability*: Let $(\Omega, \Pr)$ be a probability space, $x$ and $y$ random variables, $x_i$ any element in the space of $x$, and $y_j$ any element in the space of $y$. Then

1. $\Pr(\Omega) = 1$
2. $0 \leq \Pr(x = x_i) \leq 1$
3. $\Pr(x = x_i \text{ or } y = y_j) = \Pr(x = x_i) + \Pr(y = y_j)$ if the set of events for which $x = x_i$ and $x = y_j$ is empty

**Example 2.2.** In the badminton example, an intuitive random variable is the column of an elementary event which we call $x$. The space of $x$ is $\{0, \ldots, 9\}$. Another random variable could be used to refer to the set of events that has as outcome that the shuttle appears in one of the first three columns. For this we can declare a random variable *left* with space $\{\mathit{false}, \mathit{true}\}$ that assigns *true* to each elementary event in the first three columns, and *false* otherwise. The following table represents some of the values of $x$ and *Left*:

| $e$ | *Left* | X | $\Pr(e)$ |
|---|---|---|---|
| (0,0) | *true* | 0 | 1/40 |
| ... | ... | ... | 1/40 |
| (1,0) | *true* | 1 | 1/40 |
| ... | ... | ... | 1/40 |
| (5,2) | *false* | 5 | 1/40 |
| ... | ... | ... | 1/40 |
| (9,3) | *false* | 9 | 1/40 |

We can now use these random variables to calculate the probability of events like "the shuttle appears in one of the left columns":

$$\Pr(\mathit{left} = \mathit{true}) = \Pr(\{(x, y) \mid 0 \leq x \leq 2, 0 \leq y \leq 9\}) = 12 \cdot 1/40 = 3/10$$

Suppose we also have a random variable *right* then we can express the probability for the event "the shuttle appears in one of the left or the right columns" as

$$\begin{aligned}\Pr(\mathit{left} = \mathit{true} \text{ or } \mathit{right} = \mathit{true}) &= \Pr(\mathit{left} = \mathit{true}) + \Pr(\mathit{right} = \mathit{true}) \\ &= \frac{3}{10} + \frac{3}{10} = \frac{3}{5}\end{aligned}$$

because the set of elementary events where $\mathit{left} = \mathit{true}$ and $\mathit{right} = \mathit{true}$ is empty. If we have a random variable *Top* that is true if the shuttle appears in the top 2 rows than the set of events that have $\mathit{left} = \mathit{true}$ and the set where $\mathit{top} = \mathit{true}$ is not empty and therefore their probabilities are not additive. However, we can combine the axioms of probability to state (for a proof, see Neapolitan (2003))

$$\Pr(x = x_i \text{ or } y = y_j) = \Pr(x = x_i) + \Pr(y = y_j) - \Pr(x = x_i \text{ and } y = y_j)$$

Thus in this case,

$$\begin{aligned}&\Pr(\mathit{left} = \mathit{true} \text{ or } \mathit{top} = \mathit{true}) \\ &\quad = \Pr(\mathit{left} = \mathit{true}) + \Pr(\mathit{top} = \mathit{true}) - \Pr(\mathit{left} = \mathit{true}, \mathit{top} = \mathit{true}) \\ &\quad = \frac{3}{10} + \frac{1}{2} - \frac{3}{20} = \frac{13}{20}\end{aligned}$$

◊

We can say that a random variable induces a probability distribution on its space. If it is clear from the context, $\Pr(x = x_i)$ may be shortened to $\Pr(x_i)$. With $\Pr(x)$ we mean the *probability distribution of the random variable $x$ on all values $x_i$ in its space.*

**Example 2.3.** The probability distribution over *left* is

$\Pr(left) =$

| *left* | $\Pr(\mathit{left})$ |
|---|---|
| *false* | 7/10 |
| *true* | 3/10 |

◊

## 2.1.2 Joint Probability Distribution

In the previous section, it is already shown that a set of events can be described by combining random variables, e.g., $\Pr(\textit{left} = \textit{true} \text{ or } \textit{right} = \textit{true})$. If random variables are combined with conjunction instead of disjunction, $\Pr(\textit{left} = \textit{true} \text{ and } \textit{right} = \textit{true})$, it is called the *joint probability* of the random variables. Say there are two random variables $x_1$ and $x_2$, then the combination of $x_1 = x_{1,i}$ and $x_2 = x_{2,j}$ expresses those elements in sample space $\Omega$ that are mapped by $x_1$ to $x_{1,i}$ and by $x_2$ to $x_{2,j}$.

We abbreviate $\Pr(x_1 = x_{1,i} \text{ and } x_2 = x_{2,j})$ with $\Pr(x_1 = x_{1,i}, x_2 = x_{2,j})$. If it is clear from the context this can be shortened further to $\Pr(x_{1,i}, x_{2,j})$. With $\Pr(x_1, x_2)$ we express the *joint probability distribution* over all possible values $x_{1,i}$ and $x_{2,j}$. Similar to how a random variable induces a probability function on its space, a conjunction of random variables also induces a probability function on the Cartesian product of their spaces.

**Example 2.4.** The probability for *left* $=$ *true* and *top* $=$ *true* is

$$\Pr(\textit{left} = \textit{true}, \textit{top} = \textit{true}) = \Pr(\{(x, y) \mid 0 \le x \le 2, 0 \le y \le 1\}) = 3/20$$

The probability distribution over *left* and *top* is

$\Pr(\textit{left}, \textit{top}) =$

| *left* | *top* | $\Pr(\textit{left}, \textit{top})$ |
|---|---|---|
| *false* | *false* | 14/40 |
| *true* | *false* | 6/40 |
| *false* | *true* | 14/40 |
| *true* | *true* | 6/40 |

$\Diamond$

## 2.1.3 Marginal Probability Distribution

Given a joint probability distribution, the law of total probability states that the probability distribution of any one of the random variables can be obtained by summing over all values of the other variables. Given the joint probability distribution $\Pr(x_1, x_2, \ldots, x_n)$, the *marginal probability distribution* of random variable $x_1$ can be obtained with the following formula:

$$\Pr(x_1) = \sum_{x_2,\ldots,x_n} \Pr(x_1, x_2, \ldots, x_n)$$

**Example 2.5.** Given the joint probability distribution over *left* and *top*, we can deduce the marginal probability over only *left* as follows:

$$\begin{aligned}\Pr(\mathit{left} = \mathit{true}) &= \sum_{\mathit{top}} \Pr(\mathit{left} = \mathit{true}, \mathit{top}) \\ &= \Pr(\mathit{left} = \mathit{true}, \mathit{top} = \mathit{true}) + \Pr(\mathit{left} = \mathit{true}, \mathit{top} = \mathit{false}) \\ &= 6/40 + 6/40 = 3/10\end{aligned}$$

◊

### 2.1.4 Conditional Probability

Often we want to express a relation between random variables. A useful concept in probability theory to express relationships is *conditional probability*.

Let $x$ and $y$ be random variables such that $\Pr(y) \neq 0$. Then the *conditional probability* of $x$ given $y$, denoted $\Pr(x \mid y)$, is given by

$$\Pr(x \mid y) = \frac{\Pr(x, y)}{\Pr(y)}$$

Conditional probability expresses the probability that a random variable $x$ has a particular value *given* the knowledge that the value of another random variable $y$ is known. If we know the value for a random variable we say that this random variable is *observed* or that it is *evidence*.

**Example 2.6.** Let *firstCol* be a random variable that is true if the shuttle appears in the first column, *left* is true if it is in the first three columns, and *top* is true for the two top rows. Then

$$\begin{aligned}
\Pr(\mathit{firstCol} = \mathit{true}) &= \frac{1}{10} \\
\Pr(\mathit{firstCol} = \mathit{true} \mid \mathit{left} = \mathit{true}) &= \frac{\Pr(\mathit{firstCol} = \mathit{true}, \mathit{left} = \mathit{true})}{\Pr(\mathit{left} = \mathit{true})} \\
&= \frac{1/10}{3/10} = \frac{1}{3} \\
\Pr(\mathit{firstCol} = \mathit{true} \mid \mathit{top} = \mathit{true}) &= \frac{\Pr(\mathit{firstCol} = \mathit{true}, \mathit{top} = \mathit{true})}{\Pr(\mathit{top} = \mathit{true})} \\
&= \frac{1/20}{1/2} = \frac{1}{10}
\end{aligned}$$

◊

In some cases, knowing the value of a particular random variable is irrelevant to the probability that another random value has a particular value, like for $\Pr(\mathit{firstCol} \mid \mathit{top})$ in the example. We say that the two random variables are independent.

Two random variables $x$ and $y$ are independent if

$$\Pr(x \mid y) = \Pr(x)$$

or if either $\Pr(x) = 0$ or $\Pr(y) = 0$. When this is the case, we write

$$x \perp\!\!\!\perp y$$

We can extend the notion of independence by conditioning on a third random variable.

Given a probability space $(\Omega, \Pr)$, and random variables $x, y$, and $z$. The random variables $x$ and $y$ are *conditionally independent given* $z$ if,

$$\Pr(x \mid y, z) = \Pr(x \mid z)$$

or if either $\Pr(x \mid z) = 0$ or $\Pr(y \mid z) = 0$. We denote this as

$$x \perp\!\!\!\perp y \mid z$$

**Example 2.7.** The example we used up to now assumed that we were trying to predict the position of the shuttle for a specific, untrained, opponent. We can extend our sample space with a set of players such that an elementary event can be for example $(2, 3, \mathit{player}_1)$. For this sample space we still use the random variables $x$ and $y$ with their previous meaning but we also can introduce some extra random variables. The random variable *male* with sample space $\{\mathit{true}, \mathit{false}\}$ maps to *true* for a male player and to *false* for a female player. Four other boolean random variables we are going to use are *lefthanded* indicating whether the player is left handed, *tall* indicating whether the player is tall or not, *right* which is true if the player places the shuttle on the right side, and *high* which is true if the player places the shuttle on the top of our grid.

Assume that there is a larger percentage of male players left handed, that male players are in general taller than female players, and that left handed players have a tendency to direct the shuttle to the right side. When observing that somebody is male we can use the previous assumptions that there is a higher probability that he is left handed and therefore is more likely to play to the right side. The observation that our opponent is male has an influence on the probability distribution of the position of the shuttle: $\Pr(\mathit{right}) \neq \Pr(\mathit{right} \mid \mathit{male})$, therefore $\mathit{right} \not\perp\!\!\!\perp \mathit{male}$. However when we observe that somebody is left handed, the additional observation that your opponent is male or not will not influence the probability distribution of *right* anymore: $\Pr(\mathit{right} \mid \mathit{lefthanded}) = \Pr(\mathit{right} \mid \mathit{lefthanded}, \mathit{male})$ and therefore $\mathit{right} \perp\!\!\!\perp \mathit{male} \mid \mathit{lefthanded}$. ◊

An application of conditional probability is to express a joint probability distribution as a product of conditional probabilities. This expansion is called the *chain rule* for random variables. Given a set of random variables $x_1, x_2, \dots, x_{n-1}, x_n$ defined on the sample space $\Omega$, by applying the definition of conditional probability, we can show that,

$$\Pr(x_1, x_2, \dots, x_{n-1}, x_n) = \Pr(x_1 \mid x_2, \dots, x_{n-1}, x_n) \cdots \Pr(x_{n-1} \mid x_n) \cdot \Pr(x_n)$$

An important application of conditional probability is the rule of Bayes. Given two random variables $x$ and $y$, we have

$$\Pr(x \mid y) = \frac{\Pr(y \mid x) \Pr(x)}{\Pr(y)}$$

This formula offers a method to reverse the direction of a conditional probability and reason in the other direction. In the next section, we show how we can use the directionality of a conditional probability to reason about a set of conditional probabilities. The rule of Bayes is used to reason about any variable in our world while making abstraction from other variables no matter how the variables are conditionally dependent on each other.

### 2.1.5 Bayesian network

When combining the chain rule and conditional independence, an interesting observation can be made. Some conditional probabilities created by means of the chain rule can be simplified using conditional independence. This means that in such a case the joint probability distribution can be written down in a simpler form.

**Example 2.8.** Continuing with Example 2.7, we can apply the chain rule using the five random variables used in the model: *male* ($m$), *lefthanded* ($l$), *tall* ($t$), *right* ($r$), and *high* ($h$). Suppose we know that the following independencies are true:

$$\begin{aligned} &\textit{male} \perp\!\!\!\perp \textit{right} \mid \textit{lefthanded} && \textit{male} \perp\!\!\!\perp \textit{high} \mid \textit{tall}, \textit{lefthanded} \\ &\textit{lefthanded} \perp\!\!\!\perp \textit{tall} \mid \textit{male} && \textit{high} \perp\!\!\!\perp \textit{right} \mid \textit{lefthanded} \\ &\textit{right} \perp\!\!\!\perp \textit{tall} \mid \textit{lefthanded} \end{aligned}$$

We can try different orderings in how we apply the chain rule. Suppose we take the order $m, t, l, h, r$ (abbreviating the random variables to their first letter)

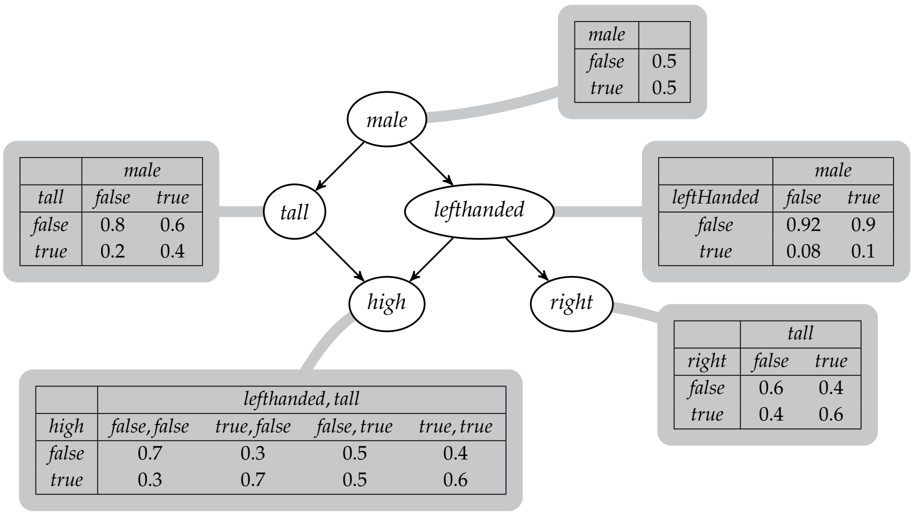
$$\begin{aligned} &\Pr(m, t, l, h, r) \\ &\quad = \Pr(m \mid t, l, h, r) \cdot \Pr(t \mid l, h, r) \cdot \Pr(l \mid h, r) \cdot \Pr(h \mid r) \cdot \Pr(r) \\ &\quad = \Pr(m \mid t, l) \cdot \Pr(t \mid l, h) \cdot \Pr(l \mid h, r) \cdot \Pr(h \mid r) \cdot \Pr(r) \end{aligned}$$

The first term of the multiplication $\Pr(m \mid t, l, h, r)$ can be simplified to $\Pr(m \mid t, l)$ since we know $m \perp\!\!\!\perp h \mid t, l$ and $m \perp\!\!\!\perp r \mid l$. A similar reasoning can be used for the second term.

Another ordering could be $r, h, l, t, m$:

$$\begin{aligned}
&\Pr(r, h, l, t, m) \\
&\quad = \Pr(r \mid h, l, t, m) \cdot \Pr(h \mid l, t, m) \cdot \Pr(l \mid t, m) \cdot \Pr(t \mid m) \cdot \Pr(m) \\
&\quad = \Pr(r \mid l) \cdot \Pr(h \mid l, t) \cdot \Pr(l \mid m) \cdot \Pr(t \mid m) \cdot \Pr(m)
\end{aligned}$$

This second ordering results in a more compact form of the chain rule. A more compact factorization has some computational advantages and makes it easier to reason about the model since it reduces the number of related random variables in a conditional probability. ◊



| *male* | |
|---|---|
| *false* | 0.5 |
| *true* | 0.5 |

| | *male* | |
|---|---|---|
| *tall* | *false* | *true* |
| *false* | 0.8 | 0.6 |
| *true* | 0.2 | 0.4 |

| | *male* | |
|---|---|---|
| *leftHanded* | *false* | *true* |
| *false* | 0.92 | 0.9 |
| *true* | 0.08 | 0.1 |

| | *tall* | |
|---|---|---|
| *right* | *false* | *true* |
| *false* | 0.6 | 0.4 |
| *true* | 0.4 | 0.6 |

| | *lefthanded, tall* | | | |
|---|---|---|---|---|
| *high* | *false, false* | *true, false* | *false, true* | *true, true* |
| *false* | 0.7 | 0.3 | 0.5 | 0.4 |
| *true* | 0.3 | 0.7 | 0.5 | 0.6 |

Figure 2.2: Bayesian network representing the probability where an opponent hits the shuttle to based on some characteristics of the player.

This expansion of the joint probability distribution into conditional probability distributions can be represented by a graph illustrating the conditional dependencies. The vertices in the graph are the random variables in the joint probability distribution, and the directed edges indicate by which random variables a random variable is conditioned in the set of conditional probability distributions created by the expansion. Summarized this is:

$$\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{Pa}(x_i))$$

with $\mathrm{Pa}(x)$ is the set of parents of $x$ with respect to the graph.

**Example 2.9.** The more compact factorization in the previous example is depicted in Fig. 2.2 by means of a graph theoretic interpretation. The conditional probability distributions represented in the factorization are shown on the sides of the graph. As a comparison, the Bayesian network of the less compact factorization is shown in Fig. 2.3. $\Diamond$

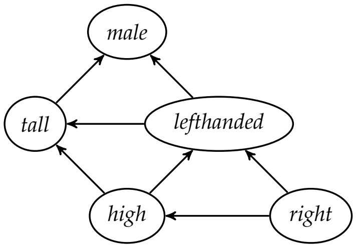


Figure 2.3: The same probability distribution represented by the Bayesian network in Fig. 2.2 but with a factorization that results in a less compact set of conditional probability distributions.

In fact, the graph representing the chain expansion is a *directed acyclic graph* (DAG) and it satisfies the *Markov condition*:

Given a joint probability distribution Pr of the random variables in some set $V$ and a DAG $\mathbb{G} = (V, E)$ with $V$ the set of nodes and $E$ the set of edges. We say that $(\mathbb{G}, \Pr)$ satisfies the *Markov condition* if for each variable $x \in V$, $\{x\}$ is conditionally independent of the set of all its nondescendants given the set of all its parents. A nondescendant is a random variable that not a descendent with respect to the graph.ical representation Let $\mathrm{Pa}(x)$ express the parents of $x$ and $\mathrm{Nd}(x)$ the nondescendants of $x$, we can state

$$x \perp\!\!\!\perp \mathrm{Nd}(x) \mid \mathrm{Pa}(x)$$

By interpreting the Markov condition, we can reason about the joint probability distribution by only looking at the graph.

### 2.1.6 Factors and variable elimination

Given a Bayesian network, a common inference task is the computation of the posterior probability distribution over a set of random variables, possibly given some evidence. Suppose we have the probability distribution $\Pr(x_1, \ldots, x_n)$ and we want to compute the posterior distribution over $x_1, \ldots, x_m$ with $1 \leq m \leq n$:

$$\Pr(x_1, \ldots, x_m) = \sum_{x_{m+1}, \ldots, x_n} \Pr(x_1, \ldots, x_n) = \sum_{x_{m+1}, \ldots, x_n} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{Pa}(x_i))$$

This type of computation is an application of the *generalized distributive law* (Aji and Mceliece 2000) and efficient algorithms for the special case of Bayesian networks are *Belief Propagation* (Pearl 1988), *Variable Elimination* (VE) (Zhang and Poole 1994; Dechter 1999; Bertelè and Brioschi 1972), and *Arithmetic Circuits* (Darwiche 2003). In Chapter 6, we focus on the VE algorithm that uses *factors* to represent the input problem instances, the results of intermediate local computations and the final solution. Factors are used because they can represent calculations with conditional probabilities in a compact and uniform way.

A *factor* on random variables $x_1, \ldots, x_n$ is a function from the Cartesian product of the domains of the random variables onto the real numbers: $\mathrm{dom}(x_1) \times \mathrm{dom}(x_2) \times \ldots \times \mathrm{dom}(x_n) \to \mathbb{R}$. A conditional probability distribution maps the possible value assignments of random variables to probabilities, which are real numbers, and can therefore be represented by a factor. In the remainder of the text can factors be considered as a general representation of different types of probabilistic interactions like conditional probability distributions and marginal probability distributions.

**Example 2.10.** Consider the Bayesian network in Figure 2.2. The probability distribution of *lefthanded* given *male* can be represented with the following factor:

$$\Pr(\mathit{lefthanded} \mid \mathit{male}) = \phi(\mathit{lefthanded}, \mathit{male}) =$$

| | *male* | |
|---|---|---|
| *lefthanded* | *false* | *true* |
| *false* | 0.8 | 0.6 |
| *true* | 0.2 | 0.4 |

◊

In the algorithms in this dissertation we make use of the following operators on factors:

- *Setting* variables to a specific value. Assigning a set of variables $\mathbf{c}$ specific values $\mathbf{c_i} \in \operatorname{dom}(\mathbf{c})$, denoted as $\mathbf{c} = \mathbf{c_i}$, results in a new factor as follows

  $$\phi_2(\mathbf{y} = \mathbf{y_i}) = \phi_1(\mathbf{y} = \mathbf{y_i}, \mathbf{c} = \mathbf{c_i})$$

  with $\mathbf{y}$ and $\mathbf{c}$ sets of variables, $\mathbf{y_i}$ any set of values in the domain of $\mathbf{y}$, and $\mathbf{c_i}$ the particular value assignment.

- The *product* of factors $\phi_1$ and $\phi_2$, written $\phi_1 \otimes \phi_2$, is a factor on the union of the variables in $\phi_1$ and $\phi_2$ and defined by

  $$(\phi_1 \otimes \phi_2)(\mathbf{x} = \mathbf{x_i}, \mathbf{y} = \mathbf{y_j}, \mathbf{z} = \mathbf{z_k})$$
  $$= \phi_1(\mathbf{x} = \mathbf{x_i}, \mathbf{y} = \mathbf{y_j}) \cdot \phi_2(\mathbf{y} = \mathbf{y_j}, \mathbf{z} = \mathbf{z_k})$$

  where $\mathbf{y}$ is the set of variables both in $\phi_1$ and $\phi_2$, $\mathbf{x}$ the variables only in $\phi_1$, $\mathbf{z}$ those only in $\phi_2$, $\mathbf{x_i} \in \operatorname{dom}(\mathbf{x})$, $\mathbf{y_j} \in \operatorname{dom}(\mathbf{y})$, and $\mathbf{z_k} \in \operatorname{dom}(\mathbf{z})$.

- The *sum* of factors $\phi_1$ and $\phi_2$, written $\phi_1 \otimes \phi_2$, is a factor on the union of the variables in $\phi_1$ and $\phi_2$ and defined by

  $$(\phi_1 \oplus \phi_2)(\mathbf{x} = \mathbf{x_i}, \mathbf{y} = \mathbf{y_j}, \mathbf{z} = \mathbf{z_k})$$
  $$= \phi_1(\mathbf{x} = \mathbf{x_i}, \mathbf{y} = \mathbf{y_j}) + \phi_2(\mathbf{y} = \mathbf{y_j}, \mathbf{z} = \mathbf{z_k})$$

  where $\mathbf{y}$ is the set of variables both in $\phi_1$ and $\phi_2$, $\mathbf{x}$ the variables only in $\phi_1$, $\mathbf{z}$ those only in $\phi_2$, $\mathbf{x_i} \in \operatorname{dom}(\mathbf{x})$, $\mathbf{y_j} \in \operatorname{dom}(\mathbf{y})$, and $\mathbf{z_k} \in \operatorname{dom}(\mathbf{z})$.

- *Summing out* a random variable $x$ from a factor $\phi$, written $\sum_x \phi$ is the factor with variables $\mathbf{y} = \{v \mid v \text{ variable in } \phi\} - \{x\}$ such that

  $$\left(\sum_x \phi\right)(\mathbf{y}) = \sum_{v_i \in \operatorname{dom}(x)} \phi(\mathbf{y} \wedge x = v_i)$$

**Example 2.11.** Using the probability distributions from the Bayesian network in Figure 2.2. Given the factor

$\phi_1(\mathit{lefthanded}, \mathit{male}) =$

| | *male* | |
|---|---|---|
| *lefthanded* | *false* | *true* |
| *false* | 0.8 | 0.6 |
| *true* | 0.2 | 0.4 |

setting *male* to *true* results in

$$\phi_1(\mathit{lefthanded}, \mathit{male} = \mathit{true}) = \begin{array}{|c|c|} \hline \mathit{lefthanded} & \mathit{true} \\ \hline \mathit{false} & 0.6 \\ \mathit{true} & 0.4 \\ \hline \end{array}$$

Multiplication of two factors $\phi_1(\mathit{lefthanded}, \mathit{male}) \otimes \phi_2(\mathit{male})$ results in a new factor $\phi_3$:

$$\begin{array}{|c|cc|} \hline & \multicolumn{2}{c|}{\mathit{male}} \\ \hline \mathit{lefthanded} & \mathit{false} & \mathit{true} \\ \hline \mathit{false} & 0.8 & 0.6 \\ \mathit{true} & 0.2 & 0.4 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline \mathit{male} & \\ \hline \mathit{false} & 0.5 \\ \mathit{true} & 0.5 \\ \hline \end{array} = \begin{array}{|c|cc|} \hline & \multicolumn{2}{c|}{\mathit{male}} \\ \hline \mathit{lefthanded} & \mathit{false} & \mathit{true} \\ \hline \mathit{false} & 0.4 & 0.3 \\ \mathit{true} & 0.1 & 0.2 \\ \hline \end{array}$$

And summing out variable *male* in factor $\phi_3$ results in:

$$\sum_{\mathit{male}} \phi_3(\mathit{lefthanded}, \mathit{male}) = \sum_{\mathit{male}} \begin{array}{|c|cc|} \hline & \multicolumn{2}{c|}{\mathit{male}} \\ \hline \mathit{lefthanded} & \mathit{false} & \mathit{true} \\ \hline \mathit{false} & 0.4 & 0.3 \\ \mathit{true} & 0.1 & 0.2 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|} \hline \mathit{lefthanded} & \\ \hline \mathit{false} & 0.7 \\ \mathit{true} & 0.3 \\ \hline \end{array}$$

$\Diamond$

### 2.1.7 Barren nodes and D-separation

One of the advantages of Bayesian networks is that much of the knowledge is captured in the graphical structure. Statements of conditional independence (or irrelevance) can be verified based on the graph. Whether sets of nodes are independent or not can be defined using *d-separation* (Pearl 1988) and its deterministic generalization *D-separation* (Geiger, Verma, and Pearl 1990). The original d-separation makes use of the concept of an *active path*. An active path from a set of nodes $\mathbf{a}$ to a set of nodes $\mathbf{b}$ given a third set of nodes $\mathbf{c}$ is an undirected path between $i \in \mathbf{a}$ and $j \in \mathbf{b}$, such that every node with two incoming edges that are on the path is or has a descendent in **c**. **c** is said to *d-separate* **a** from **b** if there is no active path from **a** to **b** given **c**. This condition determines the irrelevancies between nodes in the network. The deterministic generalization takes into account *deterministic nodes* where the value of the node is functionally defined by the values of the parents. A node is functionally defined by the values of the parents if for those values the value of the child can have only one value, like in a function $child = f(parents = values)$. For D-separation, the definition of an active path is changed such that an active path from a set of nodes **a** to a set of nodes **b** given a third set of nodes **c** is an undirected path between $i \in \mathbf{a}$ and $j \in \mathbf{b}$, such that every node with two incoming edges that are on the path is or has a descendent in **c** and every other node on the path is not functionally determined by **c**.

Another concept is that of a *barren node*. A node is barren if it is not part of the query, it has not received evidence and its children are barren. A barren node does not contribute to the probability of non-barren nodes, and therefore we do not need to take it into account when calculating the marginal probability.

## 2.2 Logic

In this section we give a short introduction to some aspects of the language of logic that we will be using. For a more detailed exposition see Dewdney (1993), Huth and Ryan (2004), and Lloyd (1987) (for a more philosophical treatment we also suggest Hofstadter (1979)).

### 2.2.1 Propositional logic

Where probability theory uses random variables, propositional logic has *boolean variables* that can have one of only two possible values, *true* or *false*. To illustrate the connection to random variables we can define a boolean variable $x$ as a function on the sample space $\Omega$ that assigns *true* or *false* to each element in $\Omega$. If it is clear from the context we shorten $x = true$ and $x = false$ to $x$ and $\neg x$.

Where there are probability functions in probability theory, there are boolean expressions in logic. A probability functions maps random variables to a real value and a boolean expression maps boolean variables to a boolean value. A set of boolean expressions together form a *theory*. If, for a particular assignment of values to the boolean variables all the boolean expression in the theory map to *true*, we say the value assignment *satisfies* the theory.

**Example 2.12.** We, again, use the shuttle positions example. For every column in the grid (see Figure 2.1) we can define a boolean variable $x_i$ expressing if the shuttle appears in column $i$. Suppose we want to create a theory that enforces that the shuttle is only played in the left half of the playing area (the grid). In that case our theory should map all value assignments indicating that the shuttle is on the left half to *true* and those on the right half to *false*. This situation is summarized in the following table:

| $\Omega$ | $x_0$ | $x_1$ | ... | $x_5$ | $x_9$ | satisfied |
|---|---|---|---|---|---|---|
| (0,0) | *true* | *false* | ... | *false* | *false* | *true* |
| ... | ... | ... | ... | ... | ... | ... |
| (1,0) | *false* | *true* | ... | *false* | *false* | *true* |
| ... | ... | ... | ... | ... | ... | ... |
| (5,2) | *false* | *false* | ... | *true* | *false* | *false* |
| ... | ... | ... | ... | ... | ... | ... |
| (9,3) | *false* | *false* | ... | *false* | *true* | *false* |

$\Diamond$

Propositional logic uses *boolean functions* that have a number of boolean input variables, and for each possible combination of input values, such a function has a boolean valued output. Some well known boolean functions are the *constant-functions true* and *false*, the *or-function* ($\vee$), the *and-function* ($\wedge$), the *implication-function* ($\rightarrow$), the *equivalence-function* ($\leftrightarrow$), and the *not-function* ($\neg$). Their definitions are given in Fig. 2.4 and are sufficient to define all possible boolean functions since all $n$-ary boolean functions can be rewritten to this set of 2-ary and 1-ary Boolean functions[2]. For example, $x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4$ can be rewritten as $(((x_0 \vee x_1) \vee x_2) \vee x_3) \vee x_4$.

We are going to use boolean variables and boolean functions to create expressions that make up a theory. A *boolean expression* is defined using a set $\Sigma$ (the alphabet) of symbols which denote the boolean variables, and *true*, and *false*.

- If $x \in \Sigma$, then $x$ is a boolean expression.

[2]The set of Boolean functions $\neg$ and $\wedge$ is actually also a sufficient set to represent all other Boolean functions and is a minimal sufficient set.

| $x$ | $y$ | $\neg x$ | $x \vee y$ | $x \wedge y$ | $x \rightarrow y$ | $x \leftrightarrow y$ |
|---|---|---|---|---|---|---|
| *false* | *false* | *true* | *false* | *false* | *true* | *true* |
| *true* | *false* | *false* | *true* | *false* | *false* | *false* |
| *false* | *true* | *true* | *true* | *false* | *true* | *false* |
| *true* | *true* | *false* | *true* | *true* | *true* | *true* |

Figure 2.4: The basic Boolean functions

- If $a$ and $b$ are boolean expressions, then so are $a \vee b$, $a \wedge b$, $a \rightarrow b$, $a \leftrightarrow b$, and $\neg a$.

Two boolean expressions that, for every value assignment of the boolean variables, evaluate to the same value are called *equivalent*. A well known example of two such expressions are $x \wedge (y \vee z)$ and $(x \wedge y) \vee (x \wedge z)$ and this equivalence is called the distributivity of $\wedge$ over $\vee$.

A theory can be defined with a boolean expression and if a set of assignments to the boolean variables is such that the expression itself is true we say that that the set of assignments *satisfies* the theory.

Finding the most compact, equivalent, logic expression for a theory is advantageous with respect to computational effort to evaluate expressions and to the intuitive meaning of a theory.

**Example 2.13.** The table above is summarized by means of an expression using boolean variables and function in a theory as follows:

$$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4$$

This theory is satisfied for events where the shuttle is in one of the first five columns, therefore the theory expresses that the shuttle has to be in at least one of the first five columns. An equivalent theory is

$$(x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge \ldots \wedge \neg x_9) \vee \ldots \vee (\neg x_0 \wedge x_1 \wedge \neg x_2 \wedge \ldots \wedge \neg x_9) \vee \ldots$$

Or in words, for every row that evaluates to *true* in the table, we add the conjunction of the values to a disjunction. This representation is an exact replication of the table in a logic expression and is clearly not as compact as the previous expression. $\Diamond$

### 2.2.2 First order logic

First order logic, also called *predicate calculus*, is a powerful language for expressing mathematical ideas and thoughts. One can express a set of axioms in first order logic and combined with the inference technique *resolution* (Robinson 1965) it is possible to derive new theorems from old theorems and axioms.

The language of first order logic can be defined inductively as was the case with propositional logic. The basic building blocks of first-order formulas are three types of individual symbols, together the *alphabet*: *logic variables* (or object symbols), *function symbols*, and *predicate symbols*. We use the convention that logic variables start with a uppercase letter and predicate symbols and function symbols with a lowercase letter.

First, we define some building blocks for the language of first order logic. A *term* is defined recursively as

1. A variable is a term.
2. If $f/n$ is a function symbol and $t_1, t_2, \ldots, t_n$ are terms, then function $f(t_1, t_2, \ldots, t_n)$ is a term.

An *atom* is of the form $p(t_1, t_2, \ldots, t_n)$, where $p/n$ is a predicate symbol and all $t_i$ are terms. A *first-order formula* is recursively defined as

1. An atom is a formula.
2. *true* and *false* are formulas.
3. If $\phi$ and $\psi$ are formulas, then so are $\phi \vee \psi$, $\phi \wedge \psi$. $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$, and $\neg\phi$.
4. If $\phi$ is formula and $V$ is a logic variable, then $\forall V\ \phi$ and $\exists V\ \phi$ are formulas.

A formula can be true or false depending on the value of the predicates. Whether a predicate is true or false is not as simple as with propositional logic and we need the concept of an interpretation. An *interpretation* $I$ is a mapping of logic variables to domain elements and maps each n-ary predicate with these domain elements as arguments to *true* or *false*. Given an interpretation, a formula can resolve to *true* or *false*. If an interpretation makes the formula true, we say that the interpretation *satisfies* the formula. A formula is called *satisfiable* if it has at least one interpretation for which it is true. It is called *valid* if it is true under all possible interpretations.

**Example 2.14.** We revisit the badminton example and express the constraint that you can only play on the left half with a first-order formula. The positions in the grid are represented by a 2-ary predicate $pos/2$ with as arguments the x and y coordinates and a 2-ary predicate $l/2$ that is true if the first argument is smaller than the second (assuming that there is an order on them). First-order logic thus allows us to express the grid in a more abstract way since we do not need to specify a boolean variable for every column. We use the 2-ary predicate $allowed/2$ to say what positions are allowed in our setting and which not. Our rule can now be expressed by the following formula:

$$\forall X, Y\ allowed(X,Y) \leftrightarrow l(X,5) \wedge pos(X,Y)$$

For the badminton example we use a domain consisting of the numbers from 0 to 9 resulting in the predicates from $pos(0,0)$ to $pos(9,5)$, the predicates from $l(0,0)$ to $l(9,9)$, and those from $allowed(0,0)$ to $allowed(9,9)$. Given is that $l/2$ maps onto *true* if the first argument is smaller than the second and *false* otherwise. Under these conditions the only interpretation that maps $allowed/2$ to *true* or *false* that satisfies the given formula is that all the left positions map to *true* and the others to *false*. And this was the interpretation we intended with the formula:

$$I = \{allowed(X,Y) \mid 0 \leq X \leq 5, 0 \leq Y \leq 3\} \cup \{l(X,Y) \mid X < Y\}$$

By convention we include only the atoms that map to *true* and all missing atoms map to *false*. $\lozenge$

We end this section with some terminology that is needed in this dissertation. A *ground* formula is a formula that contains no logic variables. A *literal* is an atom or its negation. The negation of an atom is called a negative literal and an atom without negation is a positive literal. A *clause* is a formula consisting only out of conjunction of literals. We call a variable *free* if it is not quantified as for all or there exists.

### 2.2.3 Logic programming

Logic programming uses the language of first-order logic for computer programming. It uses a subset of first-order logic where the domain is restricted to the *Herbrand universe*, i.e., the set of all ground terms that can be constructed using the constants and function symbols in the alphabet, and only *Horn clauses*, i.e., universally quantified clauses that have at most one positive literal, are allowed. Where it differs with first-order logic is that it uses a procedural interpretation for the implication operator. Best known of the logic programming languages is *Prolog* (Colmerauer and Roussel 1993).

In general a distinction is made between two types of Horn clauses. Those that have no negative literals are called *facts* and those that have negative literals are called *rules*.

**Example 2.15.** First, we express in the badminton game where the shuttle can appear if we say that it appears in the left half of the playing field. This is equivalent to saying that it has to appear in one of the first five columns and can be expressed with a Horn clause:

$$\forall X, Y \ \mathit{allowed}(X,Y) \vee \neg l(X,5) \vee \neg allX(X) \vee \neg allY(Y)$$

Notice that we have added $allX/1$ and $allY/1$ predicates. This is necessary since we are using the Herbrand universe as the domain for the logic variables and this domain is constructed with the constants and functions available in the theory. The above formula is usually written as the following rule:

$$\mathit{allowed}(X,Y) \leftarrow l(X,5), allX(X), allY(Y).$$

and can be read as follows: For all x and y, if x is smaller than 5, x is a position on the x-axis, and y is a position on the y-axis, then the position in the grid on position $(x,y)$ is allowed. We complete the program by adding some facts to the program:

$$\begin{array}{lll}
allX(0). & allY(0). & l(0,5). \\
allX(1). & allY(1). & l(1,5). \\
\quad\dots & \quad\dots & \quad\dots \\
allX(9). & allY(5). & l(4,5).
\end{array}$$

◊

The procedural interpretation for implication is enforced by using SLD-resolution (Van Emden and Kowalski 1976) as the inference method to find interpretations for a set of clauses. Important for resolution is the concept of *unification*. Two atoms can be unified if there is a substitution for the variables occurring in them that brings them into precisely the same form. For example, the atoms $p(a, f(X))$ and $p(X, Y)$ can be unified with a substitution $\{X/a, Y/f(a)\}$ resulting in the unified atom $p(a, f(a))$.

The SLD-resolution is activated by asking a question to the system, like: is position $(2, 3)$ a position in the left half? The question is written as an assertion for which the truth will be tested:

$$\neg allowed(2, 3).$$

The inference encounters the clause where $allowed(x, y)$ is the positive literal and unifies it with the predicate in the question by replacing $x$ with 2 and $y$ with 3. If such a unification is possible, the matching predicate is replaced with the negative literals in the clause. This operation is called resolution. The new question is now:

$$l(2, 3), allX(2), allY(3).$$

SLD-resolution looks at the predicates in the question from left to right and encounters now $l(2, 3)$. This predicate is the positive predicate in one of the clauses and since this clause has no negative literals it is replaced with *true* in the question. The same can be done for $allX(2)$ and $allY(3)$ resulting in a conjunction of *true* constants meaning that our question is answered positive. If SLD-resolution cannot find a clause to replace a predicate in the question, it backtracks over the clauses it selected before to find other answers. If no alternative choice for a previously selected clause is available, the system returns *false* and the question is answered negative.

## 2.3 Summary

We now summarize the main concepts introduced in this chapter and their relevance to the rest of this dissertation.

We introduced Bayesian networks as a compact representation of probability distributions over a set of random variables. In Chapter 3 we show how we can extend on this formalism by using concepts from logic. In Chapter 6 we will explain how we can make this representation even more compact and reduce the computational effort needed to do inference in a Bayesian network.

We discussed how logic can be used as an expressive language to formulate the rules governing a particular world. In Chapter 3, we use a formalism that combines logic with probability theory to offer the advantages of both fields. In Chapter 5 we show an algorithm that combines logic inference like SLD-resolution with notions from probability theory to find a minimal requisite theory necessary to answer a question.

# 3 CP-Logic

## 3.1 Introduction

There have been a plethora of language formalisms proposed in the PLL community (Getoor and Taskar 2007; De Raedt and Kersting 2003). In this dissertation we focus on directed probabilistic logic models and use CP-logic as representation language (Vennekens 2007; Vennekens, Denecker, and Bruynooghe 2009a). CP-logic has the advantage that it has solid formal semantics that are based on probability distributions over the well-founded models of certain logic programs. The language can be motivated and explained in a completely self-contained way as a representation of probabilistic causal laws in such a way that we can say precisely what a theory expresses in terms that are also understandable by a non-logician.

### 3.1.1 Bibliographical note

CP-logic was first proposed as LPADs by Vennekens, Verbaeten, and Bruynooghe (2004), it was later refined, given new semantics and renamed to CP-logic (Vennekens 2007; Vennekens, Denecker, and Bruynooghe 2009a). To extend the knowledge representation efforts in the original paper to the machine laerning setting we proposed some learning strategies in

> H. Blockeel and W. Meert (2006). "Towards learning non-recursive LPADs by transforming them into Bayesian networks". In: *Proceedings ofthe 15th International Conference on Inductive Logic Programming (ILP)*. (Bonn, Germany, Aug. 10–13, 2005). Volume 3625. Lecture Notes in Computer Science, pages 94–108

Because one of the learning strategies involved leveraging Bayesian network learning techniques we focussed also on the connection between CP-logic and Bayesian networks. This connection was later extended in the paper

> W. Meert, J. Struyf, and H. Blockeel (2007). "Learning ground CP-logic theories by means of Bayesian network techniques". In: *Proceedings ofthe 6th workshop on Multi-Relational Data Mining (MRDM)*. (Warsaw, Poland, Sept. 17, 2007), pages 93–104

### 3.1.2 Structure of this chapter

In Section 3.2, we introduce CP-logic syntax and semantics in a non-formal way. Later in Sections 3.3 and 3.4 we introduce the syntax and semantics formally. The original LPAD semantics are introduced in Section 3.5 because of historical reasons and because they offer us a connection to conditional probabilities and Bayesian networks. Next we look a little bit closer into the connections with Bayesian networks in Section 3.6 and logic programming in Section 3.7. We end with a look into the aspects of time and loops in CP-logic in Sections 3.8 and 3.9.

## 3.2 A causal probabilistic logic

A key point for CP-logic is the observation that causality is inherently a dynamic concept (Vennekens, Denecker, and Bruynooghe 2009a). When reasoning about the badminton playing robot, we can formulate statements like "Recognizing the shuttle *causes* the racket being repositioned". Intuitively, this means that when the visual system of the robot identifies the shuttle it sends this information to the controller, and if the information is received, the controller drives the mobile part to the anticipated landing location. Thus, when the shuttle is recognized, it initiates a sequence of events within the robot, and one of the possible outcomes of this process is that the racket is repositioned. As it turns out, whenever we discuss causality, we are, at least implicitly, doing so in the context of the dynamic evolution of a domain. The causal statement itself leaves the details of this process implicit and just asserts its existence: *somehow* recognition causes repositioning.

The aim of CP-logic is to offer a formal language for modeling such causal knowledge, that explicitly incorporates dynamic concepts, such as events and processes, and whose semantics follows naturally from intuitions about these concepts. The fundamental kind of information is why events occur and what the effect of events will be. Concretely, a process starts in some initial state and, through a sequence of possibly non-deterministic events, it probabilistically progresses towards any of a number of possible final states. CP-logic makes the distinction between properties that are *endogenous* (internal) to the process and properties that are *exogenous* (external). The endogenous properties are those that are affected by the process, while the extrogenous properties simply describe the context in which it is taking place.

The formal semantics of CP-logic are given in Sections 3.4 and 3.5 but the semantics adhere to the following four intuitive fundamental principles (Vennekens, Denecker, and Bruynooghe 2009a):

- The principle of *universal causation* states that an endogenous property can only be true if it has been caused by some event, i.e., all changes to the endogenous state of the domain must happen as the consequence of an event.
- The principle of *sufficient causation* states that if an event has a cause, then it must eventually occur.
- The principle of *independent causation* states that every event affects the state of the world in a probabilistically independent way, i.e., knowing the outcome of one event does not give any information about the outcome of a different event. This principle ensures the modularity and robustness of the representation.
- The principle of *temporal precedence* states that, whenever a property $\phi$ might cause an event $E$, then the part of the process that is involved in

determining the truth of $\phi$ happens *before* the event $E$ itself can happen. This principle is motivated by the fundamental property of the physical world that a cause must always preceed its effects. This principle ensures that cyclic causal processes in a CP-theory happen in a consistent way.

These five principles only describe certain aspects of a probabilistic process, but for any given CP-theory, all processes satisfying these principles generate precisely the same probability distribution. This is interesting because, typically, we are not really interested in the actual details of the evolution of a domain, but only care about the probability of arriving at a certain end result. For instance, we are not interested in the relays involved in sending the information from the visual system to the controller, we only want to know the probability that the controller actually receives the information. This shows why causality is such an important concept, causal information is a compact and robust way of specifying the behaviour of a non-deterministic process. For more background on the relation between CP-logic and causality, see Vennekens, Bruynooghe, and Denecker (2010).

## 3.3 Syntax

In this section, we define the language of CP-logic[1]. For simplicity, we assume that we have a logical vocabulary $\Sigma$ available such that any particular state of our domain corresponds to a Herbrand interpretation of $\Sigma$, i.e., a set of ground atoms.

A CP-theory is a set of CP-events or rules of the following form:

$$(h_1 : \alpha_1) \vee (h_2 : \alpha_2) \vee \ldots \vee (h_n : \alpha_n) \leftarrow b_1, b_2, \ldots, b_m.$$

with $h_i$ atoms and $b_i$ literals in the logical sense[2], and $\alpha_i$ *causal* probabilities; $\alpha_i \in [0,1]$, $\sum_{i=1}^{n} \alpha_i \leq 1$, $n \geq 1$, and $m \geq 0$. We call the set of all $(h_i : \alpha_i)$ the *head* of the rule, and the conjunction of literals $b_i$ the *body*. We also refer to the $h_i$ as consequences, and to the $b_i$ as conditions. If the head contains only

[1]CP-logic theories are equivalent to Logic Programs with Annotated Disjunctios (CP-theories) (Vennekens 2007). The research about CP-theories has evolved into CP-logic and we use both terms interchangeable.

[2]The CP-logic semantics allow arbitrary first-order formulas in the body of a rule. For simplicity we restrict ourselves to a conjunction of literals. Since all formulas can be transformed in a disjunction of conjunctions we do not loose generality.

one atom $h : 1$, we may write it as $h$. We assume all variables to be universally quantified and a variable appearing in the head should also appear in the body. If $\sum_{i=1}^{n} \alpha_i < 1$, there is a probability that nothing is caused. This operation can be made explicitly by introducing an *anonymous atom* in the head called *nil* with probability $1 - \sum_{i=1}^{n} \alpha_i$ that is not used in any body.

Each of these rules may have several possible consequences in its head, and each consequence $h_i$ has a causal probability $\alpha_i$ assigned to it. If the body of the rule is true, then the rule makes at most one of these consequences true; the probability that the rule causes $h_i$ to become true is given by $\alpha_i$.

**Example 3.1.** Take a person, named John, who may go to the shop to buy dinner and chooses to buy either spaghetti or steak. This can be formalized as follows:

$$\begin{gathered} shops(john) : 0.2. \\ bought(spaghetti) : 0.5 \vee bought(steak) : 0.5 \leftarrow shops(john). \end{gathered}$$

These rules express that John may go to the shop, and if he does, he buys either spaghetti or steak for dinner, each with 50% chance. (The fact that John goes shopping *causes* the availability of either spaghetti or steak, but not both.)

We can extend the example with John's girlfriend Mary, who may also buy dinner. We assume that she cannot contact John during the day and would like to have dinner in the evening. She, however, buys either spaghetti or fish:

$$\begin{gathered} shops(mary) : 0.9. \\ bought(spaghetti) : 0.3 \vee bought(fish) : 0.7 \leftarrow shops(mary). \end{gathered}$$

This results in multiple rules in the CP-theory that may lead to the same consequence: if John and Mary both buy dinner[3], it is possible that they both buy spaghetti. If they buy something different, they can choose what they will have for dinner, because two meals have been bought. ◊

It is part of the semantics of CP-logic that each rule *independently of all other rules* makes one of its head atoms true when triggered. CP-logic is therefore particularly suitable for describing models that contain a number of independent stochastic events or causal processes.

[3]It is also possible to model the situation where John and Mary decide in advance who goes shopping in order to avoid this situation. To do so, the model can be changed to include the rule $(shops(john) : \alpha_1) \vee (shops(mary) : \alpha_2) \leftarrow .$ instead of the two separate rules for John and Mary.

**Causal Probabilities.** It may be tempting to interpret the CP-theory parameters as the conditional probability of the head atom given the body, e.g., $Pr(bought(spaghetti)|shops(mary)) = 0.3$, but this is incorrect. The conditional probability that spaghetti is bought, given that Mary bought dinner, is higher than 0.3, because there is a second possible cause, namely that John bought spaghetti.

For instance, with

$$\begin{aligned} shops(john) &: 0.2. \\ shops(mary) &: 0.9. \\ bought(spaghetti) : 0.5 \vee bought(steak) &: 0.5 \leftarrow shops(john). \\ bought(spaghetti) : 0.3 \vee bought(fish) &: 0.7 \leftarrow shops(mary). \end{aligned}$$

we can say that $\Pr(bought(spaghetti)|shops(mary)) = 0.3 + 0.7 \cdot 0.2 \cdot 0.5 = 0.37$. Mary buys spaghetti with probability 0.3, but there is also a probability of $1 - 0.3 = 0.7$ that Mary does not buy spaghetti. In that case, it is possible that John goes to the shop with probability 0.2, and buys spaghetti with probability 0.5. This is illustrated in Fig. 3.1.

There is also an alternative explanation for this behaviour. There are two probable reasons why there will be spaghetti, John goes shopping and buys spaghetti ($p_1 = 0.2 \cdot 0.5 = 0.1$), or Mary goes shopping and buys spaghetti ($p_2 = 1 \cdot 0.3 = 0.3$). These two reasons combined give us three possible scenarios for $p_1$ and $p_2$, namely, true-true, false-true, and true-false. Thus, we can say

$$\begin{aligned} \Pr(bought(spaghetti)|shops(mary)) &= p_1 \cdot p_2 + (1 - p_1) \cdot p_2 + p_1 \cdot (1 - p_2) \\ &= 0.1 \cdot 0.3 + 0.9 \cdot 0.3 + 0.1 \cdot 0.7 \\ &= 0.3 + 0.1 \cdot 0.7 = 0.37 \end{aligned}$$

which is the same formula as before. This method is actually applying a noisy-or relation which we will discuss further in Chapter 4.

Thus, for head atoms that occur in multiple rules, the mathematical relationship between the CP-theory parameters and conditional probabilities is somewhat complex, but it is not unintuitive. The meaning of the probabilities in the rules is quite simple: they reflect the probability that the body *causes* the head to become true. This is different from the conditional probability that the head is true given the body, and among the two, *the former is*

$$\begin{tikzcd}
 & \{s(m)\} \arrow[dl, "0.3"'] \arrow[dr, "0.7"] & & & \\
\boxed{\{s(m), b(sp)\}} & & \{s(m), b(fi)\} \arrow[dl, "0.8"'] \arrow[dr, "0.2"] & & \\
 & \{s(m), b(fi)\} & & \{s(m), b(fi), s(j)\} \arrow[dl, "0.5"'] \arrow[dr, "0.5"] & \\
 & & \boxed{\{s(m), b(fi), s(j), b(sp)\}} & & \{s(m), b(fi), s(j), b(st)\}
\end{tikzcd}$$

Figure 3.1: Representation of the causal process given that $shops(mary)$ is true towards all situations where $bought(spaghetti)$ is true (indicated with a rectangle). The $\{a_0, \ldots, a_n\}$ represent an interpretation and the atoms that are true in that interpretation. From left to right the CP-events on line 4, 1 and 3 of the theory are applied to the current interpretation. The probability of a given interpretation is the multiplication of the $\alpha_i$ on the edges, which indicate which head atom is chosen.

*the more natural one to express*. Indeed, the former is local knowledge: an expert can estimate the probability that $shops(mary)$ causes $bought(spaghetti)$ without considering any other possible causes for $bought(spaghetti)$. To infer $Pr(bought(spaghetti)|shops(mary))$, we need global knowledge: we need to know all possible causes for $bought(spaghetti)$, the probability of them occurring, and how they interact with $shops(mary)$.

The fact that CP-theory parameters are local makes it impossible to estimate them directly from training data by simple counting, this in contrast to the global conditional probabilities, which are present in the conditional probability distributions of BNs. However, in Section 3.6, we show that CP-theory parameters can be mapped to BN parameters by introducing unobserved nodes. Such a BN can be learned from training data using BN learning methods such as expectation maximization (EM) (Dempster, Laird, and Rubin 1977), as we will see in Section 7.2.

## 3.4 Process semantics

In this section, we give an intuitive explanation of the semantics of CP-logic. A formal definition can be found in (Vennekens, Denecker, and Bruynooghe 2009a). The basic structure that is considered is a probability tree (Shafer 1996), i.e., a finite tree in which each edge is labeled with a probability, such that the labels of all edges leaving an internal node always sum up to one. Intuitively, such a tree $\mathcal{T}$ represents a probabilistic process: each inner node $n$ is a state of the process representing an interpretation $\mathcal{I}(n)$ and can be mapped to a CP-event $e = \mathcal{E}(n)$ in the theory. Together, the edges leaving a node represent the outcome of the associated CP-event and cause a probabilistic transition to one of its children states. The root is the initial state and the leaves are final states. With $\Pr_{\mathcal{T}}$ we define the probability distribution that a tree $\mathcal{T}$ defines over its nodes. For each leaf $l$, $\Pr_{\mathcal{T}}(l)$ is the product of labels of the edges that lead to $l$.

We now outline how a probability tree $\mathcal{T}$ is build such that it corresponds to the set of given CP-events in a CP-theory. We assume that the CP-events have been grounded with respect to the Herbrand interpretation (and have a finite grounding).

1. Start with a root $\perp$, with $\Pr(\perp) = 1$ and an interpretation $\mathcal{I}(\perp) = \{\emptyset\}$.
2. Associate with the current node $n$ a CP-event $e$ from the CP-events that fulfill the following properties:
    (a) The ground CP-event $e$ is not yet associated with another node.
    (b) The ground CP-event $e$ has at least one atom in $\mathit{head}(e)$ that is not in $\mathcal{I}(n)$.
    (c) All the literals in $\mathit{body}(e)$ are true given the current interpretation $\mathcal{I}(n)$.
    (d) For all negative literals in $\mathit{body}(e)$, all CP-events with those literals in the head are associated with a node on the path to the root or their body cannot become true in the current or a future state. In other words, the precondition is in its final state, it is not only true at this point but is guaranteed to remain true in all potential future states. This step ensures that cyclic causal processes are handled consistently. We first need to apply all possible causes for something to happen before concluding something is not caused.

3. For every $(h_i : \alpha_i)$ in $head(e)$, create a childnode $c_i$ with $\mathcal{I}(c_i) = \mathcal{I}(n) \cup h_i$ and $\Pr(c_i) = \Pr(n) \cdot \alpha_i$.
4. Go to step 2 for every childnode. If no event is found in step 2, stop.

**Example 3.2.** Consider a network where the edges have a probability attached to them. We want to know what the probability is that we can get from one node the another.

$$
\begin{aligned}
&p(X,Y) : 1 \leftarrow e(X,Z), p(Z,Y).\\
&p(X,Y) : 1 \leftarrow e(X,Y).\\
&e(1,2) : 0.5.\\
&e(1,3) : 0.5.\\
&e(2,3) : 0.5.\\
&e(2,4) : 0.5.\\
&e(3,2) : 0.5.
\end{aligned}
$$

We deduce the probability distribution of the theory shown. To keep it compact, we restrict the theory to groundings where $Y = 4$. With this grounding we can calculate the probability of the query $\Pr(p(1,4))$. As is shown in Figure 3.2, we start with an empty interpretation and apply the events in the theory until no event can be applied to any of the branches (in the example we limit ourselves to those choices that can proof the query for simplicity). The leafs are the *possible worlds* for this theory. The result for $\Pr(p(1,4))$ is the sum of all interpretations where $p(1,4)$ is true. $\diamondsuit$

## 3.5 LPAD semantics

Next to the process semantics for CP-logic, we also introduce a second type of semantics for CP-logic: the LPAD semantics (Vennekens, Verbaeten, and Bruynooghe 2004). The LPAD and process semantics have been proven to be equivalent (Vennekens 2007) and allow us to easily define some subclasses of CP-logic. We build on the logic programming concepts that were introduced in chapter 2.

The LPAD semantics of a CP-theory is defined using its grounding.We denote the set of all ground CP-theories with $\mathcal{P}_{\mathcal{G}}$. Given a CP-theory $P$, $\mathcal{I}_P$ is the set of all Herbrand interpretations of $P$.The Herbrand base of $P$ is denoted $H_P$.

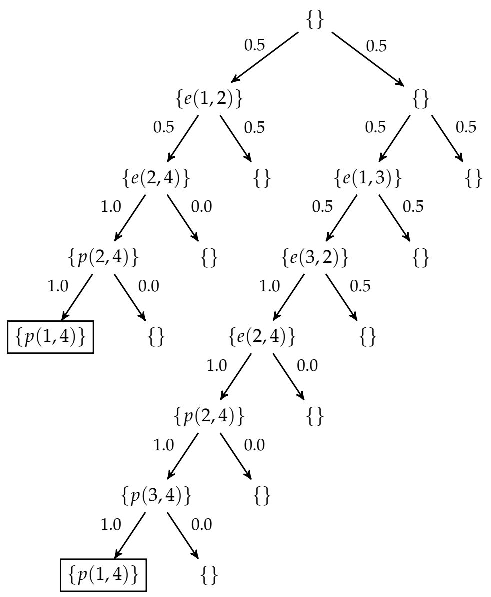


Figure 3.2: Graphical representation of inference for query $p(1,4)$ for Example 3.2. To save space we do not show complete states but the increment with respect to the previous state.

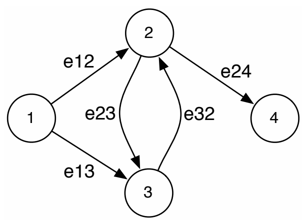


Figure 3.3: Graphical representation of Example 3.2

The semantics of a CP-theory is defined as a probability distribution on $\mathcal{I}_P$, as follows.

**Definition 3.1.** Let $P \in \mathcal{P}_\mathcal{G}$. An admissible probability distribution $\pi$ on $\mathcal{I}_P$ is a mapping from $\mathcal{I}_P$ to $[0,1]$ such that $\sum_{I \in \mathcal{I}_P} \pi(I) = 1$.

**Definition 3.2.** Let $P \in \mathcal{P}_\mathcal{G}$. A selection $\sigma$ is a function that selects one pair $(h : \alpha)$ from each rule of $P$, i.e., $\sigma : P \to (H_P \times [0,1])$ such that for each $c \in P$,

$\sigma(c) \in \mathit{head}(c)$. With $\sigma(c) = h : \alpha$, we also write $\sigma_{atom} = h$ and $\sigma_{prob} = \alpha$. The set of all selections $\sigma$ is denoted by $\mathcal{S}_P$.

**Definition 3.3.** Let $P \in \mathcal{P}_\mathcal{G}$ and $\sigma \in \mathcal{S}_P$. The instance $P_\sigma$ chosen by $\sigma$ is defined as $P_\sigma = \{\sigma_{atom}(c) \leftarrow \mathit{body}(c) \mid c \in P\}$.

**Definition 3.4.** Let $P \in \mathcal{P}_\mathcal{G}$ and $\sigma \in \mathcal{S}_P$. The probability of $\sigma$ is

$$C_\sigma = \prod_{c \in P} \sigma_{prob}(c).$$

This definition of the probability of a selection implies that the selection of a head atom in one rule is stochastically independent from the selection of head atoms in all other rules.

The following definition defines the CP-theories to which we can give meaning:

**Definition 3.5.** A CP-theory $P$ is sound iff for each $\sigma \in \mathcal{S}_P$, the well-founded model of $P_\sigma$, denoted $\mathit{WFM}(P_\sigma)$, is two-valued (an interpretation exists where all atoms are either true or false).

Since we only consider two-valued well-founded models (Gelder, Ross, and Schlipf 1991), we can represent the well-founded model as a single interpretation. We will use this convention in the remainder of this chapter.

Given an interpretation $I$, we denote the set of all $\sigma \in \mathcal{S}_P$ for which $\mathit{WFM}(P_\sigma) = I$ as $\mathcal{S}_P^I$.The semantics of a sound CP-theory is then defined as follows.

**Definition 3.6.** Let $P \in \mathcal{P}_\mathcal{G}$ be a sound CP-theory. For each of its interpretations $I \in \mathcal{I}_P$, the probability $\pi_P^*(I)$ assigned by $P$ to $I$ is the sum of the probabilities of all selections that lead to $I$, i.e.,

$$\pi_P^*(I) = \sum_{\sigma \in \mathcal{S}_P^I} C_\sigma.$$

Vennekens, Verbaeten, and Bruynooghe (2004) prove that if $P$ is a sound CP-theory in $\mathcal{P}_\mathcal{G}$, then $\pi_P^*$ is an admissible probability distribution over $\mathcal{I}_P$. This defines the semantics of any sound CP-theory.

We next recall the definition of the probability of a logic formula, again from Vennekens et al.

**Definition 3.7.** For any logic formula $\phi$, the set of Herbrand models of $\phi$ is denoted and defined as

$$\mathcal{I}_P^{\phi} = \{I \in \mathcal{I}_P \mid I \models \phi\}.$$

**Definition 3.8.** Let $P$ be a sound CP-theory in $\mathcal{P}_{\mathcal{G}}$. The probability of $\phi$ according to $P$, denoted $\pi_P^*(\phi)$, is defined as

$$\pi_P^*(\phi) = \sum_{I \in \mathcal{I}_P^{\phi}} \pi_P^*(I).$$

We add the notion of conditional probability:

**Definition 3.9.** Let $P$ be a sound CP-theory in $\mathcal{P}_{\mathcal{G}}$. The conditional probability of $\phi$ given $\psi$, according to $P$, is denoted and defined as

$$\pi_P^*(\phi|\psi) = \frac{\pi_P^*(\phi \wedge \psi)}{\pi_P^*(\psi)}$$

if $\pi_P^*(\psi) > 0$ (and undefined otherwise).

When $P$ is clear from the context, we will often denote $\pi_P^*(\phi)$as $\Pr(\phi)$ and $\pi_P^*(\phi \mid \psi)$ as $\Pr(\phi \mid \psi)$.

As said before, one should take care not to interpret $\alpha_i$ as $\Pr(h_i \mid B)$, the conditional probability of $h_i$ given the body $B$. However, CP-theories generally do have the property that $\Pr(h_i \mid B) \geq \alpha_i$ (Vennekens, Verbaeten, and Bruynooghe 2004).

## 3.5.1 CP-logic subclasses

When we want to relate the causal probabilities in a CP-theory to conditional probabilities it is useful to look at some specific types of CP-theories. When we know that all causal probabilities in a CP-theory are conditional probabilities, learning is simple as we can use frequency counting to learn the parameters. In this section we identify when the probabilities in a CP-theory are also conditional probabilities and when not.

A first type of theory are CP-theories with mutual exclusive bodies which we call ME-compliant CP-theories and are defined as follows:

**Definition 3.10** (ME-compliant CP-theories)**.** An ME-compliant CP-theory is a CP-theory in which for each two rules $H_1 \leftarrow B_1$ and $H_2 \leftarrow B_2$ it holds that (a) $H_1$ and $H_2$ do not share any atoms, or (b) $B_1$ and $B_2$ are mutually exclusive.

Under these conditions, it holds for each head atom with associated probability $h_i : \alpha_i$ in a rule that $\Pr(h_i \mid B) = \alpha_i$ with $B$ the body of the rule. In general, we call CP-theories fulfilling this property where the causal probability can be expressed as a conditional probability COND-compliant.

**Definition 3.11** (COND-compliant CP-theories)**.** A COND-compliant CP-theory is a CP-theory in which for each rule $H \leftarrow B$ it holds that $\forall(h_i : \alpha_i) \in H : Pr(h_i \mid B) = \alpha_i$.

COND-compliance is important because conditional probabilities can easily be estimated from data: if a CP-theory is COND-compliant, then its parameters can be estimated equally easily. This property is exploited by Riguzzi to learn the $\alpha_i$ parameters of ME-compliant CP-theories from data (Riguzzi 2004).

Now we introduce a different subclass of CP-theories, which (as we shall prove) also has the property that all parameters to be estimated are conditional probabilities. We call this subclass 1-compliant CP-theories. The name refers to the property that each head atom either occurs only in the head of a single rule, or its annotation is 1 in all the heads where it occurs.

**Definition 3.12** (1-compliant CP-theories)**.** A 1-compliant CP-theory is a CP-theory in which for each atom $h$ that occurs in the head of a rule, it holds that either $h$ occurs in only one rule (i.e., it cannot be unified with any atom in the head of any other rule), or it always occurs with an annotation of 1.

In the syntactic sense, our 1-compliant CP-theories are neither a subclass nor a superclass of ME-compliant CP-theories. Riguzzi allows several rules to share head atoms as long as their bodies are mutually exclusive, which is generally not allowed in COND-compliant CP-theories. On the other hand, we allow rules to share head atoms even if their bodies are not mutually exclusive, as long as the probabilities of these atoms are one.

Riguzzi shows that ME-compliant CP-theories are COND-compliant. We now show that 1-compliant CP-theories are COND-compliant.

**Theorem 3.1.** *In a 1-compliant CP-theory, for each rule of the form $h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n \leftarrow B$, $\Pr(h_i \mid B) = \alpha_i$. That is, each $\alpha_i$ can be interpreted as the conditional probability that its atom is true, given that the rule body is true.*

**Algorithm 1** Algorithm for transforming CP-theories into COND-compliant CP-theories

```
**function** Transform($P$: CP-theory) **returns** COND-compliant CP-theory
    $P' := \emptyset$
    **for each** rule $(h_{i1} : \alpha_{i1} \vee \ldots \vee h_{in_i} : \alpha_{in_i} \leftarrow B_i) \in P$:
        let $h'_{ij}$ be $h_{ij}$ with its predicate name $p$ changed into $p_i$
        $P' := P' \cup \{h'_{ij} : \alpha_{i1} \vee \ldots \vee h'_{in_i} : \alpha_{in_i} \leftarrow B_i\}$
        $P' := P' \cup \bigcup_{j=1}^{n_i} \{h_{ij} \leftarrow h'_{ij}\}$
    **return** $P'$
```

*Proof.* According to the definition of a 1-compliant CP-theory, for each $h_i : \alpha_i$ in a rule head with body $B$, it holds that either (a) $h_i$ does not occur in any other rule heads, or (b) $\alpha_i = 1$.

Case (a): $\Pr(h_i \mid B) = \alpha_i$ follows from Riguzzi's proof of Theorem 1 (Riguzzi 2004). While the theorem states that for any rule, $\alpha_i = \Pr(h_i \mid B)$ if all the rules (in the whole CP-theory) sharing head atoms have mutually exclusive bodies, the proof in fact just exploits the mutual exclusion property for the rule for which the equality is proven. Case (a) implies this property.
Case (b): We know from the definition of CP-theories and their semantics that $\alpha_i \leq \Pr(h_i \mid B) \leq 1$. If $\alpha_i = 1$, this implies $\Pr(h_i \mid B) = \alpha_i$. □

## 3.5.2 Transforming CP-theories to 1-compliant CP-theories

We know that 1-compliant CP-theories have causal probabilities that are conditional probabilities. It is known how to learn conditional probabilities from a set of data (Chapter 7), therefore, 1-compliant theories have the advantage that known methods can be used to learn the parameters. In this section we show how to transform any CP-theory into a 1-compliant equivalent CP-theory.

**Example 3.3.** Consider the following CP-theory:

$$bought(spaghetti) : 0.5 \vee bought(steak) : 0.5 \leftarrow shops(john).$$
$$bought(spaghetti) : 0.3 \vee bought(fish) : 0.7 \leftarrow shops(mary).$$

The $i$-th rule is transformed by just adding an index $i$ to each atom in the head:

$$\begin{aligned} bought(spaghetti)_1 : 0.5 \vee bought(steak)_1 : 0.5 &\leftarrow shops(john). \\ bought(spaghetti)_2 : 0.3 \vee bought(fish)_2 : 0.7 &\leftarrow shops(mary). \end{aligned}$$

and the following rules are added:

$$\begin{aligned} bought(spaghetti) &\leftarrow bought(spaghetti)_1. \\ bought(steak) &\leftarrow bought(steak)_1. \\ bought(spaghetti) &\leftarrow bought(spaghetti)_2. \\ bought(fish) &\leftarrow bought(fish)_2. \end{aligned}$$

Note that if the two given CP-events make up the entire theory it is not necessary to transform *bought*(*steak*) and *bought*(*fish*) since they appear in only one head.

$\Diamond$

An algorithm for transforming CP-theories into 1-compliant CP-theories is shown in Algorithm 1. The algorithm adds an index $i$ to the predicate names of all the head atoms of each rule $c_i$, and adds rules stating that the original (unindexed) version of the atom must be true if its indexed version is true.

**Theorem 3.2.** *The transformation yields a 1-compliant CP-theory.*

*Proof.* The resulting program consists of two types of rules: rules with indexed atoms in the head (type 1 rules) and rules with original atoms in the head (type 2 rules). A type 1 rule cannot share a head atom with any other rule: not with a type 2 rule because it has only indexed atoms in the head (and type 2 rules contain only original atoms), and not with other type 1 rules because the indexes differ. Only type 2 rules can therefore share head atoms, but they all have a single head atom with annotation 1. Consequently, the conditions for 1-compliance are fulfilled. $\square$

We now prove that the transformation preserves the LPAD semantics of the CP-theory, in the sense that any logic formula $\phi$ defined over the original CP-theory has the same probability according to the transformed CP-theory.

**Theorem 3.3.** *Let $P \in \mathcal{P}_G$ be a sound CP-theory, and let $P'$ be the transformed version of $P$. For each formula $\phi$ defined over $P$,*

$$\pi^*_P(\phi) = \pi^*_{P'}(\phi).$$

*Proof.* First, we expand the left hand side of the equation:

$$\pi^*_P(\phi) = \sum_{I \in \mathcal{I}^\phi_P} \sum_{\sigma \in \mathcal{S}^I_P} \prod_{r \in P} \sigma_{prob}(r)$$

Define $\mathcal{S}^\phi_P$ as the set of all selections $\sigma$ for which $WFM(P_\sigma) \models \phi$; that is, $\mathcal{S}^\phi_P = \bigcup\{\mathcal{S}^I_P | I \models \phi\}$. We can then shorten the above expression to

$$\pi^*_P(\phi) = \sum_{\sigma \in \mathcal{S}^\phi_P} \prod_{r \in P} \sigma_{prob}(r)$$

Similarly, for the right hand side we have

$$\pi^*_{P'}(\phi) = \sum_{\sigma \in \mathcal{S}^\phi_{P'}} \prod_{r \in P'} \sigma_{prob}(r)$$

So we need to prove

$$\sum_{\sigma \in \mathcal{S}^\phi_P} \prod_{r \in P} \sigma_{prob}(r) = \sum_{\sigma \in \mathcal{S}^\phi_{P'}} \prod_{r \in P'} \sigma_{prob}(r) \tag{3.1}$$

We can define a one-to-one correspondence between $\mathcal{S}_P$ and $\mathcal{S}_{P'}$ as follows. Let $\sigma \in \mathcal{S}_P$ and $\sigma' \in \mathcal{S}_{P'}$ be such that

$$\sigma(P) = \{(h_{1s_1} : \alpha_{1s_1}), \ldots, (h_{ms_m} : \alpha_{ms_m})\}$$

$$\sigma'(P') = \left\{(h'_{1s_1} : \alpha_{1s_1}), \ldots, (h'_{ms_m} : \alpha_{ms_m})\right\} \bigcup \cup_{i=1}^{m} \cup_{j=1}^{n_i} \{h_{ij}\}$$

where $m$ is the number of rules, $n_i$ is the number of head atoms in rule $i$ and $s_i \in [1, n_i]$. This is clearly a one-to-one correspondence because both $\sigma$ and $\sigma'$ map one-to-one to a vector $(s_1, s_2, \ldots, s_m)$. Fig. 3.4 illustrates this correspondence more graphically.

To prove Equation 3.1, it suffices to show that (1) this one-to-one-correspondence carries over to $\mathcal{S}^\phi_P$ and $\mathcal{S}^\phi_{P'}$, that is, $\sigma \in \mathcal{S}^\phi_P \Leftrightarrow \sigma' \in \mathcal{S}^\phi_{P'}$, and (2) the probabilities associated with corresponding selections are the same.

(1) We need to prove $\sigma \in \mathcal{S}^\phi_P \Rightarrow \sigma' \in \mathcal{S}^\phi_{P'}$ and $\sigma \notin \mathcal{S}^\phi_P \Rightarrow \sigma' \notin \mathcal{S}^\phi_{P'}$. But since $\sigma \notin \mathcal{S}^\phi_P$ is equivalent to $\sigma \in \mathcal{S}^{\neg\phi}_P$ (because we require 2-valued logic), it suffices

to prove that the first implication holds for any formula $\phi$.

So assume $\sigma \in \mathcal{S}_P^\phi$; this implies there is an $I$ such that $\sigma \in \mathcal{S}_P^I$ and $I \models \phi$. Now define $I' = I \cup \{h'_{ij} | h_{ij} \in \sigma_{atom}(P)\}$. We will prove that (1a) $\sigma' \in S_{P'}^{I'}$ and (1b) $I' \models \phi$; from this follows $\sigma' \in \mathcal{S}_{P'}^\phi$.

(1a) $\sigma'$ is defined in such a way that $P_\sigma$ contains $h_{ij} \leftarrow B_i$ if and only if $P'_{\sigma'}$ contains the clauses $\{h_{ij} \leftarrow h'_{ij}, h'_{ij} \leftarrow B_i\}$. Thus, whenever $h_{ij}$ can be derived in $P_\sigma$, it can be derived in $P'_{\sigma'}$, and vice versa. In addition, whenever $h_{ij}$ can be derived in $P_\sigma$, $h'_{ij}$ can be derived in $P'_{\sigma'}$. This proves that $WFM(P_\sigma) = I$ if and only if $WFM(P'_{\sigma'}) = I'$, in other words, $\sigma \in \mathcal{S}_P^I \Leftrightarrow \sigma' \in S_{P'}^{I'}$.

(1b) The formula $\phi$ refers only to original (non-indexed) predicates. Since $I'$, restricted to non-indexed predicates, equals $I$, $I' \models \phi$ if and only if $I \models \phi$.

This proves the one-to-one correspondence between $\mathcal{S}_P^\phi$ and $\mathcal{S}_{P'}^\phi$.

(2) If we multiply all the $\sigma_{prob}$ as defined by $\sigma$ and $\sigma'$ we get:

$$\prod_{r \in P} \sigma_{prob}(r) = \alpha_{1s_1} \dots \alpha_{ms_m}$$

$$\prod_{r \in P'} \sigma'_{prob}(r) = \alpha_{1s_1} \dots \alpha_{ms_m} \cdot \underbrace{1 \dots 1}_{\sum_{i=1}^m n_i \text{ times}}$$

Thus the probability of $\sigma$ and $\sigma'$ is the same. This concludes the proof.

□

**Corollary 3.1.** *Any CP-theory can be transformed into a equivalent 1-compliant, and therefore COND-compliant, CP-theory.*

**Corollary 3.2.** *ME-compliant CP-theories can be transformed into equivalent 1-compliant CP-theories.*

## 3.6 Relating CP-logic to Bayesian networks

In a 1-compliant CP-theory, all the causal probabilities less than 1 are also conditional probabilities. Since Bayesian networks are built upon the concept of conditional independence it seems natural that there is a connection between

$$\begin{aligned}
\underline{a:0.3} \vee b:0.4 \vee c:0.3 &\leftarrow B_1. \\
a:0.1 \vee d:0.5 \vee \underline{e:0.1} &\leftarrow B_2. \\
d:0.5 \vee \underline{f:0.5} &\leftarrow B_3.
\end{aligned}
\qquad
\begin{aligned}
\underline{a_1:0.3} \vee b_1:0.4 \vee c_1:0.3 &\leftarrow B_1. \\
a_2:0.1 \vee d_2:0.5 \vee \underline{e_2:0.1} &\leftarrow B_2. \\
d_3:0.5 \vee \underline{f_3:0.5} &\leftarrow B_3. \\
\underline{a} &\leftarrow a_1. \\
\underline{a} &\leftarrow a_2. \\
\underline{b} &\leftarrow b_1. \\
\underline{c} &\leftarrow c_1. \\
\underline{d} &\leftarrow d_2. \\
\underline{d} &\leftarrow d_3. \\
\underline{e} &\leftarrow e_2. \\
\underline{f} &\leftarrow f_3.
\end{aligned}$$

Figure 3.4: An illustration of the one-to-one correspondence between selections in $P$ and in $P'$. For the program $P$ to the left, the 1-compliant version $P'$ is shown to the right. For each selection $\sigma$ for $P$ there is precisely one selection $\sigma'$ for $P'$ according to the defined correspondence, and they have the properties that the probabilities of $\sigma$ and $\sigma'$ are equal and $\text{WFM}(P'_{\sigma'})$ restricted to $H_P$ equals $\text{WFM}(P_\sigma)$.

Bayesian networks and 1-compliant CP-theories. In this section we look deeper into the relation between Bayesian networks and CP-theories.

For now, we assume CP-theories to be non-recursive and have a finite Herbrand universe. We explain the relationship by means of a method to transform a CP-theory to a Bayesian network in such a way that the resulting Bayesian network's parameters are either the original parameters from the CP-theory or 0 or 1. We will call the resulting Bayesian network an *equivalent Bayesian network* (EBN). Infinite Herbrand universes can be related to dynamic Bayesian networks and is discussed in Section 3.8. Recursive theories do not allow for a direct mapping to BNs as they do not allow any cycles but we discuss a transformation from recursive theories to non-recursive theories in Section 3.9.

First, the CP-theory needs to be grounded. From here on, when we refer to the CP-theory, we mean the ground CP-theory. The CP-theory we start from does not need to be 1-compliant, this transformation is part of the conversion to a Bayesian network. Next, the following three steps construct the EBN of a ground CP-theory.

1. For every atom in the CP-theory, a Boolean random variable is created in the EBN. This is a so-called *atom variable* and it is represented by an *atom node* in the network.

2. For every rule in the CP-theory, a *choice variable* is created in the EBN. This variable can take $n + 1$ values, where $n$ is the number of atoms in the head. It is represented by a *choice node* in the network. Note that the choice nodes are unobserved, they are artificial nodes that do not correspond to atoms in the domain. These nodes are included instead of the indexed atoms as explained before. When $C_i = j$, this means the $j$'th atom of rule $i$ has been selected. $C_i = 0$ will be used to denote that no listed head atom was selected, which may be either because the rule body is false, or because no atom was selected when de probabilities do not add up to one.

3. If an atom is in the head of a rule, an edge is created from its corresponding choice node towards the atom's node. If a literal is in the body of a rule, an edge is created from the literal's atom node towards the rule's choice node.

   For positive body literals, we denote the edge to the choice node with '$\leftarrow$'; for negative body literals, we use a dashed arrow '$\dashleftarrow$'. This notation makes it possible to distinguish positive from negative body literals in the EBN, and it ensures that there is a one to one mapping between the EBN structure and the CP-theory rules. Having such a one to one mapping between both structures is relevant for learning CP-theories and will be discussed in Chapter 7. Both types of edges are regular BN edges, the difference in meaning between positive and negative body literals is encoded in the CPT of the choice node, as we will see in the next paragraph.

For the CPTs we have the following two cases:

1. The CPT of a *choice variable* (e.g., Fig. 3.5, CPT for $c_3$). Such a variable can take $n + 1$ values with $n$ the number of atoms in the head. The variable takes the value $i$ if the $i^{th}$ atom from the head is chosen by the probabilistic process. It takes the value 0 if none of the head atoms is chosen (this can only happen if the $\alpha_i$ do not sum to one). If the body of the rule is true, then the probability that the variable takes the value $i \neq 0$ is precisely the causal probability $\alpha_i$ given in the head of the rule. The probability that it takes the value 0 is equal to $1 - \sum_{i=1}^{n} \alpha_i$. If the body is not true, then

the probability that the choice variable takes the value 0 is 1.0 and all the other values have probability 0.0. Formally,

$$\Pr(C_i = j \mid \text{all parents } \mathit{true}) = \alpha_j, \quad \text{for all } j > 0$$

$$\Pr(C_i = 0 \mid \text{all parents } \mathit{true}) = 1 - \textstyle\sum_j \alpha_j$$

$$\Pr(C_i = 0 \mid \text{not all parents } \mathit{true}) = 1$$

$$\Pr(C_i = j \mid \text{not all parents } \mathit{true}) = 0, \quad \text{for all } j > 0$$

The exact column of the CPT that corresponds to 'the body is true' will depend on which body literals are positive and which are negative (as indicated with '$\leftarrow$' or '$\leftarrow$--' in the EBN). For example, if *shops*(*john*) would be negated in the example theory, then the two columns of the CPT for $c_3$ would be swapped.

2. The CPT of an *atom variable* (e.g., Fig. 3.5, CPT for *bought*(*spaghetti*)) is structured differently. It essentially represents a deterministic OR function of the different rules having the atom in the head. More specifically, if one of the choice variables representing a rule with the given atom in position $i$ of its head takes the value $i$, then the atom variable will be true with probability 1.0. In all other cases it will be false, also with probability 1.0. This is because the second part of our CP-theory is essentially a standard logic program. Formally, we can represent the CPD as follows:

$$\Pr(h_i = \mathit{true} \mid \text{all parents false}) = 0$$

$$\Pr(h_i = \mathit{true} \mid \text{at least one parent true}) = 1$$

**Example 3.4.** We repeat the CP-theory in Example 3.3 and extend it to the full shopping example we have used before:

$$\begin{aligned} \mathit{shops}(\mathit{john}) &: 0.2. \\ \mathit{shops}(\mathit{mary}) &: 0.9. \\ \mathit{bought}(\mathit{spaghetti}) : 0.5 \vee \mathit{bought}(\mathit{steak}) &: 0.5 \leftarrow \mathit{shops}(\mathit{john}). \\ \mathit{bought}(\mathit{spaghetti}) : 0.3 \vee \mathit{bought}(\mathit{fish}) &: 0.7 \leftarrow \mathit{shops}(\mathit{mary}). \end{aligned}$$

This theory can be transformed into a 1-compliant form using the algorithm in Section 3.6. The resulting CP-theory is

$$\begin{aligned}
bought(spaghetti)_1 : 0.5 \vee bought(steak)_1 &: 0.5 \leftarrow shops(john).\\
bought(spaghetti)_2 : 0.3 \vee bought(fish)_2 &: 0.7 \leftarrow shops(mary).\\
shops(john)_3 &: 0.2.\\
shops(mary)_4 &: 0.9.\\
bought(spaghetti) &: 1.0 \leftarrow bought(spaghetti)_1.\\
bought(steak) &: 1.0 \leftarrow bought(steak)_1.\\
bought(spaghetti) &: 1.0 \leftarrow bought(spaghetti)_2.\\
bought(fish) &: 1.0 \leftarrow bought(fish)_2.\\
shops(john) &: 1.0 \leftarrow shops(john)_3.\\
shops(mary) &: 1.0 \leftarrow shops(mary)_4.
\end{aligned}$$

This 1-compliant CP-theory can be represented with an equivalent Bayesian net using the algorithm in Section 3.5.2 and the result is shown in Fig.3.5. ♢

**Example 3.5.** Consider the following CP-theory:

$$\begin{aligned}
&a : 0.5 \vee b : 0.5 \leftarrow c.\\
&b : 0.2 \vee c : 0.8 \leftarrow d.
\end{aligned}$$

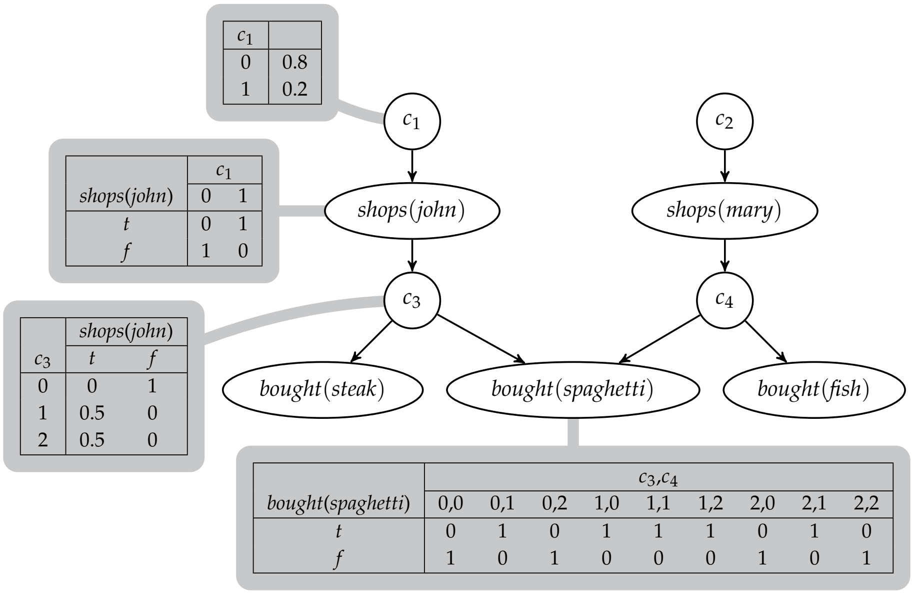
After transformation to a 1-compliant CP-theory this becomes:

$$\begin{aligned}
&a_1 : 0.5 \vee b_1 : 0.5 \leftarrow c.\\
&b_2 : 0.2 \vee c_2 : 0.8 \leftarrow d.\\
&a \leftarrow a_1.\\
&b \leftarrow b_1.\\
&b \leftarrow b_2.\\
&c \leftarrow c_2.
\end{aligned}$$

The equivalent Bayesian network is shown in Fig. 3.6.

The algorithm to produce a Bayesian net from a 1-compliant CP-theory is shown in Algorithm 2.

Thus, given a CP-theory with certain parameters, it can be transformed into a Bayesian network with a specific structure, consisting of two kinds of variables:



| $c_1$ | |
|---|---|
| 0 | 0.8 |
| 1 | 0.2 |

| | $c_1$ | |
|---|---|---|
| *shops*(*john*) | 0 | 1 |
| *t* | 0 | 1 |
| *f* | 1 | 0 |

| | *shops*(*john*) | |
|---|---|---|
| $c_3$ | *t* | *f* |
| 0 | 0 | 1 |
| 1 | 0.5 | 0 |
| 2 | 0.5 | 0 |

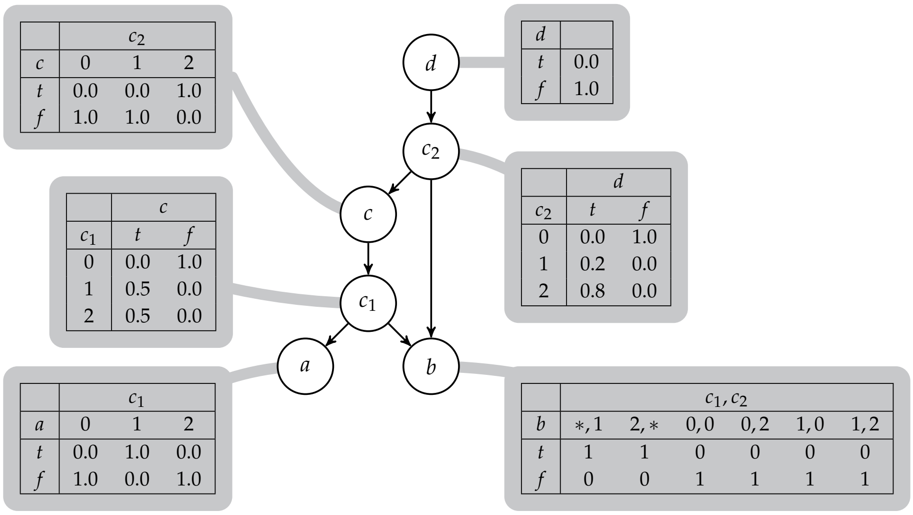
| | $c_3,c_4$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *bought*(*spaghetti*) | 0,0 | 0,1 | 0,2 | 1,0 | 1,1 | 1,2 | 2,0 | 2,1 | 2,2 |
| *t* | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| *f* | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |

Figure 3.5: The equivalent Bayesian network (EBN) for the CP-theory representing the 'shopping example'. The CP-theory parameters appear in the CPTs of the choice nodes $c_i$. The CPT for the *bought*(*spaghetti*) node represents the deterministic relationship $c_3 = 1 \vee c_4 = 1$.

atom variables and choice variables. The CPD's for the atom variables have a fixed structure that is independent of the probabilities in the CP-theory; they always express a logical or. The CPD's for the choice variables have a fixed structure and contain only 0's, 1's, and the CP-theory probabilities $\alpha_{ij}$.

Since we can convert a CP-theory to a specific type of BN, we have two different representations for the same CP-theory. To differentiate between them we will name them. All the possible CP-theories expressed in the CP-logic syntax and semantics will be called the *CP-logic space*. The EBNs resulting from the conversion are part of what we will call the *Bayesian network space*. So, this is the space of all the possible BNs that are equivalent to a valid CP-theory.

The transformation to a Bayesian network shown here introduces some redundant nodes in the network. It is possible to optimize this transformation to obtain a smaller Bayesian network that is equivalent to the original one. These optimizations include (1) avoiding the creation of choice nodes with only one



|  | $c_2$ |  |  |
|---|---|---|---|
| $c$ | 0 | 1 | 2 |
| $t$ | 0.0 | 0.0 | 1.0 |
| $f$ | 1.0 | 1.0 | 0.0 |

| $d$ |  |
|---|---|
| $t$ | 0.0 |
| $f$ | 1.0 |

|  | $d$ |  |
|---|---|---|
| $c_2$ | $t$ | $f$ |
| 0 | 0.0 | 1.0 |
| 1 | 0.2 | 0.0 |
| 2 | 0.8 | 0.0 |

|  | $c$ |  |
|---|---|---|
| $c_1$ | $t$ | $f$ |
| 0 | 0.0 | 1.0 |
| 1 | 0.5 | 0.0 |
| 2 | 0.5 | 0.0 |

|  | $c_1$ |  |  |
|---|---|---|---|
| $a$ | 0 | 1 | 2 |
| $t$ | 0.0 | 1.0 | 0.0 |
| $f$ | 1.0 | 0.0 | 1.0 |

|  | $c_1, c_2$ |  |  |  |  |  |
|---|---|---|---|---|---|---|
| $b$ | $*, 1$ | $2, *$ | $0, 0$ | $0, 2$ | $1, 0$ | $1, 2$ |
| $t$ | 1 | 1 | 0 | 0 | 0 | 0 |
| $f$ | 0 | 0 | 1 | 1 | 1 | 1 |

Figure 3.6: Bayesian net corresponding to the CP-theory given in Example 3.5. Since there is no CP-event causing $d$, it will be false with probability 1.

child and (2) merge choice nodes for CP-events with mutual exclusive bodies. These optimizations are explained in more detail in Chapter 6.

## 3.7 Relating CP-logic to logic programming

When looking at the LPAD semantics for CP-logic, it is clear that there is a relation to logic programming. A CP-theory represents a set of logic programs, each resulting in a single interpretation and with a probability associated with it. When using the LPAD semantics, a CP-theory can be seen as a probabilistic extension of logic programming that is based on disjunctive logic programs (Lobo, Minker, and Rajasekar 1992). This is a natural choice, because disjunctions express a kind of uncertainty. The idea behind the LPAD semantics was to quantify this uncertainty by annotating the disjuncts in the head with a probability.

**Algorithm 2** Algorithm for transforming non-recursive COND-compliant CP-theories into Bayesian networks. By convention, $h_{ij}$ refers to original atoms in this description, and $h'_{ij}$ to indexed atoms.

```
function BN(P: 1-compliant CP-theory) returns a Bayesian net
    $N := \emptyset$ // nodes of the BN
    $E := \emptyset$ // edges of the BN
    for each rule $(h'_{i1} : \alpha_{i1} \vee \ldots \vee h'_{in_i} : \alpha_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}) \in P$:
        $N := N \cup \{C_i\} \cup \bigcup_j \{b_{ij}\}$
        $E := E \cup \bigcup_j \{(b_{ij}, C_i)\}$
        associate with $C_i$ a CPD as follows:
            $P(C_i = j|\text{all parents true}) = \alpha_{ij}$, for all $j > 0$
            $P(C_i = 0|\text{all parents true}) = 1 - \sum_j \alpha_{ij}$
            $P(C_i = 0|\text{not all parents true}) = 1$
            $P(C_i = j|\text{not all parents true}) = 0$, for all $j > 0$
    for each clause $(h_{ij} \leftarrow h'_{ij}) \in P$:
        $N := N \cup \{h_{ij}\}$
        $E := E \cup \bigcup_j \{(C_i, h_{ij})\}$
    for each $l \in N$ that is not a $C_i$:
        associate with $l$ a CPD as follows:
            $C := \bigvee_{ij:(l \leftarrow h'_{ij}) \in P} C_i = j$
            $P(l = true|C) = 1$
            $P(l = false|C) = 0$
            $P(l = true|\neg C) = 0$
            $P(l = false|\neg C) = 1$
    return $(N, E, CPD)$
```

## 3.8 Time in CP-logic

Since the CP-logic semantics describe a process, the concept of time is implicitly present in a theory. Often, we do not care about the exact order of events with respect to time and are only interested in the final states. But for some problems it is necessary to explicitly encode time. For example, when forecasting patient conditions we explicitly refer to observations from previous days.

The first type of time representation, implicit, is similar to how time is represented in, for example, causal Bayesian networks. The second type is more typical of logic-based languages but is also used in, for example, dynamic Bayesian networks. Both styles are useful and natural ways of reasoning about causal events. CP-logic allows both types of representation, even together in the same theory.

**Example 3.6.** The shopping example (Ex. 3.1) uses an implicit representation of time. We do not need to encode who goes shopping when since it is irrelevant. Now, suppose we want to add the knowledge that John does not like spaghetti two days in a row. In that case we can alter the theory to explicitely encode time information that John uses to decide what to buy. To save space, we represent *bought* by $b$ and *shops* by $s$.

$$\begin{aligned}
s(john, T) &: 0.2. \\
s(mary, T) &: 0.9. \\
b(spaghetti, T) : 0.5 \vee b(steak, T) &: 0.5 \leftarrow s(john, T), \neg b(spaghetti, T-1). \\
b(steak, T) &: 1.0 \leftarrow s(john, T), b(spaghetti, T-1). \\
b(spaghetti, T) : 0.3 \vee b(fish, T) &: 0.7 \leftarrow s(mary, T).
\end{aligned}$$

The buying process of the two people is still implicitely timed, but the shopping behaviour with respect to the previous day is now explicitely timed. ◊

CP-logic with an explicit time step can be seen as a dynamic BN. A special case of a dynamic BN is an HMM which represents a sequence of states and the state can only be indirectly observed. In the case of an HMM, the set of CP-events defining the state at a particular time step based on the previous time step can be represented by a finite state diagram like in the unobserved nodes in an HMM.

## 3.9 Probabilistic loops

A CP-theory is called *cyclic* if it contains at least two events that depend on each other. Transforming such a theory using the method from Section 3.6 yields a network with a directed cycle which is not allowed in a Bayesian network. For such theories the CP-logic semantics differ too much from those of Bayesian networks.

**Example 3.7.** Consider the following theory:

$$a : 0.5.$$
$$b : 0.5.$$
$$a : 0.5 \leftarrow b.$$
$$b : 0.5 \leftarrow a.$$

The third event depends on $b$ to cause $a$ and the fourth event depends on $a$ to cause $b$. They depend on each other to cause the atoms in their heads. CP-logic uses the concept of temporal precedence to handle such cycles, resulting in a finite and consistent probability distribution over the final states (see Figure 3.7). Bayesian networks have no such concept as temporal precedence, therefore the

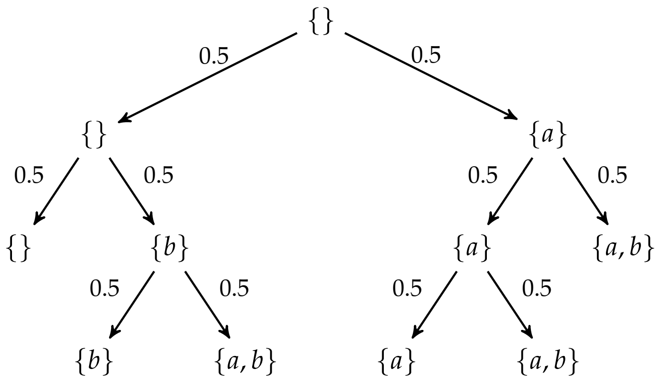


Figure 3.7: Graphical representation of the process semantics for Example 3.7.

method mentioned before cannot be applied naievely to create an equivalent Bayesian network. ♢

However, for every cyclic CP-theory (using a finite Herbrand universe) it is possible to convert the theory to an equivalent acyclic one. An example for a loop of size 2 is given in Vennekens (2007) where it is proposed to insert artificial predicates to break the cycles and generalized for only a limited set of larger cycles. Here we extend this reasoning to arbitrary cycles of arbitrary size.

A cyclic theory is transformed into an acyclic one by unfolding the cycles (also called probabilistic loops) in the theory. If the theory contains multiple interacting cycles we speak about a *strongly connected component* and every possible path in the component needs to be unfolded. In general we can transform the strongly connected component of size $n$ into an $n$ by $n$ set of nodes interconnected such that this grid represents all possible paths in the strongly connected component.

Concretely, a strongly connected component of size $n$ can be transformed as follows:

1. For a predicate $p(args)$ in the strongly connected component add $n-1$ artificial predicates $p_i(args)$ with $0 \leq i < n$.
2. Replace all occurrences of $p(args)$ in the head of events not in the strongly connected component with $p_0(args)$.
3. Replace all events where at least one predicate in the strongly connected component appears in the head and one in the body with $n-1$ events. All the predicates that are in the head and in the strongly connected component are replaced with a predicate $p_{i+1}(args)$ and all predicates in the body and strongly connected component with a predicate $p_i(args)$.
4. The artificial predicates are connected such that if one of the artificial prediactes is caused, the original predicate is also caused. This is done by introducing events $p(args) : 1.0 \leftarrow p_i(args)$.

The resulting structure can be simplified further for those atoms that appear in the strongly connected component but are only caused by other atoms in the component. For such an atom $p(args)$ there will be no event where $p_0(args)$ is in the head and this atom will always be false and can be removed again. This reasoning can be applied iteratively on all atoms in the grid.

Once an acyclic theory has been found, conversions to Bayesians networks are again possible to perform inference.

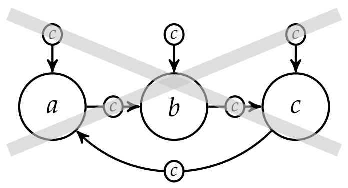
**Example 3.8.** Given the following cyclic theory

$$\begin{aligned}
&\quad a : \alpha \leftarrow c. \\
&\quad b : \beta \leftarrow a. \\
&\quad c : \gamma \leftarrow b. \\
&a : 0.5. \\
&b : 0.5. \\
&c : 0.5.
\end{aligned}$$

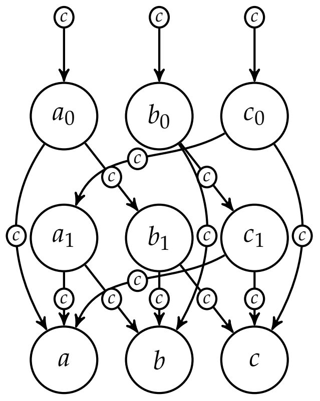
The algorithm above transform this into

$$\begin{array}{lll}
a : 1.0 \leftarrow a_0. & b : 1.0 \leftarrow b_0. & c : 1.0 \leftarrow c_0. \\
a : 1.0 \leftarrow a_1. & b : 1.0 \leftarrow b_1. & c : 1.0 \leftarrow c_1. \\
\\
a : \alpha \leftarrow c_1. & a_1 : \alpha \leftarrow c_0. & a_0 : 0.5. \\
b : \beta \leftarrow a_1. & b_1 : \beta \leftarrow a_0. & b_0 : 0.5. \\
c : \gamma \leftarrow b_1. & c_1 : \gamma \leftarrow b_0. & c_0 : 0.5.
\end{array}$$

The equivalent Bayesian network is shown in Figure 3.8. ◊



(a)



(b)

Figure 3.8: Equivalent Bayesian network of the theory in Example 3.8 obtained after (a) naievely applying the transformation, and (b) first unfolding the cyclic theory to an acyclic on. To make the figure more clear we anotated all choice nodes with $c$ but these are all diferent random variables representing the different CP-events in the theory.

## 3.10 Implementations

Up to now, CP-logic inference has been implemented into three systems: Corporal, ProbLog and cplint.

**Corporal** Corporal is the system implemented for the experiments performed in this dissertation. It is based on the transformation introduced in Sec. 3.6 from CP-logic to Bayesian networks and utilizes specialized Bayesian network inference techniques that are introduced in Chapter 6. As indicated before, the transformation to a Bayesian network starts from a ground CP-theory. The Corporal system will not naively ground the entire system but uses the more intelligent approach explained in Chapter 5.

**ProbLog** The ProbLog system (De Raedt, Kimmig, and Toivonen 2007; Kimmig et al. 2008) is a powerful low-level language to express probabilistic logic models and can be used as an inference engine for most existing probabilistic logic languages. The language is a probabilistic extension to Prolog and the inference engine is based on BDD manipulations and dynamic programming. CP-logic can be translated to the ProbLog language (De Raedt et al. 2008) and a CP-logic pre-processing script is available in the ProbLog distribution.[4] ProbLog is further discussed in Sections 4.7 and 6.2.1.

**Cplint** Inspired by ProbLog, Riguzzi (2007) proposed "cplint"[5], which is a CP-theory inference system that makes use of BDDs in a similar way as ProbLog but it uses a different encoding for the BDDs. In Section 6.2.1 we discuss cplint.

## 3.11 Conclusions

In this chapter we gave an introduction to the CP-logic formalism, a language in the field of probabilistic logic learning. We introduced an algorithm to convert CP-theories to Bayesian networks without losing the original structure. Such a transformation allows us to leverage Bayesian network inference and learning methods and use them for CP-logic.

[4] http://dtai.cs.kuleuven.be/problog/

[5] http://www.ing.unife.it/software/cplint/

The transformation to Bayesian networks forces us to remove cycles in a CP-theory in order to create acyclic CP-theories as Bayesian networks cannot contain directed cycles. This problem is more general and also shows in other PLL formalisms like ProbLog. Such probabilistic loops cannot be handled as loops in a logic program or cycles in a Bayesian network (where they are not allowed).

Since CP-logic is more fine-grained than Bayesian networks, the structure of the equivalent Bayesian network cannot express the structure of the CP-theory on the same level. This is handled in the structure of the factors. As it turns out, this is not always the most efficient approach and some optimizations are described in Chapters 5 and 6.

# 4 Related formalisms

## 4.1 Introduction

In this chapter we compare CP-logic to other probabilistic logic formalisms. CP-logic is closely related to other probabilistic logic formalisms that also originate from logic, such as *Independent Choice Logic (ICL)* (Poole 1997), *Programming in Statistical Modelling (PRISM)* (Sato and Kameya 2008) and *ProbLog* (De Raedt, Kimmig, and Toivonen 2007). On the other hand, because we can translate a grounded CP-theory to a Bayesian network, CP-logic is also related to the *Knowledge Based Model Construction (KBMC)* approach (Breese, Goldman, and Wellman 1994). Given a specific query, these languages instantiate from the logical representation a propositional model and then perform inference. Examples of this approach are *Bayesian Logic Programs (BLP)* (Kersting and De Raedt 2008), *Logical Bayesian Networks (LBN)* (Fierens et al. 2004) and *Probabilistic Relational Models (PRM)* (Friedman, Goldszmidt, and Wyner 1999). All these formalisms can be categorized as *directed probabilistic logic* formalisms. Probabilistic logics that are less related to CP-logic include *Stochastic Logic Programs (SLP)* (Muggleton 2000) and *Markov Logic Networks (MLN)* (Richardson and Domingos 2006).

### 4.1.1 Bibliographical note

The comparison between CP-logic and Bayesian networks and Bayesian logic programs were presented in

> W. Meert, J. Struyf, and H. Blockeel (2008b). "Learning ground CP-logic theories by leveraging Bayesian network learning techniques." In: *Fundamenta Informaticae* 89(1), pages 131–160

A more practical comparison between some of the probabilistic logic models can be found in Bruynooghe et al. (2009) for which the author is a co-author. CHRiSM (see Section 4.6) is introduced by J. Sneyers in (Sneyers, Meert, and Vennekens 2009) and (Sneyers et al. 2010b) and for both papers the author is a co-author.

### 4.1.2 Structure of this chapter

In the following, we first offer some more insight in the relation between Bayesian networks and CP-logic in Section 4.2, then we compare CP-logic to some of the most related probabilistic logic formalisms: ICL (Section 4.4), PRISM (Section 4.5), CHRiSM (Section 4.6), ProbLog (Section 4.7) and BLPs (Section 4.8).

## 4.2 Bayesian networks

We first offer some insight in how the difference in language bias of CP-logic and Bayesian networks results in different representations and interpretations. In this section we assume ground CP-theories and do not consider the use of logic variables. In the next section more insight is given in the use of logic variables.

This section compares the EBN networks introduced in Section 3.6 to regular BNs defined over the original atoms, that is, networks with precisely one Boolean node for each atom in the domain, no additional (unobserved) nodes, and with the conditional probability distributions represented with tables (CPTs). Given that BNs can represent any probability distribution, they can also represent any CP-theory (remember, a BN can directly be directly encoded as a

CP-theory). However, as we will see shortly in the comparison, some simple CP-theories can be represented with simple networks (with a small number of edges and known CPT structures, such as noisy-OR), while other, equally simple, CP-theories can only be represented with fully connected networks. We start by looking at the most simple BN structure that is equivalent to a CP-theory, and then gradually consider more complex ones.

This comparison extends an earlier theoretical comparison by Vennekens (Vennekens, Denecker, and Bruynooghe 2009b). We start by comparing CP-theories to BNs with noisy-OR nodes (Pearl 1988; Vomlel 2006; Heckerman and Brees 1994). Noisy-OR nodes appear in BNs when there is a set of independent and noisy causes for a given variable. We show that BNs with noisy-OR nodes are not sufficient to model CP-theories in a compact way. That is, representing CP-theories with regular BNs may require close to fully connected networks, unless artificial unobserved nodes are introduced in the network. Precisely such nodes are created by Blockeel and Meert's transformation; these nodes model the non-determinism in the outcome of the event.

### 4.2.1 CP-Theories Representing Noisy-OR

Noisy-OR nodes are well known in the field of BNs (Pearl 1988). We first briefly explain the semantics for the general case of causal independence models of which noisy-OR is a special case and after that we show that they model the same probability distribution as the most simple CP-theories.

The global structure of a causal-independence model is given in Figure 4.1 and expresses the idea that causes $b_1, \ldots, b_n$ influence a given common effect $h$ through intermediate variable $c_1, \ldots, c_n$ and a Boolean function $b$, the *interaction function* (Lucas 2005). The influence of each cause $b_i$ on the common effect $h$ is independent of the other causes $b_j$, $j \neq i$. The function $b$ represents in which way he intermediate effects $c_i$, and indirectly also $b_i$, interact to yield the final effect $h$.

The factorization of the BN given in Figure 4.1 can be denoted as:

$$\Pr(h \mid b_1, \ldots, b_n) = \sum_{c_1, \ldots, c_n} \Pr(h \mid c_1, \ldots, c_n) \Pr(c_1, \ldots, c_n \mid b_1, \ldots, b_n)$$

We see that the causes only influence the common effect when $\Pr(h \mid c_1, \ldots, c_n) = 1$. This is a functional dependency since this corresponds to the Boolean function $b$ where $b(c_1, \ldots, c_n) = h$. We know that the causes are independent of each other, thus

$$\Pr(c_1, \ldots, c_n \mid b_1, \ldots, b_n) = \prod_{i=1}^{n} \Pr(c_i \mid b_i)$$

and we can simplify the original formula to

$$\Pr(h \mid b_1, \ldots, b_n) = \sum_{b(c_1=v_1, \ldots, c_n=v_n)=h} \prod_{i=1}^{n} \Pr(c_i = v_i \mid b_i)$$

The size of the Boolean formula $b$ is exponential in the number of its arguments, thus the previous equation is not guaranteed to be computationally tractable. An important subclass of causal independence models, however, are those whose deterministic function $b$ is *decomposable*. In that case, the Boolean function can be defined in terms of separate functions $g_i(c_i, c_{i+1})$ introducing symmetry we can exploit. Typically, examples of decomposable causal independence models are the noiys-OR and noisy-MAX models, where the function $g_i$ represents a logical OR and MAX, respectively.

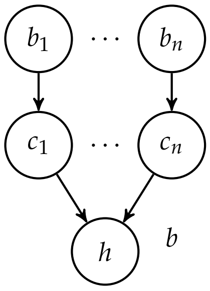
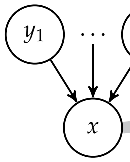


Figure 4.1: Causal independence model.

We focus on noisy-OR causal independence models as they are implicitly present in CP-logic. It is also possible to represent a noisy-OR model without the intermediate nodes by simply marginalizing them out. Noisy-OR can then be represented with a BN node as shown in Fig. 4.2.a. Its conditional probability distribution takes the following form (Neapolitan 2003):



a. BN with noisy-or node $x$

|  | $y_1, y_2$ |  |  |  |
|---|---|---|---|---|
| $x$ | *true, true* | *true, false* | *false, true* | *false, false* |
| *true* | $1 - \bar{\alpha}_1\bar{\alpha}_2$ | $\alpha_1$ | $\alpha_2$ | 0.0 |
| *false* | $\bar{\alpha}_1\bar{\alpha}_2$ | $\bar{\alpha}_1$ | $\bar{\alpha}_2$ | 1.0 |

b. CPT for a noisy-or node with two inputs ($\bar{\alpha}_i = 1 - \alpha_i$)

Figure 4.2: A BN representing the noisy-OR relationship.

$$Pr(x = true \mid y_1, \ldots, y_n) = 1 - \prod_{i:(y_i = T)} (1 - \alpha_i) \tag{4.1}$$

Fig. 4.2.b shows this distribution represented as a table.

With the above definition of noisy-OR, $x$ can only become true if one of the causes $y_i$ is true. In practice, there may also be unknown causes for $x$. These can either be modelled by adding an unobserved node to the network and making this one of the causes for $x$ (lumping together the effect of the unknown causes). Alternatively, one can also modify the definition of noisy-OR and include an additional parameter $\beta$ representing the unknown causes. The distribution then becomes:

$$Pr(x = true \mid y_1, \ldots, y_n) = 1 - (1 - \beta) \cdot \prod_{i:(y_i = T)} (1 - \alpha_i) \tag{4.2}$$

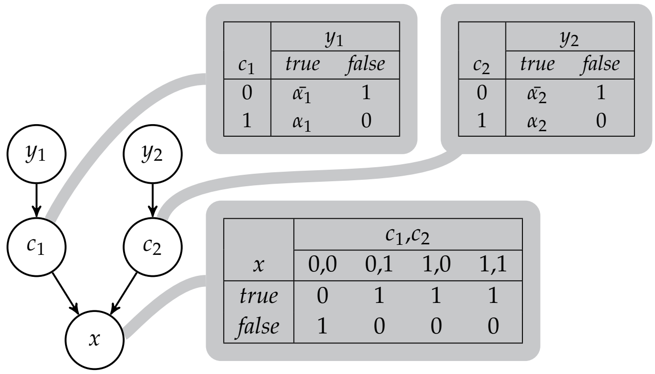
A noisy-OR node $x$ with inputs $y_i$ represents exactly the same probability distribution as the CP-theory consisting of the rules $x : \alpha_i \leftarrow y_i$ (Fig. 4.3.a). This can be seen most easily by looking at the EBN of the theory, which is shown in Fig. 4.3.b. The EBN matches the definition of noisy-OR given above: the choice nodes $c_i$ of the EBN represent the noisy versions of the inputs $y_i$, and the CPT for node $x$ represents exactly the deterministic OR of these noisy inputs.

Because the CP-theory of Fig. 4.3.a is equivalent to a noisy-OR node, it can also be represented, without the choice nodes, as the regular BN shown in Fig. 4.2. More generally, any non-recursive ground CP-theory consisting of

$x : \alpha_1 \leftarrow y_1.$

$x : \alpha_2 \leftarrow y_2.$

a. CP-theory



b. EBN

|  | $y_1$ |  |
|---|---|---|
| $c_1$ | *true* | *false* |
| 0 | $\bar{\alpha}_1$ | 1 |
| 1 | $\alpha_1$ | 0 |

|  | $y_2$ |  |
|---|---|---|
| $c_2$ | *true* | *false* |
| 0 | $\bar{\alpha}_2$ | 1 |
| 1 | $\alpha_2$ | 0 |

|  | $c_1,c_2$ |  |  |  |
|---|---|---|---|---|
| $x$ | 0,0 | 0,1 | 1,0 | 1,1 |
| *true* | 0 | 1 | 1 | 1 |
| *false* | 1 | 0 | 0 | 0 |

c. CPTs for the EBN ($\bar{\alpha}_i = 1 - \alpha_i$)

Figure 4.3: A CP-theory representing noisy-OR together with its EBN and CPTs. The choice nodes $c_i$ represent the noisy versions of the inputs $y_i$, and the node $x$ represents the deterministic OR. This EBN is equivalent with the noisy-OR in Fig. 4.3, as can be seen by marginalizing out the choice nodes : $Pr(x|y_1, y_2) = \sum_{c_1,c_2} Pr(x|c_1, c_2) \cdot Pr(c_1|y_1) \cdot Pr(c_2|y_2)$.

rules with precisely one atom in the head and at most one atom in the body can be represented as a BN consisting of only noisy-OR nodes. These may include an unknown cause as in Eq. 4.2 to account for rules with empty bodies. This network has precisely one edge $x_j \leftarrow y_i$ for each CP-theory rule $x_j : \alpha_{i,j} \leftarrow y_i$. The connection between the rules in a CP-theory and noisy-OR is not only present in CP-logic but already appeared in rule-based expert systems before they became popular in BNs (Lucas 2001).

Assume that the CP-theory has the structure defined in the previous paragraph, then we know the corresponding structure for the BN and for its conditional probability distributions. We now briefly discuss learning the parameters of such a network. For theories with a large number of causes for a given atom, parameter learning with techniques only taking conditional independence into account (thus, estimating CPT entries by means of counting) is intractable. Indeed, represented as a table, the conditional probability distribution of a noisy-OR node with $n$ inputs has $2 \cdot 2^n$ table entries (Fig. 4.2.b), and learning all these entries from training data is infeasible for large values of $n$. Therefore, we need techniques that explicitly take the special format of the conditional probability distributions into account. We discuss such a parameter learning technique for CP-theories in Section 7.2.

Some known BN learning algorithms have special support for noisy-OR nodes. These algorithms represent the noisy-OR conditional probability distribution implicitly as in Eq. 4.1. This avoids representing and learning exponentially large CPTs because Eq. 4.1 contains only one parameter for each cause. An example of an algorithm in this category is the noisy-OR classifier (Vomlel 2006; Heckerman and Brees 1994), which consists of precisely one noisy-OR node for the class with one input node for each descriptive attribute. The noisy-OR classifier has been shown to perform well in tasks with a large number of attributes. It follows trivially from the above discussion that the noisy-OR classifier can be compactly represented as a CP-theory. Therefore, CP-theories can also be used in such classification tasks. Training the classifier then corresponds to learning the CP-theory parameters.

While noisy-OR can be represented as a CP-theory, the opposite does not generally hold, as we will see next.

### 4.2.2 CP-Theories with Multiple Literals in the Rule Bodies

When multiple literals occur in the body of a rule, it is no longer possible to represent the theory as a BN with only noisy-OR nodes. One either has to add additional nodes to represent the deterministic AND between the literals in the body, or one has to combine the noisy-OR and AND function in one CPT. Fig. 4.4 illustrates both options. Note that combining noisy-OR and AND results in a redundant CPT: many entries have the same value. The addition of separate AND nodes is necessary to avoid this redundancy.

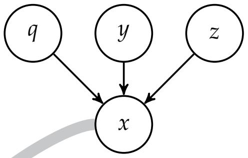
### 4.2.3 CP-Theories with Multiple Atoms in the Rule Heads

The BN conversion becomes more difficult when there are multiple atoms in the head of a rule. Take for example the rule in Fig. 4.5.a. This rule causes precisely one of the atoms in the head to become true (assuming $\alpha + \beta + \gamma = 1$), that is, the head atoms are mutually exclusive. Modelling such a relationship in a BN, without any additional nodes, requires a fully connected network, which is shown in Fig. 4.5.b. In the particular factorization shown in the figure, $z$ depends on both $x$ and $y$ because it can only become true if $x$ and $y$ are false. It is not possible to represent mutual exclusivity with fewer edges, and in general,

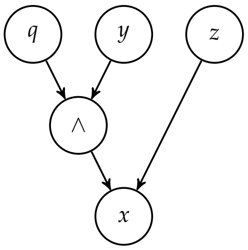
$x : \alpha \leftarrow q, y.$

$x : \beta \leftarrow z.$

a. CP-theory.



b. Corresponding BN.



c. Alternative BN with AND node.

| | $q,y,z$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x$ | $t,t,t$ | $t,t,f$ | $t,f,t$ | $f,t,t$ | $f,f,t$ | $t,f,f$ | $f,t,f$ | $f,f,f$ |
| $t$ | $1-(1-\alpha)(1-\beta)$ | $\alpha$ | $\beta$ | $\beta$ | $\beta$ | 0 | 0 | 0 |
| $f$ | $(1-\alpha)(1-\beta)$ | $1-\alpha$ | $1-\beta$ | $1-\beta$ | $1-\beta$ | 1 | 1 | 1 |

d. CPT for node $x$ in (b)

Figure 4.4: CP-theory with multiple literals in the body of a rule and corresponding BNs.

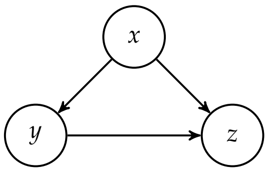
rules with multiple atoms in the head will always result in fully connected sub-networks.

EBNs, on the other hand, avoid these fully connected sub-networks by introducing choice nodes: the choice variable encodes exactly which of the head atoms is selected by the probabilistic process. This trivially ensures mutual exclusiveness because the choice variable cannot have more than one value at a given time. To summarize, to represent a CP-theory, it is not sufficient that the BN has noisy-OR and deterministic AND nodes because it also has to express the mutual exclusiveness of the head atoms; additional nodes are needed for this.

The fact that CP-logic offers an elegant way of encoding mutually exclusive consequences is an important point. Not only does it allow for a more intuitive representation, it also does this with fewer parameters than a regular BN without unobserved nodes. Because such relationships are easily representable in CP-logic, it will also be more easy to learn them, as we will show later.

$$x : \alpha \vee y : \beta \vee z : \gamma \leftarrow .$$

a. CP-theory.



b. Corresponding BN.

| $x$ | |
|---|---|
| *true* | $\alpha$ |
| *false* | $1-\alpha$ |

c. CPT for $x$.

| | $x$ | |
|---|---|---|
| $y$ | *true* | *false* |
| *true* | 0 | $\frac{\beta}{1-\alpha}$ |
| *false* | 1 | $1-\frac{\beta}{1-\alpha}$ |

d. CPT for $y$.

| | $x,y$ | | | |
|---|---|---|---|---|
| $z$ | *true, true* | *true, false* | *false, true* | *false, false* |
| *true* | 0 | 0 | 0 | $\frac{\gamma}{1-\alpha-\beta}$ |
| *false* | 1 | 1 | 1 | $1-\frac{\gamma}{1-\alpha-\beta}$ |

e. CPT for $z$.

Figure 4.5: CP-theory with multiple atoms in the head of a rule and corresponding BN.

### 4.2.4 Multiple literals in head and shared across events

**Example 4.1.** We repeat the CP-theory from Example 3.1 representing a shopping situation. This theory is used as a running example in this section. The equivalent Bayesian network is depicted in Figure 4.6.

$$\begin{aligned} shops(john) &: 0.2. \\ shops(mary) &: 0.9. \\ bought(spaghetti) : 0.5 \vee bought(steak) &: 0.5 \leftarrow shops(john). \\ bought(spaghetti) : 0.3 \vee bought(fish) &: 0.7 \leftarrow shops(mary). \end{aligned}$$

◊

The CP-theory of the shopping example presented in Example 4.1 contains multiple causes for the same atom as well as multiple atoms in the head of a rule. Therefore, it is necessary to combine the noisy-OR and the mutual

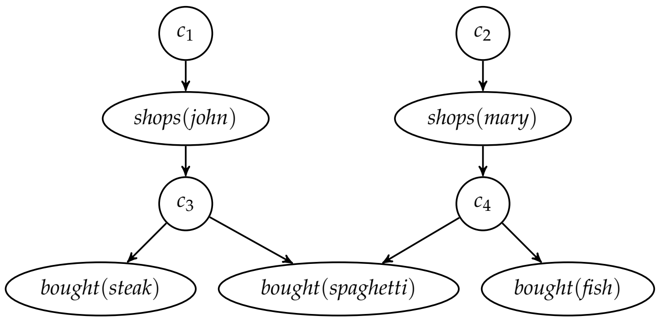


Figure 4.6: The equivalent Bayesian network for the shopping theory in Example 4.1.

exclusivity relationships explained before. This leads to redundancy unless additional nodes are introduced in the network. Fig. 4.7 shows the resulting BN if no additional nodes are introduced, which is almost fully connected.

This representation is clearly less intuitive than the CP-theory. Even though the act that John goes shopping does not cause the buying of fish, there is a direct edge from *shops*(*john*) towards *bought*(*fish*). This is because a BN is designed to represent correlations whereas CP-theories are designed to represent causal interactions. If one observes that *bought*(*spaghetti*) is true, *bought*(*steak*) is false and *shops*(*mary*) is true, then *shops*(*john*) and *bought*(*fish*) are not independent[1] and an edge between them is necessary because all other paths are d-separated.

We conclude that in such cases the model represented as a CP-theory is (a) more intuitive, because there are only edges between elements that actually have a causal relation, and (b) more efficient, since in the BN model, we also need parameters to quantify $Pr(bought(fish)|shops(john), ...)$, which are not needed in the CP-theory.

### 4.2.5 Summary

In this section, we considered representing CP-theories as regular BNs, without any additional nodes besides the domain atoms. Below we summarize the main conclusions.

[1]This can be illustrated as follows: if *shops*(*john*) is not true, the spaghetti has been bought by Mary and if *shops*(*john*) is true, he must have bought spaghetti and there is a possibility that Mary has bought fish.

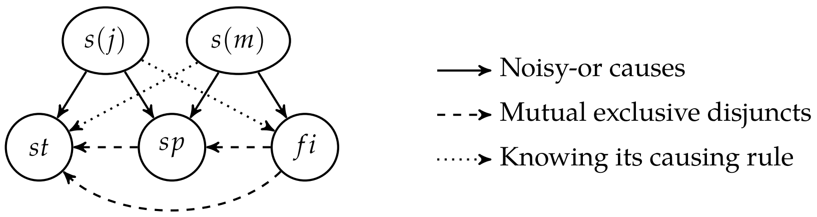


Figure 4.7: BN representation of the CP-theory from the shopping example. Edges are necessary for three different reasons: (a) connecting the head atoms in a rule with the body literals (noisy-OR), (b) connecting the head atoms of a rule (mutual exclusive disjuncts) and (c) connecting the head atoms of a rule with the body literals of another rule to know which rule caused which atom.

CP-theories with rules with at most one atom in the head can be represented with a regular BN with precisely one edge for each rule with a non-empty body. The structure of the conditional probability distributions of such networks is either noisy-OR (Section 4.2.1) or a combination of noisy-OR and deterministic AND (Section 4.2.2). Learning the parameters of this type of BN is intractable, unless dedicated techniques are used that take the special structure of the distributions into account. Such algorithms are known for simple networks with noisy-OR nodes (Heckerman and Brees 1994).

CP-theories with multiple atoms in the rule heads cannot be compactly represented with a regular BN over the domain atoms; the resulting networks tend to have many connections, which leads to large CPTs with much redundancy (Section 4.2.3). To avoid this, additional nodes are required that encode which head atom is selected. This is a function of the choice nodes in the EBN. More precisely, the choice nodes encode both the deterministic AND of the body literals and the mutual exclusivity of the head atoms.

From the two above observations it follows that regular BNs over the domain atoms in combination with traditional parameter learning techniques are not suitable in domains where the target theory satisfies the assumptions made by CP-theories. Either the resulting network will poorly approximate the target theory, or it will be overly complex, which undermines interpretability and makes parameter learning intractable. We conclude that the introduction of the choice nodes in the EBN is crucial to efficiently represent a CP-theory as a BN.

## 4.3 Knowledge Based Model Construction

Breese (1992) was among the first to note that the fixed models used in probabilistic graphical models like Bayesian networks have some disadvantages and a more dynamic concept of models is necessary. By using a representation that implicitly encodes an enormous number of possible model structures, some of which quite large, he initiated the combination of probability theory and relations.

Based on a particular query and information state at run time, he constructs a belief network that is custom-tailored, and that can be used in a probabilistic inference algorithm. The advantages of such an approach are that, first, general patterns can be stored in a database and assembled to apply to specific instances. For this we can use deductive reasoning techniques like SLD-resolution. Second, probabilistic reasoning can be reduced by using information about the context. Knowledge that is not relevant to the current query because it is blocked by evidence can be left out of the probabilistic reasoning phase. Third, a decision-theoretic model can be used to exclude parts of the model because of limited relevance.

Breese proposes a language that consists of three types of statements: (1) facts and rules represented with universally quantified Horn clauses to express the relations, (2) alternative outcomes to express possible worlds, and (3) probability distributions to quantify the probability of a possible world.

A Horn clause is a formula of the form

$$h \leftarrow b_1 \wedge b_2 \wedge \ldots \wedge b_n$$

where $h$ and $b_i$ are *positive* atoms and read as "$h$ if $b_1$ and $b_2$ …". These rules express the deterministic part of the model. The uncertain part is represented in terms of disjunctions, we know $a$ or $b$ or $c$ has occurred but are uncertain about the individual truth values of the components. Combinations of realizations of these possibilities define possible world states (i.e., the sample space in probability theory). It is typical for probabilistic models to need the notion of mutual exclusive choices, i.e., the case where exactly one outcome out of a set of possible outcomes is true. Breese offers a shorthand for such situations with the concept of *alternative outcome expressions*. For example $bought(\{spaghetti, fish, nothing\}, mary)$ tells us that Mary buys either spaghetti, fish, or nothing. The choices in the example are quantified by an expression representing a probabilistic dependency:

$$h \mid b_1 \wedge b_2 \wedge \ldots \wedge b_n = \Pr(\omega_h \mid \omega_{b_1} \wedge \ldots \wedge \omega_{b_n})$$

where $h$ is an alternative outcome expression, each $b_i$ is an atomic formula (possibly an alternative outcome expression), $\omega_h$ is the set of possible alternative outcomes, and Pr is a conditional probability distribution over the alternative outcomes of $h$ given the alternative outcomes for $b_1, \ldots, b_n$. We can quantify the buying habit of Mary as follows:

$$bought(\{spaghetti, fish, nothing\}, mary) \mid shops(\{false, true\}, mary)$$

$$= \Pr(\omega_{bought} \mid \omega_{shops})$$

=

| | *shops*(*false*,*mary*) | *shops*(*true*,*mary*) |
|---|---|---|
| *bought*(*spaghetti*,*mary*) | 0.0 | 0.5 |
| *bought*(*fish*,*mary*) | 0.0 | 0.5 |
| *bought*(*nothing*,*mary*) | 1.0 | 0.0 |

The representation Breese uses contains the powerful idea of combining first order deterministic relations and probabilistic statements. The structure he uses: a database with rules, possible outcomes of a variable and a quantification over these possible outcomes is the basis of many other formalisms that build further on this representation. The inference, however, is limited in several aspects. First there is no concept of combining functions. When multiple probabilistic dependencies unify with a particular subgoal a heuristic is used to select one. Second, there is no notion of logic concepts like negation or optimizations like tabling. Third, the probabilistic dependencies can only be expressed as simple conditional probabilities, there is not yet any notion of contextual independence. Fourth, recursive Horn clauses that result in directed cycles in the Bayesian network are not allowed. The probabilistic logic formalisms discussed in the following sections (and CP-logic) have been proposed to deal with these shortcomings.

With respect to CP-logic, the deterministic rules can be seen as a CP-event with one head atom and probability 1. The probabilistic statements are represented with one CP-event for every combination of the possible outcomes of all the variables. It is not possible to express every possible CP-theory in the representation of Breese without first reducing the theory to a Bayesian network.

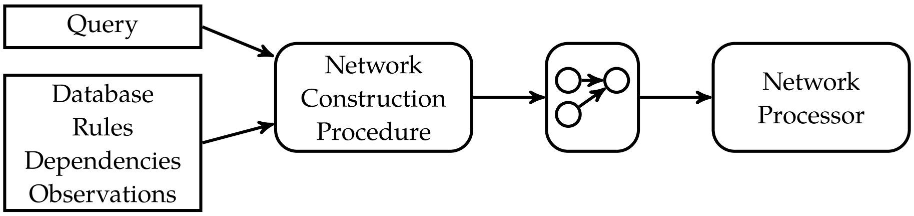


Figure 4.8: An overview of the network construction process as introduced in Breese (1992)

## 4.4 ICL

CP-logic is most closely related to the *Independent Choice Logic (ICL)* (Poole 1997), which uses a similar language. ICL is a probabilistic extension of abductive logic programming that extends the earlier formalism of *Probabilistic Horn Abduction* (Poole 1993).

An ICL theory consists of a logical and a probabilistic part. The logical part is an acyclic logic program, and the probabilistic part is a set of rules of the following form (in CP-logic syntax):

$$(h_1 : \alpha_1) \vee (h_2 : \alpha_2) \vee \ldots \vee (h_n : \alpha_n) \leftarrow .$$

Each atom $h_i$ may appear only once in the probabilistic part and only in the clause bodies of the logical part of the theory. This ensures that probabilities are associated to exactly one atom.

The syntax of ICL can be seen as a subset of the syntax of CP-logic, in which the CP-events are either deterministic or have an empty body (and for which the constraint on the $h_i$ holds). Also the semantics of both logics coincide for this subset, so every ICL theory can be read as a CP-theory. Therefore, the methods proposed in this dissertation may also be applicable to learn (ground) ICL theories by constraining the refinement operator used by the search so that it only generates syntactically valid ICL theories (see Chapter 7). To our knowledge, no learning methods have been published specifically for learning ICL theories.

Vennekens et al. (Vennekens 2007) shows that every acyclic CP-theory can also be transformed into an ICL theory by introducing one additional atom for each head atom of the CP-theory. Trying to directly learn such theories would be difficult because the learning algorithm would need to automatically introduce these atoms (which would correspond to invented predicates (Muggleton and Buntine 1988)). Therefore, an ICL learning method would not be directly applicable to learn CP-theories.

## 4.5 PRISM

*PRISM* (Sato and Kameya 2008) is similar to ICL, it also extends Prolog with a probabilistic part. For this, PRISM uses the probabilistic built-in *multi valued random switch* (msw) which is identical to the probabilistic part in ICL. A multi valued switch assigns a set of ground terms as the range of a discrete random variable. For every switch there is a probability distribution over the possible values (the range) a random variable can take.

A PRISM program thus consists out of two parts, $R$ and $F$. $R$ is a set of definite clauses whose head is not appearing in an msw atom. $F$ is a set of msw atoms together with a base distribution $\Pr_F$ defining probabilities (parameters) of msw atoms in $F$. The combination of both, $DB = R \cup F$, defines a probability measure $\Pr_{DB}(\cdot)$ over the set of Herbrand interpretations (*distribution semantics*).

**Example 4.2.** We can represent the choices John makes in the shopping with the following PRISM program:

```
values(shopsjohn,[john,nil]).
values(productsjohnbuys,[spaghetti,steak]).
shops(Who) :- msw(shopsjohn,Who).
bought(Product) :- msw(productsjohnbuys,Product), shops(john).
```

$\Diamond$

## 4.6 CHRiSM

Constraint Handling Rules (CHR) (Sneyers et al. 2010a; Frühwirth 2009) is a formalism that extends high-level languages with multi-headed rules. CHR has its roots in constraint solving but has evolved to a general purpose programming language. CHR is a language extension which is implemented on top of an existing programming language. For example, several CHR(Prolog) systems are available.

A CHR program consists of rules for simplification, propagation and simpagation (a combination of simplification and propagation) of conjunctions of constraints. The current set of constraints is kept in a multi-set: the CHR constraint store. Thus, CHR rules are constraint handling rules that refine the store of CHR terms according to some given constraints.

**Example 4.3.** Consider the following simplification rule which expresses that if a value is both smaller or equal and larger or equal than another value, they are the same value:

```
X <= Y , Y <= X <=> X=Y.
```

If the store contains the constraints `X <= Y` and `Y <= X` (where the logic variables `X` and `Y` can be unified with any constant) then they are removed from the store and replace with the new constraint `X=Y`. ◊

CHRiSM (Sneyers, Meert, and Vennekens 2009; Sneyers et al. 2010b) extends CHR with probabilities by building on PRISM as the host language. A CHRiSM program consists of a sequence of CHRiSM rules of the following form:

```
P ?? Hk \ Hr <=> G | B.
```

where `P` is a probability expression, `Hk` is a conjunction of constraints that are matched and kept in the store, `Hr` is a conjunction of constraints that are matched and removed from the store, `G` is a guard condition (a Prolog goal to be satisfied), and `B` is the body of the rule. Such a rule means that, whenever this rule could in principle be applied to resolve a CHR term, this will only happen with probability `P`. If `Hk` is empty, the rule is called a simplification rule and the backslash is omitted; if `Hr` is empty, the rule is called a propagation rule and it is written as "`P ?? Hk ==> G | B`". If both `Hk` and `Hr` are non-empty, the rule is called simpagation rule. The guard is optional. The body `B` is recursively defined as a conjunction of CHRiSM constraints and Prolog goals. It is also allowed to replace the body with a CP-logic like disjunct expressing a mutual exclusive choice. In that case the body is represented as "$p_1 : h_1 ; \ldots ; p_n : h_n$" with $p_i$ the probabilities and $h_i$ atoms and the probability in the beginning of the rule is dropped.

CHRiSM is useful for high-level rapid prototyping of complex statistical models by means of "chance rules". CHR has some advantages over Prolog, including complexity-wise completeness and the expressivity of multi-headed rules. It is expected that CHRiSM has the same advantages over plain PRISM.

**Example 4.4.** The following CHRiSM program implements the shopping example used before in Example 4.1. First, with only spaghetti to show the simple syntax:

```
go <=> shops(john), shops(mary).
0.5 ?? shops(john) <=> spaghetti.
0.3 ?? shops(mary) <=> spaghetti.
spaghetti, spaghetti <=> spaghetti.
```

We can add the last rule to express that we do not care about the number of times spaghetti is bought. This could be useful since CHRiSM keeps track of individual instantiations of an atom.

To extend the example to include also steak and fish we can use a CP-logic like disjunct in the body:

```
go <=> shops(john), shops(mary).
shops(john) <=> 0.5:spaghetti; 0.5:steak.
shops(mary) <=> 0.3:spaghetti; 0.7:fish.
spaghetti, spaghetti <=> spaghetti.
```

◊

We explain, informally, the semantics by giving the transformation from CP-logic to CHRiSM. Consider a CP-logic rule (or CP-event) $r$:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow \phi$$

with the $h_i$ propositions, the $\alpha_i$ probabilities (with $\sum \alpha_i = 1$) and $\phi$ a formula. Such a rule expresses that if the formula $\phi$ is satisfied, a random event will take place, which causes one of the $h_i$; each $\alpha_i$ represents the probabilities that the associated $h_i$ is the atom that is in fact caused.

We can translate such a rule to CHRiSM by introducing some new symbols. We introduce a symbol $\mathsf{b}_r$ to denote that the body $\phi$ of this rule $r$ is satisfied, and for each $h_i$ in its head, we introduce a symbol $\mathtt{c}_r^{h_i}$ to represent that $h_i$ is the atom caused by $r$. For all the symbols $s$ that we now have, we also introduce a symbol not_$s$ to represent its negation. We can then translate the above CP-logic rule as:

$$\mathtt{b}_r \texttt{ ==> } \alpha_1\texttt{:}\mathtt{c}_r^{h_1} \texttt{ ; ... ; } \alpha_n\texttt{:}\mathtt{c}_r^{h_n}$$

and

$$\texttt{not\_b}_r \texttt{ ==> } \texttt{not\_c}_r^{h_1}\texttt{,... , }\texttt{not\_c}_r^{h_n}.$$

In CP-logic, each rule can "fire" at most once. To avoid that the multiset semantics of CHR would give a different result, we add $\mathtt{b}_r$\ $\mathtt{b}_r$ <=> true to the top of the CHRiSM program.

To relate the $\mathtt{c}_r^{h_i}$ and $\texttt{not\_c}_r^{h_j}$, we use: $\mathtt{c}_r^{h_i}$ ==> $\texttt{not\_c}_r^{h_j}$, for all $i \neq j$.

The semantics of CP-logic has a non-monotonic aspect, which stems from its use of "negation-as-failure" as in the well-founded semantics for logic programs. This can be captured in CHRiSM by expressing that a atom $h$ is true if and only if it is caused by at least one rule $r_i$:

$$\begin{array}{c} \mathtt{c}_{r_1}^{h} \texttt{ ==> h} \\ \vdots \\ \mathtt{c}_{r_n}^{h} \texttt{ ==> h} \\ \texttt{not\_c}_{r_1}^{h}\texttt{,... , }\texttt{not\_c}_{r_n}^{h} \texttt{ ==> not\_h.} \end{array}$$

Here, $r_1, \ldots, r_n$ are all the rules that have $h$ in their head.

All that remains is to define the $\mathtt{b}_r$ and $\texttt{not\_b}_r$ in such a way that they correctly correspond to the truth of the bodies of the rules $r$. This requires us to encode the logical connectives in CHR. If we push negation down to the atom level, we can simply replace each $\neg h$ with not_h.

## 4.7 ProbLog

ProbLog (De Raedt, Kimmig, and Toivonen 2007) is an extension of Prolog with probabilities. Like in PRISM, a program consists of a probabilistic part and a logic part. The probabilistic part consists out of *probabilistic facts* that attach a probability to an atom. The logic part is a regular Prolog program without cuts. The probabilistic facts can only be used in the bodies of the Prolog rules in the logic part.

The language is similar to that of PRISM but PRISM avoids programs where inference is computational expensive by limiting the allowed programs to a specific subclass where all bodies are mutual exclusive. ProbLog on the other hand is designed with not only semantics but also with inference in mind. ProbLog allows bodies that are not mutual exclusive. This makes that different proofs for a query possibly share common subproofs and inference is not anymore straightforward and the inclusion-exclusion property needs to be taken into account. ProbLog uses an underlying transformation to BDDs to solve the disjoint-sum problem.

The main goal of ProbLog is to be an underlying, powerful, low-level language for probabilistic logic learning. Most probabilistic logic languages can be transformed into ProbLog and so can CP-logic (De Raedt et al. 2008).

ProbLog is based on the distribution semantics (Sato 1995) and uses a transformation to a binary decision diagram (BDD) to perform inference efficiently. ProbLog's inference engine works as follows. Given a query, it first computes all proofs of the query and collects these in a DNF formula. Next, it converts this formula to a BDD. Relying on this BDD representation, ProbLog then computes the query's probability in one bottom-up pass through the BDD (using dynamic programming).

**Example 4.5.** The shopping example (see Example 3.1) can be modelled with ProbLog as follows:

```
% Probabilistic facts
0.2::f_john_shops.
0.9::f_mary_shops.
0.5::f_john_buys_sp.
0.3::f_mary_buys_sp.
% Prolog rules
shops(john) :- f_john_shops.
shops(mary) :- f_mary_shops.
bought(spaghetti) :- shops(john), f_john_buys_sp.
bought(steak) :- shops(john), problog_not(f_john_buys_sp).
bought(spaghetti) :- shops(mary), f_mary_buys_sp.
bought(fish) :- shops(mary), problog_not(f_mary_buys_sp).
```

The meta-predicate `problog_not` is true when the probabilistic fact it has as argument is false. ◊

## 4.8 Bayesian Logic Programs

Bayesian Logic Programs (BLPs) (Kersting and De Raedt 2008) are one of the formalisms that are based on the *Knowledge Based Model Construction (KBMC)* (Breese, Goldman, and Wellman 1994) approach. They lift propositional Bayesian networks to the first-order case.

A BLP consists of a set of rules of the following form:

$$h_1 \mid b_1, \ldots, b_n.$$

with $h_i$ and $b_i$ atoms. Each rule has a conditional probability distribution (CPD) associated to it. For each predicate symbol, the BLP also defines a domain and a so-called combining rule.

The user must define a domain, which can be discrete or continuous, for each predicate symbol in the BLP. This then becomes the domain for all nodes in the BN that are built on this predicate symbol. This is a major difference with the equivalent BN of a CP-theory, in which all atom nodes need to be Boolean. In the following, we will therefore assume Boolean domains.

Given a query, the BLP is grounded and converted to a BN, and the answer to the query is then computed by applying BN inference techniques. The BN is created as follows. For each atom in the ground BLP, one node is created in the BN. For each clause in the ground BLP, all body atoms become parents of the head atom in the BN. The CPDs for the BN nodes are computed from the CPDs associated to the BLP clauses by applying the combining rules. This is necessary because multiple clauses in the ground BLP may have the same head atom. The combining rule then converts this multi-set of CPDs into one single CPD. Several authors propose techniques for learning BLPs (and similar formalisms) with combining rules, e.g., Kersting et al. (Kersting and De Raedt 2008), Natarajan et al. (Natarajan et al. 2006) and Jaeger (Jaeger 2007).

Because of BLP's strong roots in Bayesian networks, most of the results of the comparison between CP-theories and BNs from the previous section carry over to BLPs. BLPs may have more parameters that need to be learned, and CP-theories with multiple atoms in the rule heads cannot be compactly and intuitively represented with BLPs, unless additional predicates are introduced to represent the choice nodes. That is, the learning of such BLPs would require a type of *predicate invention* (Muggleton and Buntine 1988). A number of issues with BLPs have led to *Logical Bayesian networks* (Fierens et al. 2004; Fierens et al. 2006). The issues pointed out in this section are valid for both languages and therefore we only consider BLPs.

**Example: Noisy-OR.** BLPs can express noisy-OR structures elegantly by using the noisy-OR combining rule. This makes BLPs modular and compact. The representation is almost equal to the way this is expressed in CP-logic, as is illustrated in Fig. 4.9. For BLPs, however, it is important to specify the 1 and 0 parameter values in the CPTs; in CP-logic, these are implicit. This difference indicates that, even though the rules in both formalisms may look similar, there is a difference in semantics. We elaborate further on this difference in the next example.

**Example: Multiple literals in the rule bodies.** Suppose that we have a CP-theory in which one CP-event has multiple literals in the body (see Figure 4.4):

$$x : \alpha \leftarrow q, y.$$
$$x : \beta \leftarrow z.$$

We could represent this theory as the following BLP:

$$x \mid q, y.$$
$$x \mid z.$$

with noisy-OR as combining rule for predicate $x$. This, however, would require a CPT with $dom(q) \cdot dom(y) = 2 \cdot 2 = 4$ parameters for the first clause compared to one parameter in CP-logic. This can be avoided by introducing a new predicate $a$, and by using the noisy-OR combining rule for $x$ and the AND combining rule for $a$, as is assumed in the following BLP:

$$x \mid a. \qquad\qquad a \mid q.$$
$$x \mid z. \qquad\qquad a \mid y.$$

Although the CPTs are now small, an extra predicate needs to be added. Adding new predicates during learning, is not trivial and is further investigated in Chapter 7.

**Example: Multiple atoms in the rule head.** When we try to express the knowledge represented by a CP-event with multiple head atoms (see Figure 4.5) with a BLP, we encounter the same problem as for BNs (although the BLP may have fewer parameters because of the combining rules). Just like in the BN of Fig. 4.7, it is necessary to add extra (confusing) dependencies when representing the CP-event as a BLP.

$$x \mid y_1$$
$$x \mid y_2$$

(a) BLP: Structure.

| $x$ | $y_1$ | $\neg y_1$ |
|---|---|---|
| t | $\alpha_1$ | 0 |
| f | $\bar{\alpha_1}$ | 1 |

| $x$ | $y_2$ | $\neg y_2$ |
|---|---|---|
| t | $\alpha_2$ | 0 |
| f | $\bar{\alpha_2}$ | 1 |

(b) BLP: CPDs (with $\bar{\alpha} = 1 - \alpha$).

$$x : \alpha_1 \leftarrow y_1$$
$$x : \alpha_2 \leftarrow y_2$$

(c) CP-theory.

Figure 4.9: A BLP with a noisy-OR combining rule and the equivalent CP-theory, both with the same parameters (the combining rule specification is not shown).

## 4.9 Relational Bayesian networks

Relational Bayesian Networks (RBNs) (Jaeger 1997) combine a relational representation with Bayesian networks. The random variables in the Bayesian network are predicate symbols and the states of these random variables are the possible interpretations of the symbols over an arbitrary, finite domain. An RBN can be compiled into a network that is similar to that generated by a BLP. The nodes correspond to domain atoms instead of predicate symbols. RBNs are modelled with the underlying Bayesian network in mind and have similar difficulties, e.g., cyclic relations are difficult to express. Because RBNs are a relational lifting of Bayesian networks like BLPs are, the conclusions from Section 4.8 carry over and we do not discuss RBNs in depth. For a more detailed comparison between RBNs and CP-logic, see Vennekens, Denecker, and Bruynooghe (2009a).

## 4.10 Other

In this section, we give an overview of some other probabilistic logic formalisms.

CPT-L (Thon, Landwehr, and De Raedt 2008) is based on and a subset of CP-logic. Its main goal is to make the implicit notion of time in a CP-theory explicit (see Section 3.8). Inference and learning for this subset of CP-logic can be done efficiently and makes CPT-L therefore suited for dynamic domains like a relational stochastic process.

*Stochastic Logic Programs* (Muggleton 2000) are a probabilistic extension of Prolog like ProbLog but probabilities are now attached to the selection of clauses in Prolog's SLD-resolution algorithm. This can be seen as a first-order version of stochastic context free grammars. It is difficult to relate stochastic logic programs to CP-logic because they represent a different type of probability that can be associated with first order logics. This difference is formalized in Halpern (1990) and introduces two types of probabilistic structures. A structure of type I represents the *degree of belief* of an agent, e.g., Bayesian networks and CP-logic. A structure of type II represents statistical knowledge like, e.g., the probability that a randomly chosen object has some property. Stochastic logic programs represent this second type.

Less related are MLNs (Richardson and Domingos 2006), they cannot be categorized as directed probabilistic logic models but are rather *undirected probabilistic logic models*. The statements in an MLN have no implicit sense of direction attached to them like in, e.g., CP-logic, Bayesian networks. An MLN consists out of first-order logic formulas that have weight attached to them. An MLN can be transformed into a Markov random field that can be used to perform inference.

## 4.11 Conclusions

In this chapter, we have given an overview of some of the many probabilistic (logic) languages. Because of the large number of formalisms, it is difficult to describe them all. Therefore, we summarize this chapter by given an overview of features where probabilistic logic languages typically differ.

- What happens if you add multiple rules defining the same ground atom (thus in the head of rules in a logic programs)? Such a situation is present in the spaghetti example where two different CP-events can cause the buying of spaghetti. This is for example also the case when there is a logic variable that only appears in the body of a CP-event and CP-logic handles such a situation implicitly with a noisy-or. The same is true for PRISM and ProbLog. BLPs, LBNs and RBNs demand the model to specify a method to combine multiple rules (a so-called *combining rule*), this can be for example noisy-or, noisy-and, or others. KBMC on the other hand cannot deal with this situation.

- Is negation of atoms allowed, and how is it handled? CP-logic allows negation by means of the temporal precedence property. Other languages such as ProbLog use the closed world assumption.

- Is recursion allowed? Or otherwise stated, are cycles allowed? CP-logic allows cycles and thus any type of recursion. The formalisms based on Bayesian networks like KBMC, BLPs, LBNs, RBNs only allow theories that can be stratified since these result in acyclic Bayesian networks. PRISM and ProbLog allow all types of recursion but are not well-defined when the recursion causes an infinite proof. The cycles, however, can be broken by means of tabling and this method turns out to be similar to the unfolding of cycles in CP-logic[2].

- Expresses the formalism directed or undirected probabilistic dependencies (or a mixture)? CP-logic and the methods described in detail in this chapter are all directed since they have a notion of head and body (or children and parents). MLNs on the other hand are the prototypical undirected probabilistic logic formalism. Like for the probabilistic models, directed models deal with probabilities directly and undirected with weights.

- What do the probabilistic parameters express? In a Bayesian network the parameters are conditional probabilities but for CP-logic they are causal probabilities. All the formalisms based on Bayesian networks use conditional probabilities. PRISM and ProbLog uses yet another type

[2]Based on private communication about unpublished work with the developers of ProbLog.

of parameters since they represent independent probabilities that are distributed over the logic (non-probabilistic) atoms.

# 5 First-order Bayes-ball

## 5.1 Introduction

Inference is a bottleneck in probabilistic logic learning, affecting also the cost of learning these models from data. Many attempts have been made to make inference more efficient in these formalisms, including lifted inference methods which try to exploit the symmetries present in the first-order probabilistic model (Poole 2003; Salvo Braz, Amir, and Roth 2005; Milch et al. 2008; Singla and Domingos 2008; Kisynski 2010). In this chapter, we make use of the first-order structure to identify random variables (ir)relevant to the inference task at hand. To answer a specific probabilistic query, there is a minimum set of random variables which are required to be included in the computations. This set is called the *minimal requisite network* (MRN) (Shachter 1999). Inference becomes more efficient by restricting computations to the MRN.

An added benefit of the MRN is that it allows us to reason about the model on a local level. Since the MRN represents a submodel containing all the variables that influence the query, we can reason about the interactions in the model while ignoring a possibly large part of the model that does not influence the query.

Bayes-ball (Shachter 1999) is an efficient algorithm that finds the MRN for inference in (propositional) Bayesian networks. The naive way of applying Bayes-ball to a probabilistic logic model is to ground the entire model and apply Bayes-ball on the resulting propositional network. This can be computationally expensive or impossible because the grounded network is in general large for real world models, and its construction can be a significant part of the total inference cost.

Another way to compute the MRN for a probabilistic logic model consists of two steps: In the first step, all logic programming proofs for the query and for all evidence atoms are computed (e.g., using SLD resolution (Kersting and De Raedt 2000)), and a network is built using all ground events that are used therein. In the second step, the MRN is computed by applying Bayes-ball to this network. The second step is necessary since some atoms encountered in certain proofs of an evidence atom may be D-separated (Shachter 1999) from the query. This method has the disadvantage that it initially computes a Bayesian network that may be larger than the MRN. Breese (1992) proposed a method that combines SLD resolution with searching for evidence like in Bayes-ball. This method is not directly applicable for CP-logic because the formalism used by Breese is much more restricted than CP-logic (see Section 4.3).

None of the mentioned methods take full advantage of the first-order representation of the probabilistic logic model. First-order probabilistic models introduce many random variables which are essentially identical with respect to inference, and hence also share the same status of relevance for a specific probabilistic query. We propose a first-order version of the Bayes-ball algorithm called first-order Bayes-ball (FOBB) that exploits these symmetries to efficiently compute the MRN for probabilistic logic inference. FOBB works directly at the first-order level while building the MRN; there is no need to build the ground network in the beginning. This algorithm treats indistinguishable random variables as one first-order atom and can process them in one single step (performing identical operations only once).

Another advantage of FOBB is in the *first-order representation of the MRN*. This is valuable for lifted inference algorithms, which are designed specifically to make use of the first-order representation. The existing methods for computing the MRN all produce a ground MRN which does not preserve the first-order structure of the original model and hence destroys the possibility of using lifted inference. FOBB, on the other hand, produces a first-order MRN for each query. This network can be used by lifted inference algorithms and can also make lifted inference more efficient by removing unnecessary but costly operations on the irrelevant parts of the model (such as shattering (Milch et al. 2008; Kisynski and Poole 2009a)). To the best of our knowledge the issue of relevance has not been addressed in the lifted inference settings before.

Although we illustrate FOBB for CP-logic, FOBB is a general method that applies to several directed probabilistic logic formalisms such as ProbLog (De Raedt, Kimmig, and Toivonen 2007), BLPs (Kersting and De Raedt 2007) and ICL (Poole 1997).

### 5.1.1 Bibliographical note

We first introduced the concept of First-Order Bayes Ball for CP-logic in the following paper

> N. Taghipour et al. (2009). "First-order Bayes-ball for CP-logic". In: *Proceedings ofthe International Workshop on Statistical Relational Learning (SRL)*. (Leuven, Belgium, July 2–4, 2009), pages 1–3.

The author, together with N. Taghipour, devised the FOBB algorithm for CP-logic but N. Taghipour was the main responsable for the text (the paper has shared first authorship). We later refined the algorithm and made it more general to be applicable to all directed probabilistic logic models. This work was presented in

> W. Meert, N. Taghipour, and H. Blockeel (2010). "First-order Bayes-ball". In: *Proceedings ofthe 21th European Conference on Machine Learning (ECML)*. (Barcelona, Spain, Sept. 20–23, 2010). Volume 6322. Lecture Notes in Computer Science, pages 369–384.

Both the author and N. Taghipour were involved with implementation, experiments and text, but the author was the main contributor for the implementation and the experiments, and N. Taghipour was the main contributor for the text (again shared first authorship).

### 5.1.2 Structure of this chapter

In Section 5.2 we review the Bayes-ball algorithm. Section 5.3 introduces the FOBB algorithm. Experiments are presented in Section 5.5. Section 5.6 contains the conclusions.

## 5.2 Preliminaries

### 5.2.1 Bayes-Ball

Bayes-ball (Shachter 1999) identifies the MRN of a Bayesian network for a given set of query and evidence nodes. It is based on the analogy of bouncing balls that travel over the edges of the Bayesian network (Fig. 5.1). In this analogy a visit from node *a* to *b* is compared to passing a ball from *a* to *b*. The balls start at the query nodes (each query node receives a ball). Upon reaching each node, a ball may *pass through*, *bounce back* and/or be *blocked*. The action chosen depends on the direction from which it came and on whether the node is probabilistically or deterministically dependent on its parents (based on D-separation). When a ball is to be passed to multiple nodes, it means that a visit to each of those nodes is put in a schedule. At each step a visit is selected from the schedule and processed.

The rules by which the balls moves through the network can be summarized as follows:

(a) An unobserved probabilistic node passes balls from parents on, that is, if such a node receives the ball from a parent it passes the ball to all its children. When such a node receives the ball from a child, it passes the ball to both all its parents and children (including the node the ball came from).

(b) A unobserved deterministic node always passes balls through, that is, it passes the ball coming from parents to children and vice versa.

(c) An observed node bounces balls back from parents, that is, upon receiving the ball from a parent, such a node passes the ball to all its parents. However, an observed node blocks balls from children, that is, the ball is not passed on anymore from this node.

Nodes are marked at each visit of a ball, depending on the type of action performed on the ball: when the ball is passed from a node to its parents (children), the node receives a mark on top (bottom). These marks help the algorithm avoid repeating the same action, and guarantee the termination of the algorithm. Having a mark on top (bottom) indicates that there is no need to visit the parents (children) anymore. In the end, these marks indicate the relevance of each node: the MRN consists of all the nodes marked on the top together with the set of evidence atoms visited during the algorithm.

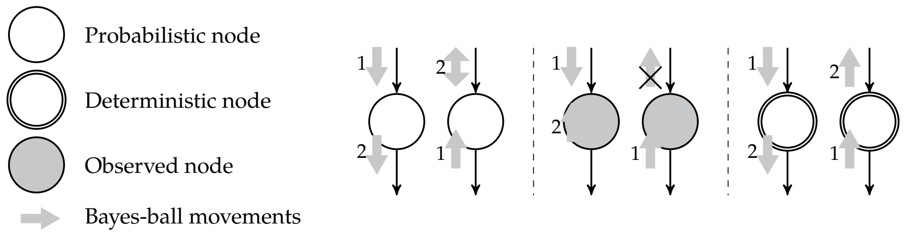


Figure 5.1: Different actions performed on the Bayes-ball depending on the type of the node and direction from which the ball comes. The numbers indicate the sequence of actions.

## 5.2.2 CP-logic with types and constraints

The basic FOBB algorithm we are presenting is meant for a typed version of CP-logic: CP-logic$^t$. Extending FOBB for regular CP-logic is discussed in Section 5.4. In CP-logic$^t$ each predicate $p$ takes on values from a specific $Range(p)$, which contains the possible states of the random variables represented by this predicate (in this thesis, we only consider predicates with range $\{true, false\}$). Moreover, each argument of a predicate takes values of a specific type. A typed CP-theory is extended with *domain declarations* defining the set of constants of each type, and *functor declarations* assigning a type to each argument of each predicate.

A typed CP-theory captures the structure of the probabilistic model through a set of typed CP-events. A typed CP-event $c_i$ is an expression of the form:

$$(c_i)\ \forall X_1,\ldots,X_l : C;\ h_1 : p_1 \vee \ldots \vee h_m : p_m \leftarrow b_1,\ldots,b_n$$

with $h_1,\ldots,h_m$ atoms and $b_1,\ldots,b_n$ literals with logic variables $X_1,\ldots,X_l$ as arguments. Each logic variable $X_i$ is implicitly assigned to a domain, considering the *functor declarations*. The constraint $C$ restricts all possible groundings of $X_1,\ldots,X_n$ to a subset of the Cartesian product of their respective domains. We define $head(c_i) = \bigcup_i h_i$ and $body(c_i) = \bigcup_i b_i$. A typed CP-events specifies that for each substitution $\theta = \{X_1/t_1,\ldots,X_n/t_n\}$ that grounds the event and is in accordance with $C$, the random variables $h_1\theta,\ldots h_m\theta$ depend on $b_1\theta,\ldots,b_n\theta$.

### 5.2.3 Parameterized Bayesian networks

To illustrate FOBB we represent intermediate steps graphically as a *parameterized Bayesian network* (PBN) (Poole 2003). Parameterized Bayesian networks are an extension of Bayesian networks to incorperate logic variables. Just like we can transform a ground CP-theory to a Bayesian network (see Section 3.6), we can transform a non-ground CP-theory to a parameterized Bayesian network. In such a model, random variables of a Bayesian network are represented by (ground) atoms, and definite clauses are used to capture the structure of the network. In this way, a first-order atom represents a class of random variables, and a first-order clause represents repeating structures in the Bayesian network. An example PBN is presented in Fig. 5.3.

Similar as to CP-logic$^t$, each predicate $p$ takes on values from a specific range $Range(p)$, which contains the possible states of the random variables represented by this predicate. Moreover, each argument of a predicate takes values of a specific type. A PBN includes *domain declarations* defining the set of constants of each type, and *functor declarations* assigning a type to each argument of each predicate.

A PBN also consists of a *theory*, which captures the structure of the probabilistic model through a set of *Bayesian clauses*. A Bayesian clause $bc$ is an expression of the form: [1]

$$(bc)\ \forall X_1,\ldots,X_n : C;\ h|b_1,\ldots,b_n$$

[1]We use a slightly different version of Poole's PBN (Poole 2003): here each rule is associated with an entire CPD, instead of declaring the probability of one specific combination of values for a node and its parents. This idea is introduced in BLPs (Kersting and De Raedt 2007), from which we also borrow the term *Bayesian clause*.

with $h, b_1, \ldots, b_n$ first-order atoms with logic variables $X_1, \ldots, X_n$ as arguments. Each logic variable $X_i$ is implicitly assigned to a domain, considering the *functor declarations*. The constraint $C$ restricts all possible groundings of $X_1, \ldots, X_n$ to a subset of the Cartesian product of their respective domains. We define $head(bc) = h$ and $body(bc) = \bigcup_i b_i$. A Bayesian clause specifies that for each substitution $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ that grounds the clause and is in accordance with $C$, the random variable $h\theta$ depends on $b_1\theta, \ldots, b_n\theta$.

Each Bayesian clause is associated with a conditional probability distribution (CPD) $P(h|b_1, \ldots, b_n)$, which specifies the same distribution $P(h\theta|b_1\theta, \ldots, b_n\theta)$ for every ground instance of the clause. If a ground atom appears in the head of more than one ground clause, then a *combining rule* is used to obtain the desired probability distribution from the CPDs associated to those clauses. Each predicate has an associated combining rule.

Where a Bayesian network represents a factorization into factors, a parameterized Bayesian network is a factorization into *parfactors* (Poole 2003). Parfactors are a data structure used in first-order probabilistic models to perform inference analogous to factors during inference in belief networks (Kisynski 2010). They represent the conditional probability distributions in directed first-order models and intermediate computation results during lifted inference (and can also be potentials in undirected first-order models). A parfactor consists of a set of first-order atoms (representing the random variables), a set of constraints on the logic variables in the atoms, and a factor from the Cartesian product of ranges of the predicates to the reals.

### 5.2.4 Equivalent Bayesian Network

Like normal CP-theories, a typed CP-theory can be transformed to a (parameterized) Bayesian network. This transformation to what we called an equivalent Bayesian network (EBN) is explained in Section 3.6. We recall the transformation briefly. The EBN is a regular Bayesian network, but we consider it as a bipartite graph containing two types of nodes: *atom nodes* and *choice nodes* (see Fig. 5.2.b).

Atom nodes correspond to the random variables; there is an atom node in the EBN for each atom in the grounding of the theory.

Choice nodes explicitly capture the factorization declared by the CP-event. For each CP-event $c_i$ and each grounding substitution $\theta$ complying with the constraint $C$ of $c_i$, there is a choice node corresponding to $c_i\theta$ in the EBN. We denote choice nodes by atoms too. Each CP-event $c_i$ is associated to an $n$-ary predicate $c_i$, with $n = |Var(c_i)|$ (we use the same name since it is clear from the context whether we mean the CP-event or the predicate). This predicate has the same domain as $head(c_i)$. The choice node corresponding to ground event $c_i\theta$ can be represented by the atom $c_i\theta$.

There are edges between choice nodes and atom nodes, but no edges between nodes of the same type. The edges of the network can be fully described with this rule: Each choice node $c_i\theta$, corresponding to a ground CP-event $c_i\theta$, has as parents all the atoms in $body(c_i\theta)$, and is the parent of the atoms $head(c_i\theta)$.

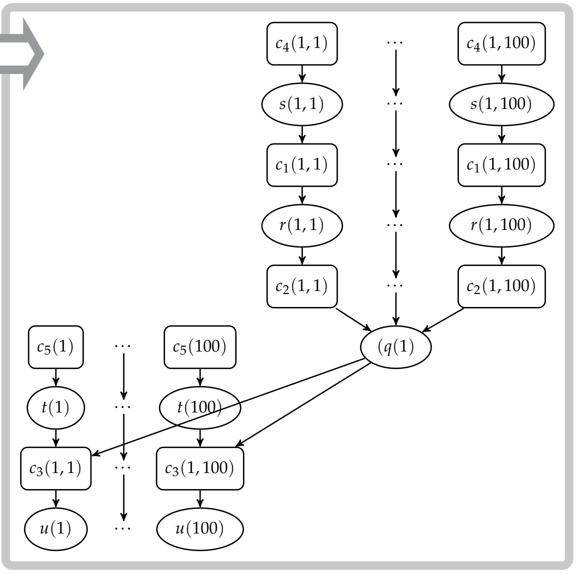
**Domains**
$Num = \{1, \dots, 100\}$

**Functor Declarations**
$q(Num)$
$r(Num, Num)$
$s(Num, Num)$
$t(Num)$
$u(Num)$

**Theory**
$(c_1)\ \forall X, Y;\ r(X,Y) : p_1 \leftarrow s(X,Y).$
$(c_2)\ \forall X, Y;\ q(X) : p_2 \leftarrow r(X,Y).$
$(c_3)\ \forall X, Y;\ u(X,Y) : p_3 \leftarrow q(X), t(Y).$

$(c_4)\ \forall X, Y : \{X = 1\};\ s(X,Y) : p_4.$
$(c_5)\ \forall X;\ t(X) : p_5.$

a



b

Figure 5.2: (a) Typed CP-theory (b) Equivalent Bayesian network having the same probability distribution as the theory. Bayes-ball can be used on such a Bayesian network to find the MRN given a query and evidence.

## 5.3 First-Order Bayes-Ball

FOBB is based on the same principles as Bayes-ball, building upon the transformability of a probabilistic logic model to an EBN. Its main advantage is the possibility to perform some steps at the first-order level. That is, several nodes can be represented by what we call a *first-order node*, and be visited in one single step. After a definition of first-order nodes and related operations on such nodes, we show the main features of the algorithm through an example; this is followed by a more detailed description.

**Definition 5.1. (Constraint)** Having logic variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, with $D(X_i)$ the associated domain of $X_i$, a constraint $C$ on $\mathbf{X}$ is a relation on $\mathbf{X}$, indicating a subset of the Cartesian product $D(\mathbf{X}) = \times_i D(X_i)$.

**Definition 5.2. (First-order node)** A first order node $\mathcal{F}$ is a pair $(p, C)$, where $p = a(X_1, \ldots, X_n)$ is a first-order atom, and $C$ is a constraint on logic variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. Each first-order node $\mathcal{F} = (p, C)$ represents the set of ground random variables $p\theta$, where $\mathbf{X}\theta \in C$. We denote the set of (ground) random variables represented by $\mathcal{F}$ as $RV(\mathcal{F})$.

For two first-order nodes $\mathcal{F}_1 = (p, C_1)$ and $\mathcal{F}_2 = (p, C_2)$, we define:

1. $\mathcal{F}_1 \subseteq \mathcal{F}_2$ iff $RV(\mathcal{F}_1) \subseteq RV(\mathcal{F}_2)$ iff $C_1 \subseteq C_2$.
2. $\mathcal{F}_1 \Delta \mathcal{F}_2 = \mathcal{F}'$ iff $\mathcal{F}' = (p, C_1 \Delta C_2)$, for $\Delta \in \{\cap, \cup, \setminus\}$.

Note that, as we have defined constraints as relations on $X_i$ (which are sets), we use set operators $\subseteq, \cap, \cup, \setminus$. Intuitively, when constraints are seen as conditions, these operators correspond to implication, conjuction, disjunction and "and not".

**Definition 5.3. (Splitting)** The result of splitting a first-order node $\mathcal{F} = (p, C)$ is a set of first-order nodes $\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, where each $\mathcal{F}_i = (p, C_i)$, and such that $\bigcup_i C_i = C$ and $\forall i, j \neq i : C_i \cap C_j = \emptyset$.

FOBB also uses the operation of *projection* on constraints:

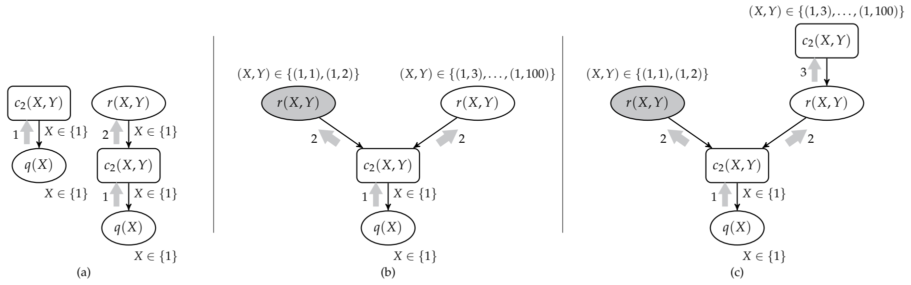
**Definition 5.4. (Projection)** Let $C$ be a constraint on logic variables $\mathbf{X}$. Projection of $C$ on a subset of its variables $\mathbf{Y} \subseteq \mathbf{X}$ is given by the constraint $\pi_Y(C) = \{\mathbf{y} = (y_1, \ldots, y_{|Y|}) | \exists \overline{\mathbf{y}} \in C$, and $\overline{\mathbf{y}}$ is an extension of $\mathbf{y}\}$.

There is no restriction on how to represent and store the constraints. The choice of representation, however, affects the efficiency of the algorithm. For example, storing them as ground tuples would cancel the advantages of FOBB over

Bayes-ball. For the implementation we opted to store constraints as decision trees with set membership tests in the nodes. This representation is different from that used in (Poole 2003) for PBNs and other work about lifted inference, where a constraint is a set of (in)equalities involving logic variables and constants. One such conjunction is equivalent to one branch in our decision tree.

## 5.3.1 Overview

FOBB computes the MRN for a probabilistic query $P(\mathbf{q}|\mathbf{e})$ on the probabilistic logic model $M$. It is assumed that the query atoms $\mathbf{q}$ are ground and that the theory $T$ of $M$ has a finite grounding. The outer structure of FOBB closely resembles the original Bayes-ball (see (Shachter 1999)), the main differences are that it works with first-order nodes instead of ground nodes, and that it uses the given first-order probabilistic logic model to compute the parents and children of a first-order node.



$c_2(X,Y)$
$q(X)$
$r(X,Y)$
$X \in \{1\}$
1
2
3
$(X,Y) \in \{(1,1),(1,2)\}$
$(X,Y) \in \{(1,3),\ldots,(1,100)\}$
(a)
(b)
(c)

Figure 5.3: Illustration of the FOBB algorithm as explained in Sec. 5.3.1. (a) A ball is passed on from a node to its ancestors. (b) When not all ground nodes represented by a first-order node respond identically to the ball, the node is split up. In this case, part of the nodes represented by the first-order node are observed and do not pass on the ball. (c) The first-order node that represents the ground nodes that are not observed and do not yet have top mark passes on the ball to its ancestors.

We use an example to illustrate FOBB and compare it with the original Bayes-ball: Suppose we need to compute $P(\mathbf{q}|\mathbf{e})$ where $\mathbf{q} = \{q(1)\}$ and $\mathbf{e} = \{r(1,1), r(1,2), u(1,1), \ldots, u(1,10)\}$, given the theory in Fig. 5.2.

Similar to Bayes-ball, FOBB schedules the nodes which are to be visited. Instead of scheduling ground nodes to visit, FOBB schedules first-order nodes. Each entry in the schedule is represented by a tuple $\langle \mathcal{F}, direction \rangle$, containing a first-order node $\mathcal{F}$, and the *direction* of the visit ($fromChild$ or $fromParent$). This entry stands for a visit to each node in $RV(\mathcal{F})$, in Bayes-ball. In the beginning, FOBB starts by scheduling the query: $\langle (q(X), \{X{=}1\}), \mathit{fromChild} \rangle$ stating the ball comes from a child such that the parents are visited first. Next, FOBB retrieves this tuple from the schedule and computes its parents by matching $(q(X), \{X{=}1\})$ to the heads of events of $T$. In this case, only CP-event $c_2$ matches, so FOBB schedules the choice node $\langle (c_2(X,Y), \{X{=}1\}), \mathit{fromChild} \rangle$. The constraint $\{X{=}1\}$ makes sure that only the subset of $c_2(X,Y)$ that are the parents of the query, are included. We will elaborate on this later. Note that the scheduled first-order node actually represents multiple nodes in the EBN (due to the free variable $Y$). This is shown in the first step in Fig. 5.3.

When a subset $\mathcal{F}'$ of the nodes in $\mathcal{F}$ interact differently with the rest of the network (e.g. they are observed variables), we need to separate them from other nodes in $\mathcal{F}$. We call this operation *splitting*, following Poole (Poole 2003). Continuing our example, when $\mathcal{F} = (r(X,Y), \{X{=}1\})$ receives the ball (from child $(c_2(X,Y), \{X{=}1\})$) it contains nodes $r(1,1)$,$r(1,2)$ which are evidence while the rest of the nodes are not observed. Hence, the algorithm splits the original first-order node $\mathcal{F}$ into $\mathcal{F}_{\neg e} = (r(X,Y), \{X = 1, Y \notin \{1,2\}\})$, consisting of unobserved nodes, and $\mathcal{F}_e = (r(X,Y), \{X = 1, Y \in \{1,2\}\})$, consisting of the evidence. Now, when $\mathcal{F}_{\neg e}$ receives the ball it passes the ball to its parents and children, while $\mathcal{F}_e$, which contains only observed nodes, blocks the ball. The splitting operation will be defined in detail later.

Passing the ball to the children involves similar operations as sending the ball to parents. For example, to find children of $(q(X), \{X = 1\})$, we need to find events which have this atom in their body. We see it only appears in the body of event $c_3$, and so its children would be $(c_3(X,Y), \{X = 1\})$.

Bayes-ball assigns *top* and/or *bottom* marks to the nodes it visits. FOBB also keeps track of marks, but here these are marks for first-order nodes. It stores the marks as pairs $\langle \mathcal{F}, \mathbf{M} \rangle$ in a mark table, with $\mathcal{F}$ being a first-order node, and $\mathbf{M}$ a set of marks. For example, the initial marks for the query atom and for $c_1(X,Y)$ are stored as $\langle (q(X), \{X = 1\}), \{top\} \rangle$ and $\langle (c_1(X,Y), \{X{=}1\}), \{top\} \rangle$, showing these two first-order nodes have passed the ball to their respective parents.

During the execution of the algorithm, it is possible that we have to split an atom in the marks table. This happens when a subset of the nodes presented by a first-order node need to be assigned additional marks. In our example, $\mathcal{F}_1 = (c_3(X,Y), \{X = 1\})$ passes the ball to its children resulting in $\langle \mathcal{F}_1, \{bottom\} \rangle$ being registered in the marks table. Later when the ball is passed up from $(u(X,Y), \{X = 1, Y \in \{1,\ldots,10\}\})$ to $\mathcal{F}_1' = (c_3(X,Y), \{X = 1, Y \in \{1,\ldots,10\}\})$ this node should in turn pass the ball up and receive a mark on top. For this reason we need to first split $\mathcal{F}_1$, keeping $\langle \mathcal{F}_1 \setminus \mathcal{F}_1', \{bottom\} \rangle$ from the original first-order node and adding $\langle \mathcal{F}_1', \{top, bottom\} \rangle$ to the marks table, for the subset which passes the ball up.

When FOBB terminates, all the first-order *choice* nodes marked with $top$, together with the visited evidence atoms constitute the MRN.

## 5.3.2 The Algorithm

The FOBB algorithm uses the same set of rules as Bayes-ball to send balls through the network, visiting the possibly relevant nodes. The main difference is that the balls are not passed between nodes as we know them from Bayesian networks but between first-order nodes that aggregate multiple ground nodes in one higher level node. This way FOBB can perform multiple identical operations on $RV(\mathcal{F})$ in one single step instead of performing $|RV(\mathcal{F})|$ equivalent steps in Bayes-ball.

FOBB schedules visits to a group of ground nodes aggregated in a first-order node, searches for parents and children of such a first-order node and assigns marks to first-order nodes. The aim is to keep the nodes as aggregated as possible, but when a subset of the nodes behave differently it is necessary to split the first-order node and treat those subsets separately. The splitting happens when needed during the execution of the algorithm. Next, we illustrate how the operations in FOBB differ from those in Bayes-ball (see also Fig. 5.4:

#### Initialization

In the initialization of the algorithm, all the query atoms are added to the schedule as if they were visited from child. This makes that the parents of the query atoms are first visited and since a ball should bounce back on at least one of the ancestors and be propagated back to the query atoms, the children of the query atoms are also visited. For this we need to represent the query nodes as first-order nodes. This is done by the GetFONode method that takes

**Algorithm 3** FOBB($M$, $\mathbf{q}$, $\mathbf{e}$)

```
**Input:** $M$: probabilistic logic model, $\mathbf{q}$: set of ground query atoms, $\mathbf{e}$: set of ground evidence atoms
**Output:** $R$: requisite network, $e_R$: requisite evidence

$S \leftarrow \emptyset, e_R \leftarrow \emptyset$
for each $q \in \mathbf{q}$ do
  $\mathcal{Q} = \textsc{GetFONode}(q); S \leftarrow S \cup \langle \mathcal{Q}, fromChild \rangle$
while $S \neq \emptyset$ do
  pick and remove a visit $\langle \mathcal{F}, direction \rangle$ from $S$
  $(\mathcal{F}_e, \mathcal{F}_{\neg e}) \leftarrow \textsc{SplitOnEvidence}(\mathcal{F}, \mathbf{e})$
  if $\mathcal{F}_e \neq \emptyset$ then
    $e_R \leftarrow e_R \cup \mathcal{F}_e$
  if $direction = fromChild \wedge \mathcal{F}_{\neg e} \neq \emptyset$ then            // Backward chaining
    $(\mathcal{F}_{\neg e}^{top}, \mathcal{F}_{\neg e}^{\neg top}) \leftarrow \textsc{SplitOnMark}(\mathcal{F}_{\neg e}, top)$
    $\textsc{AddMark}(\mathcal{F}_{\neg e}^{\neg top}, top)$
    for each $\mathcal{PA} \in \textsc{GetParents}(\mathcal{F}_{\neg e}^{\neg top}, M)$ do
      $S \leftarrow S \cup \langle \mathcal{PA}, fromChild \rangle$
    $(\mathcal{F}_{\neg e}^{btm}, \mathcal{F}_{\neg e}^{\neg btm}) \leftarrow \textsc{SplitOnMark}(\mathcal{F}_{\neg e}, bottom)$
    if $\neg$Functional$(\mathcal{F}) \wedge \mathcal{F}_{\neg e}^{\neg btm} \neq \emptyset$ then
      $\textsc{AddMark}(\mathcal{F}_{\neg e}^{\neg btm}, bottom)$
      for each $\mathcal{CH} \in \textsc{GetChildren}(\mathcal{F}_{\neg e}^{\neg btm}, M)$ do
        $S \leftarrow S \cup \langle \mathcal{CH}, fromParent \rangle$
  if $direction = fromParent$ then            // Forward chaining
    if $\mathcal{F}_e \neq \emptyset$ then
      $e_R \leftarrow e_R \cup \mathcal{F}_e$
      $(\mathcal{F}_e^{top}, \mathcal{F}_e^{\neg top}) \leftarrow \textsc{SplitOnMark}(\mathcal{F}_e, top)$
      $\textsc{AddMark}(\mathcal{F}_e^{\neg top}, top)$
      for each $\mathcal{PA} \in \textsc{GetParents}(\mathcal{F}_e^{\neg top}, M)$ do
        $S \leftarrow S \cup \langle \mathcal{PA}, fromChild \rangle$
    if $\mathcal{F}_{\neg e} \neq \emptyset$ then
      $(\mathcal{F}_{\neg e}^{btm}, \mathcal{F}_{\neg e}^{\neg btm}) \leftarrow \textsc{SplitOnMark}(\mathcal{F}_{\neg e}, bottom)$
      AddMark$(\mathcal{F}_{\neg e}^{\neg btm}, bottom)$
      for each $\mathcal{CH} \in \textsc{GetChildren}(\mathcal{F}_{\neg e}^{\neg btm}, M)$ do
        $S \leftarrow S \cup \langle \mathcal{CH}, fromParent \rangle$
$R \leftarrow \{\mathcal{R} \mid$ HasMark$(\mathcal{R}, top)\}$
return $(R, e_R)$
```

as input a ground atom $q(a_1, \ldots, a_n)$ and outputs a first-order node $(p, C)$ with $p = q(X_1, \ldots, X_n)$ and $C = \{X_i = a_i\}$.

### Scheduling visits

Where Bayes-ball has a schedule with pairs of nodes and directions to keep track of scheduled visits, FOBB utilizes a schedule containing pairs of first-order nodes and directions. An entry in the schedule containing first-order node $\mathcal{F}$ stands for a set of visits to the ground nodes in $RV(\mathcal{F})$. When a ground node in Bayes-ball receives a ball it will respond according to the rules in Sec. 5.2.1. In FOBB, however, it is possible that not all ground nodes represented by $\mathcal{F}$ pass the ball in the same way. This happens when some of the nodes are part of the evidence, or when not all the nodes have the same marks. In this case the first-order node $\mathcal{F}$ is split into new first-order nodes representing subsets of the ground nodes in $\mathcal{F}$ that pass the ball identically.

For example, if $\mathcal{F}$ receives a ball from one of its children, those ground nodes in $RV(\mathcal{F})$ that are part of the evidence and those that already have a top mark do not need to pass the ball to their parents, while the other ones do.

In Alg. 3 two methods are used to split up a first-order node. First, SplitOnEvidence uses the *evidence* to split up first-order node $\mathcal{F}$ into $\mathcal{F}_e$ (containing all the evidence nodes in $\mathcal{F}$), and $\mathcal{F}_{\neg e}$ (containing non-evidence nodes of $\mathcal{F}$). All evidence atoms of predicate $p$ can be represented as a first-order node $(p, C_e)$, then $\mathcal{F}_e = (p, C \cap C_e)$ and $\mathcal{F}_{\neg e} = (p, C \setminus C_e)$. After $\mathcal{F}$ is split on evidence, first $\mathcal{F}_e$ receives the ball (and is added to the set of visited evidence atoms), and then $\mathcal{F}_{\neg e}$.

Second, the obtained first-order node $\mathcal{F}_{\neg e}$ is split further by SplitOnMark. To split $\mathcal{F} = (p, C)$ on mark $m$, we need to consult the marks table and find entries $\langle (p, C_i), \mathbf{M}_i \rangle$, such that $m \in \mathbf{M}_i$. Using the found entries, FOBB splits $\mathcal{F}$ into first-order node $\mathcal{F}^m = \bigcup_i \mathcal{F}_i = \bigcup_i (p, C \cap C_i)$, which has the mark $m$, and $\mathcal{F}^{\neg m} = \mathcal{F} \setminus \mathcal{F}^m$, which does not have the mark $m$.

After a first-order node is split into subsets which perform the same action to the ball, each subset can pass the ball to its parents/children.

### Passing the Ball to Parents and Children

Like in Bayes-ball, passing a ball from a node to its parents or children is done by following the outgoing or incoming links and scheduling a visit to the found nodes. However, since FOBB does not construct the fully grounded Bayesian network, the PBN must be used to find the parents and children of each first-order node. This is done differently for *atom* and *choice* nodes, considering the transformation of PBNs to their EBN.

The parents of each ground *atom* node $a \in RV(\mathcal{F})$ in the EBN are those choice nodes corresponding to the ground events which have $a$ in their head. Thus, to find the parents of a first-order *atom* node $\mathcal{F} = (p, C)$, we first find the set of events $\{e_1, \ldots, e_k\}$, such that there is a (renaming) substitution $\theta_i$ where $head(e_i) = p\theta_i$. Each $e_i$ can be represented by a first-order *choice* node $\mathcal{B}_i = (e'_i, C_i)$ where $e'_i$ is the atom associated to events $e_i$ and each $C_i$ is the constraint defined on the variables of $e_i$ in the PBN. Then, the parents of first-order node $\mathcal{F}$ in the EBN can be represented as first-order nodes $\mathcal{PA}_i = (e'_i, C'_i)$, where each $C'_i$ restricts $RV(\mathcal{PA}_i)$ to those which are parent of a node in $RV(\mathcal{F})$. Each constraint $C'_i$ is equivalent to the relation acquired from the *natural join* $C_i \bowtie C$ of relations $C_i$ and $C$ (with the variables of $C$ renamed according to substitution $\theta_i$). This way, all the groundings of an event $e_i$ that have an atom $a \in RV(F)$ in their head are captured by the first-order node $\mathcal{PA}_i$. Finding the children of an atom node is similar, only there the connected events are those which have $a$ in their body.

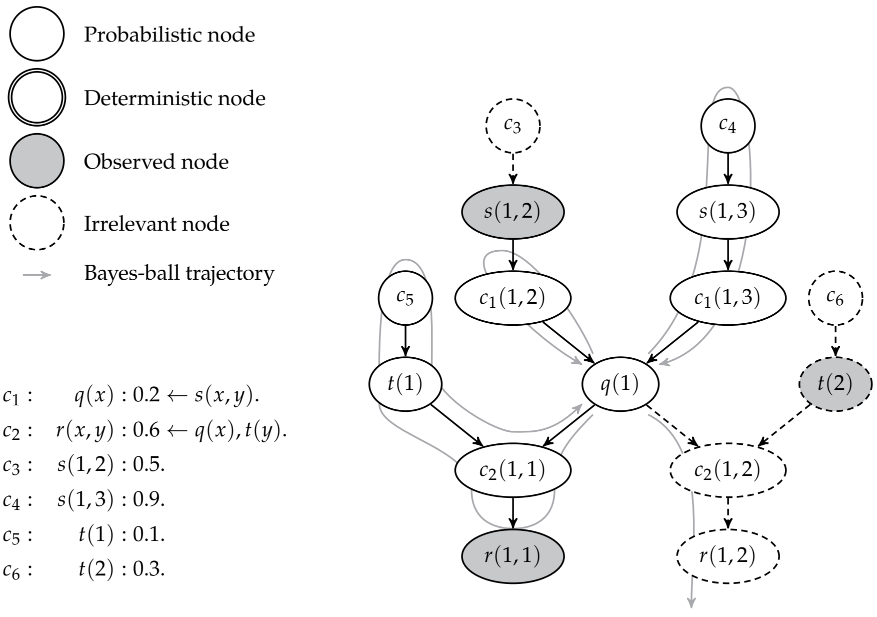
When $\mathcal{F} = (e', C)$ is a *choice* node, its parents are found by considering the body of the event $e$ in the theory, to which $e'$ is associated. Let $body(e) = \{b_1, \ldots, b_n\}$ and $C_e$ be the constraint associated to $e$ in the theory. Then, the parents of $\mathcal{F}$, are first-order nodes $\mathcal{B}_i = (b_i, C_i)$, where $C_i$ restricts $RV(\mathcal{B}_i)$ to those which are parents of a node in $RV(\mathcal{F})$. Each $C_i = \pi_{\mathbf{X}_i}(C \cap C_e)$ is the relation acquired from projecting $C \cap C_e$ on variables $\mathbf{X_i} = Var(b_i)$. Similarly, for finding the children the head of the event $e$ is considered instead of the body.

Having computed the parents $\mathcal{PA}_i$ (using GetParents), in the end an entry $\langle \mathcal{PA}_i, fromchild \rangle$ is registered in the schedule, for each $\mathcal{PA}_i \neq \emptyset$, to pass the ball to parents of $\mathcal{F}$. Similarly, to pass the ball to children an entry $\langle \mathcal{CH}_i, fromParent) \rangle$ is added to the schedule for the computed children $\mathcal{CH}_i$ (using GetChildren).

## Assigning Marks

After passing the ball to the parents (children) of $\mathcal{F}$, FOBB needs to mark $\mathcal{F}$ on top (bottom). This can be naively done by adding $\langle \mathcal{F}, \{top\} \rangle$ to the marks table. In this way, however, the marks table might include overlapping entries, that is, there might be a $\langle \mathcal{F}', \mathbf{M}' \rangle$ in the marks were $RV(\mathcal{F}) \cap RV(\mathcal{F}') \neq \emptyset$. In this case, $\mathbf{M}'$ contains only the *bottom* mark, since $\mathcal{F}$ is split on the top mark when retrieved from the schedule, guaranteeing that no subset of it has the top mark. Hence, the subset $\mathcal{F} \cap \mathcal{F}'$ should now have both the *top* and *bottom* marks, and should be grouped together. In general, when assigning a mark $m$

to $\mathcal{F} = (p, C)$, if there is an overlapping mark $\mu = \langle \mathcal{F}' = (p, C'), \mathbf{M}' \rangle$ then we need to *split the marks*: First, $\mu$ is removed from the marks table, and then the marks $\mu_1 = \langle \mathcal{F} \cap \mathcal{F}', \mathbf{M}' \cup \{m\} \rangle$, $\mu_2 = \langle \mathcal{F} \setminus \mathcal{F}', \{m\} \rangle$, and $\mu_3 = \langle \mathcal{F}' \setminus \mathcal{F}, \mathbf{M}' \rangle$ are assigned instead. (Assigning these marks might result in further splits.) In this manner all the marks $\langle (p, C_i), \mathbf{M}_i \rangle$ form a partition on all the groundings of $p$ which have been visited, such that all the nodes in each $\mathcal{F}_i = (p, C_i)$ have exactly the same marks.



$c_1: \quad q(x) : 0.2 \leftarrow s(x, y).$

$c_2: \quad r(x, y) : 0.6 \leftarrow q(x), t(y).$

$c_3: \quad s(1, 2) : 0.5.$

$c_4: \quad s(1, 3) : 0.9.$

$c_5: \quad t(1) : 0.1.$

$c_6: \quad t(2) : 0.3.$

a. CP-theory b. EBN

Figure 5.4: Illustration of how the Bayes ball moves through the network.

## 5.4 Extensions

In this section we propose some extensions to the core FOBB algorithm.

### 5.4.1 Extension for Implicit Domains

The semantics of many probabilistic logic languages, such as BLPs (Kersting and De Raedt 2007) and CP-logic (Meert, Struyf, and Blockeel 2008b), declare an implicit domain for their logic variables. Although FOBB requires explicit domains, it can be extended to deduce the domains dynamically. Formally, we want to restrict the random variables to the *least Herbrand model* of the corresponding logic program. Intuitively, the set of ground nodes represented by a probabilistic logic model $M$ are those which have a proof in $M$. To comply with these semantics, FOBB too needs to identify which random variables have a proof.

Most formalisms use some form of backward-chaining, such as SLD resolution, to find the least Herbrand model. The same idea can be adopted in FOBB. Note that Bayes-ball (and FOBB) effectively forms the backward-chains for each node from which the ball is passed to its parents and then its ancestors. Hence, FOBB is searching for proofs in a similar way to SLD resolution. The backward-chain ends whenever a root node or an evidence node receives the ball from a child. At this point we know whether this node has a proof. By chaining this information forward through the network, the nodes which have a proof can be identified.

In practice an extra mark, called a *proof mark*, is used to indicate what nodes have been proven. The MRN is then constituted by those nodes that have not only the top mark but also the proof mark. Also, the schedule has to give preference to those balls that have been passed on from proved parents.

### 5.4.2 The closed world assumption as evidence

A common concept in logic programming is the *closed world assumption*: what is missing from our knowledge base is assumed to be false. CP-logic also follows this principle since from a causal point of view nothing can happen without a reason. This means that if an atom in the body of a CP-event is in no head of another CP-event, it has to be false. If the atom is in the body as a positive literal, this means that the body is always false, a negative literal is true and can be ignored.

When FOBB is looking for a parent of an atom node and it cannot find a parent, this means that this atom will always be false. Therefore, we can consider the fact that this atom is false as part of the evidence. This knowledge, deduced during the execution of FOBB, can be used to prune the network (see next section).

### 5.4.3 Propagating evidence

A functional node that is not observed can be *effectively observed* if its parents are observed (Shachter 1999). Since a functional node is deterministically defined by its parents, knowing the values of its parent means also knowing the value of the node itself. The set of evidence can thus be extended iteratively with extra evidence in the case of functional nodes.

In some cases, knowing the values of some of the parents can be enough to propagate this evidence. Take for example a functional node expressing an and-function. If we observe that one of the parents is false, we know that this and-node also takes the value false. In the case of CP-events we know that this event will not cause any atom to be true if one of the literals in the body is false.

More formally, if the nodes $E$ are observed, a node $f$ is said to be *functionally determined* by $E$ if either $f \in E$ or $f$ is a deterministic function of $E$. The set of nodes functionally determined by $E$, $F_E$, can be described by the recursive equation

$$F_E \leftarrow E \cup \{i \text{ is functional} : Pa(i) \subseteq F_E\}$$

### 5.4.4 Shattering

Most lifted inference methods perform a pre-processing step called *shattering* (Milch et al. 2008). Given a set of parfactors $P$, the shattering operation performs all the splits and expansions necessary to ensure that for any two parameterized random variables present in parfactors in $P$, the sets of random variables represented by these two parameterized random variables are either identical or disjoint. Shattering a set of parfactors results in a set of constraints over the logic variables for every atom that is shared across parfactors. For lifted inference this ensures that lifted inference operators like the ones in Milch et al. (2008) are applicable.

Shattering has as added bonus that the shattered parfactors can be represented in a graphical way that is easy to interpret. The set of parfactors can be represented by a graph where the nodes are the atoms (with logic variables) and an edge between two nodes means that there is an edge between all the groundings of this pair of nodes. Such a graphical representation is useful in a diagnostics application like the badminton playing robot (see Section 1.2.2).

Shattering can be integrated with the FOBB algorithm in an iterative way. When visiting the parents or children of a first-order node, we add an additional check to see if the constraints on the subset of logic variables that are in both first-order nodes are identical. If this is not the case, we split the constraints on both first-order nodes and add them again to the queue to propagate this split. This extra check ensures that the resulting theory FOBB gives is also shattered.

**Example 5.1.** Consider the following CP-theory with type $\mathrm{dom}(X) = [0,9]$:

$$(c_1)\forall X\,;\,\{X \in [0,4]\}\ :\ t(X) : 0.5.$$

$$(c_2)\forall X\,;\,s(X) : 0.5 \leftarrow t(X).$$

$$(c_3)\forall X\,;\,r : 0.5 \leftarrow s(X).$$

When applying FOBB for $\Pr(r)$, FOBB will return that in this case all CP-events are requisite and therefore the MRN is the entire theory. We can now use the transformation to an equivalent Bayesian network to visualize the MRN (see Figure 5.5.a). This representation has the disadvantage that it loses the compactness of the first-order representation. We can naively represent the first-order MRN by means of a graphical model (see Figure 5.5.b) but the semantic meaning then becomes unintuitive and ambiguous. The meaning of the edge between the top nodes is not clear since both nodes represent a different number of ground nodes and it is not clear which nodes are connected. This problem is solved by shattering after which the MRN can be represented by a parameterized Bayesian network (see Figure 5.5.c). We see that some of the root nodes are atom nodes without a parent representing the CP-event causing the atom. According to the CP-logic semantics, such an atom is false and remains false. We can consider this as evidence and propagate this evidence as explained in Section 5.4.3. This simplifies the network further resulting in the parameterized Bayesian network in Figure 5.5.d.

$\Diamond$

The FOBB algorithm (see Algorithm 3) is extended as follows to incorporate shattering:

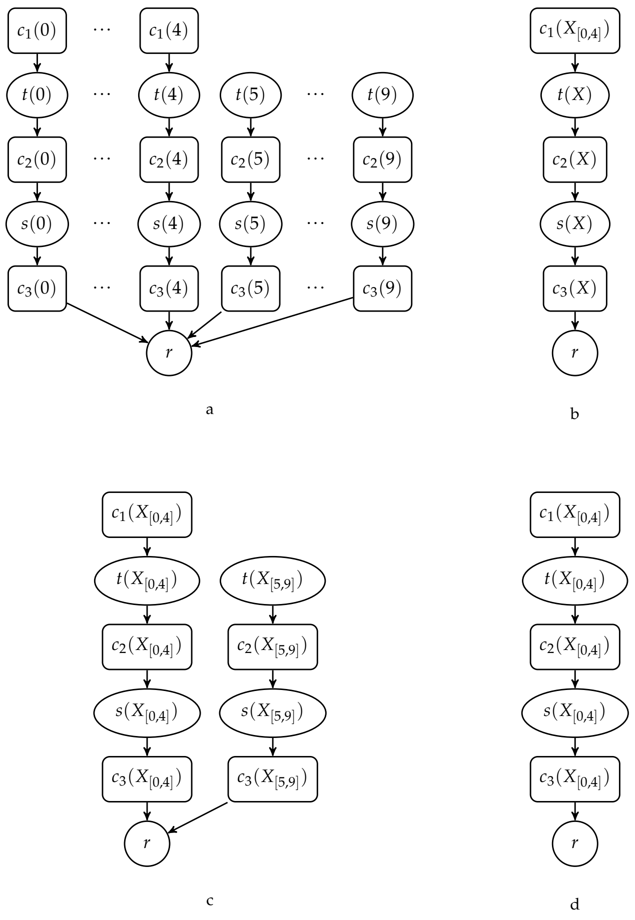

$c_1(0)$ ⋯ $c_1(4)$
$t(0)$ ⋯ $t(4)$ $t(5)$ ⋯ $t(9)$
$c_2(0)$ ⋯ $c_2(4)$ $c_2(5)$ ⋯ $c_2(9)$
$s(0)$ ⋯ $s(4)$ $s(5)$ ⋯ $s(9)$
$c_3(0)$ ⋯ $c_3(4)$ $c_3(5)$ ⋯ $c_3(9)$
$r$
a
$c_1(X_{[0,4]})$
$t(X)$
$c_2(X)$
$s(X)$
$c_3(X)$
$r$
b
$c_1(X_{[0,4]})$
$t(X_{[0,4]})$ $t(X_{[5,9]})$
$c_2(X_{[0,4]})$ $c_2(X_{[5,9]})$
$s(X_{[0,4]})$ $s(X_{[5,9]})$
$c_3(X_{[0,4]})$ $c_3(X_{[5,9]})$
$r$
c
$c_1(X_{[0,4]})$
$t(X_{[0,4]})$
$c_2(X_{[0,4]})$
$s(X_{[0,4]})$
$c_3(X_{[0,4]})$
$r$
d

Figure 5.5: Graphical representations of the MRN of the CP-theory given in Example 5.1 for the query $\Pr(r)$. The graphical obtained is (a) a Bayesian network representing the EBN, (b) a network visualizing the first-order MRN but without an intuitive semantic meaning, (c) a parameterized Bayesian network obtained after shattering the first-order MRN, and (d) a parameterized Bayesian network after shattering the first-order MRN and removing the nodes for which evidence can be propagated that the node is false.

- Calls to functions SplitOnMark and AddMark now also split a first-order node when the constraints are different even thought the marks are already set.

- If a first-order node is split because a difference in constraints, this splitting is propagated to already visited parents and children. This ensures that parents and children have the same constraints on their common logic variables.

## 5.5 Experiments

In our experiments, we investigated how the size of the domain of logic variables affects the search for the MRN, and how this MRN affects inference for probabilistic logic models. All experiments are performed on an Intel Pentium D CPU 2.80GHz processor with 1GB of memory available. FOBB itself is implemented in C++.

As a first experiment, we take the theory shown in Fig. 5.2 and compute the conditional probability of the atom $q(1)$ while varying the size of the domain $Num$. One third of the ground nodes is chosen at random and considered as observed. For the inference, while any Bayesian network inference could be applied, we used an implementation in C++ performing variable elimination with the optimization proposed in (Díez and Galán 2003) to obtain linear inference for noisy-or nodes. Five approaches were used to obtain a Bayesian network from the original theory: (a) ground the entire network based on the domains; (b) ground the entire network and use Bayes-ball to limit the resulting equivalent Bayesian network to the MRN; (c) ground the network by means of SLD resolution (using Prolog and setting the query and evidence as goals); (d) ground by means of SLD resolution and use Bayes-ball to find the MRN; and (e) use FOBB to find the MRN directly from the theory and ground the MRN. For methods (a) and (b) the theory shown in Fig. 5.2 was transformed first to a ground theory and afterwards compiled to an equivalent Bayesian network. Methods (c) and (d) required to first ground the facts (events with empty body) according to the domains.

Fig. 5.7 shows the results of the first experiment. The bottom graph shows that FOBB is magnitudes faster in finding the (grounded) MRN than any of the other methods. The top graph shows that the complexity of performing inference grows faster than that of grounding. As a consequence, for large networks the approach used to find the MRN became of less importance in the total inference time. These results, however, confirm the importance of restricting the network to the MRN: The two methods that do not restrict the network, full grounding and SLD resolution, could not handle even the smaller networks and ran out of memory. This result motivates investigating the effect of restricting computations to the MRN in lifted inference, for which no method like Bayes-ball has been proposed to date. We applied FOBB to such a case in our second experiment.

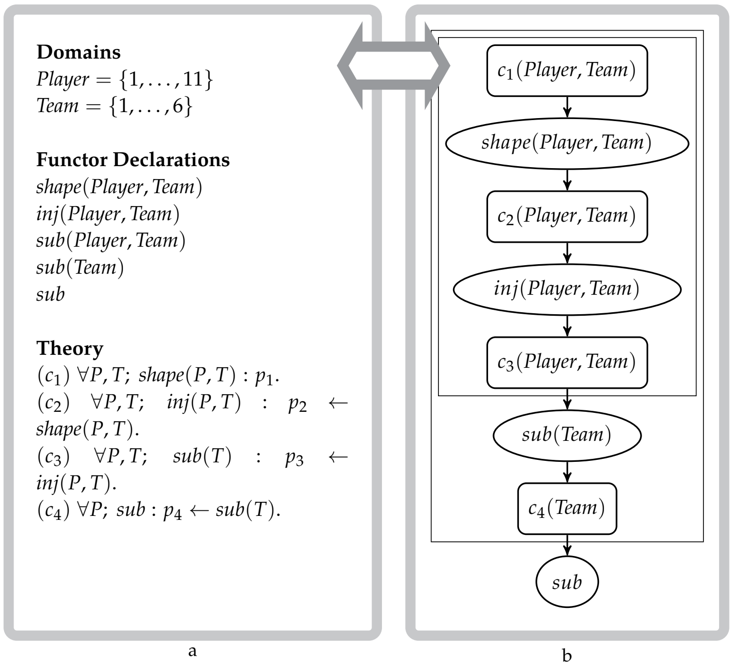
In the second experiment we used the theory shown in Fig. 5.6. This theory is an extension of the theory used in (Kisynski and Poole 2009b) to benchmark lifted inference methods. This theory represents that whether a player in a soccer tournament is substituted during the tournament depends on whether he gets injured. The probability of an injury depends on the physical condition of the player. We compute the conditional probability of $substitution(1)$ given that six teams participate, while varying the number of players in a team. For four of the teams there is evidence that some player has been injured.

For the results in Fig. 5.8 we used the same strategy as for the previous experiment. In addition to propositional inference we also used the lifted inference technique C-FOVE (Milch et al. 2008) available in BLOG[2] (Java) to calculate the conditional probability of the query. The factors are created based on the optimizations mentioned in (Kisynski and Poole 2009b).

The soccer model is very symmetric and inference is therefore efficient. The results in Fig. 5.8 show that the complexity of grounding and inference are both linear. In this case the efficiency of grounding has a noticeable influence on the total inference time.

For this model, a lifted inference method can make abstraction of the domain size for performing probabilistic inference, and can therefore calculate the marginal probability of the query in constant time. This is shown in Fig. 5.9. FOBB allows us to find the MRN in a form that can be interpreted by a lifted inference method. With this combination, we can thus not only make abstraction of the domain size but also ignore non-requisite parts of the first-order probabilistic model. FOBB can have a greater influence on the inference when applied to more comprehensive models, since it is possible to have non-requisite parts of arbitrary complexity. Such an effect can be observed, for example, when the model contains an extra team that uses a more complex combining rule for $sub$ than noisy-or. This causes inference to be exponential on the non-requisite parts. These unnecessary computations are avoided when using FOBB, as shown in Fig. 5.10.

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



**Domains**
*Player* = $\{1, \ldots, 11\}$
*Team* = $\{1, \ldots, 6\}$

**Functor Declarations**
$shape(Player, Team)$
$inj(Player, Team)$
$sub(Player, Team)$
$sub(Team)$
$sub$

**Theory**
$(c_1)$ $\forall P, T;\ shape(P, T) : p_1.$
$(c_2)$ $\forall P, T;\ inj(P, T) : p_2 \leftarrow shape(P, T).$
$(c_3)$ $\forall P, T;\ sub(T) : p_3 \leftarrow inj(P, T).$
$(c_4)$ $\forall P;\ sub : p_4 \leftarrow sub(T).$

Figure 5.6: (a) First-order model and (b) its equivalent belief network. The scopes of logic variables are indicated by the rectangles (plates).

## 5.6 Conclusions

In this work, we presented a first-order version of Bayes-ball called FOBB, which finds the minimum relevant network for a given set of query and evidence atoms. The advantages of using FOBB are twofold; first, it is more efficient to find the ground network needed to calculate the probability of the query than current methods. Second, the resulting relevant network is first-order, permitting it to be used as input for lifted inference methods which have shown to offer magnitudes of gain in speed and memory. FOBB resembles the approach by Singla and Domingos (2008) in aggregating ground nodes as one unit and building a *lifted network*; major differences are that FOBB is meant for directed graphs instead of undirected graphs, that it is dependent on the query and, that it is not a compilation technique.

In general, empirical evaluations of lifted inference algorithms are done using

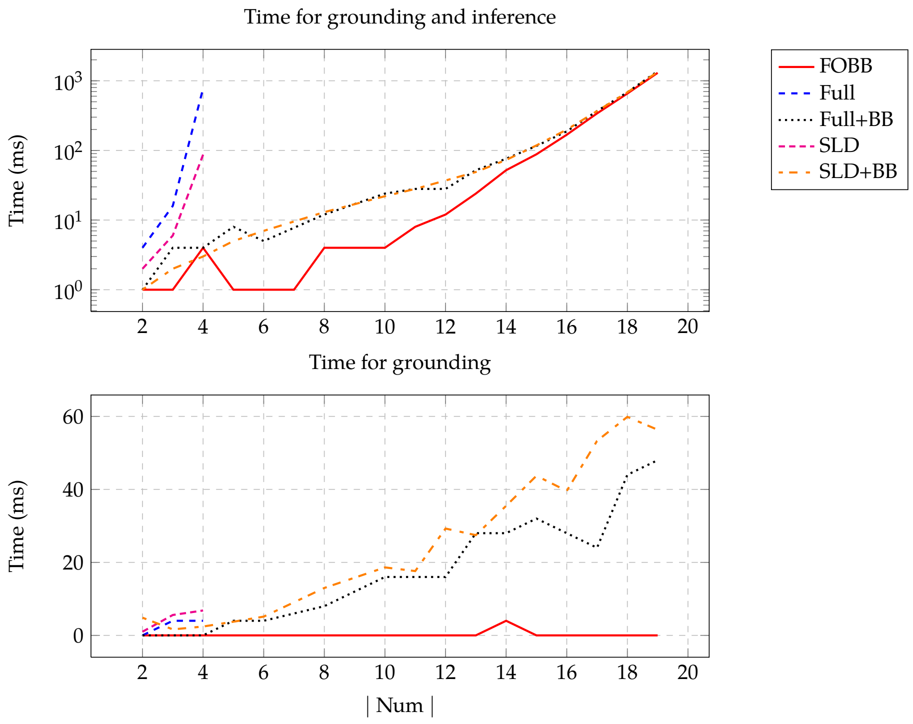


Figure 5.7: Performance on the model in example in Fig. 5.2. The top graph shows the combined time needed to ground the theory and perform inference, the bottom graph shows only the time need for grounding.

simple first-order probabilistic models. FOBB is a valuable companion to existing lifted inference methods like the one proposed in (Kisynski and Poole 2009b) to handle more comprehensive and real-life models instead of these simple models.

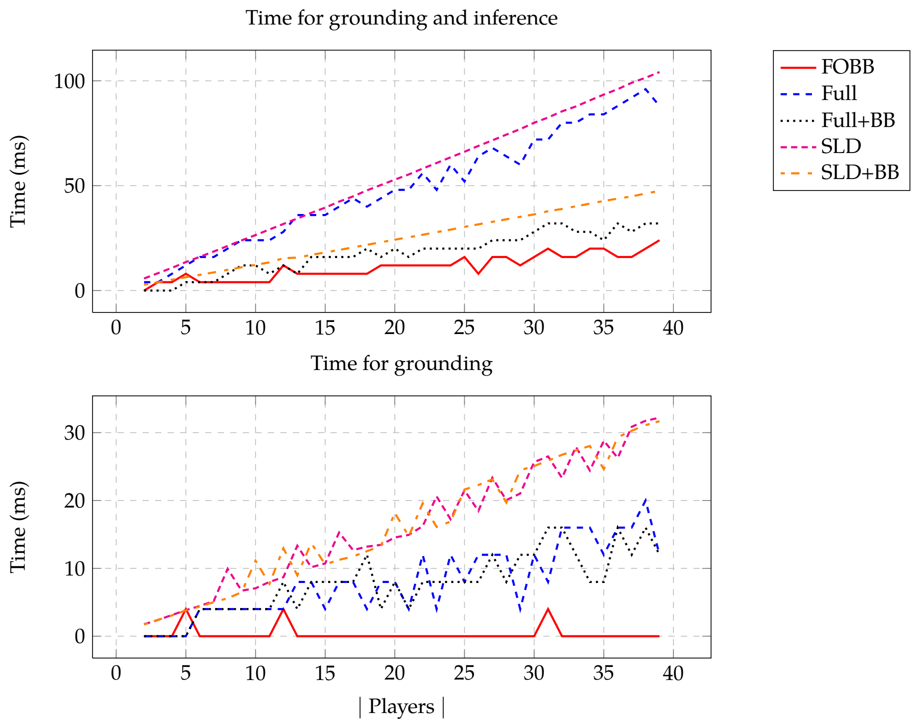


Figure 5.8: Performance on the soccer example using propositional inference. The top graph shows the combined time needed to ground the theory and perform inference, the bottom graph shows only the time need for grounding.

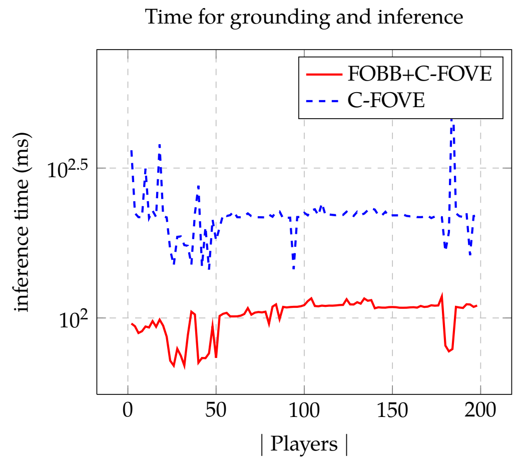


Figure 5.9: Performance on the soccer example using lifted inference.

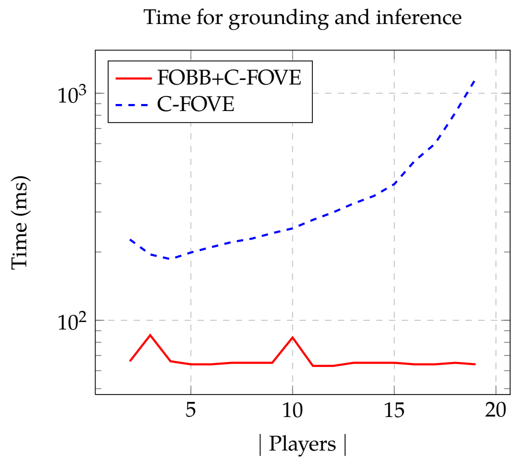


Figure 5.10: Performance on the soccer example using lifted inference with a more complex interaction than noisy-or for one extra team.

6

# Contextual Variable Elimination with Overlapping Contexts

## 6.1 Introduction

In this chapter we build further upon Bayesian networks (BN) (Pearl 1988) as an inference method for PLL. The BN defines the variables' joint probability distribution, which is compactly represented as a product of factors. This product includes for each variable one factor, which is typically encoded as a table that represents the conditional probability distribution (CPD) of the variable given its parents in the BN. BN inference (computing the conditional probability of a query variable given certain evidence) can be performed by summing out all non-query non-evidence variables from the factorization. This is essentially what the variable elimination (VE) algorithm (Zhang and Poole 1996) does.

A large number of the formalisms used for PLL can be converted into BNs (e.g., CP-logic, ProbLog, ICL, PRISM, BLP, see Chap. 3). For these formalisms, only exploiting the notion of independence does not yield the most efficient representation possible. The CPDs resulting from the conversion exhibit a particular internal structure, which we will call *local structure*. For example, a CPD may be given as a decision tree (Ramon et al. 2008), may express that the different conditions influence the variable independently (noisy-or or noisy-max) (Cozman 2004), or may impose constraints on the range of a variable in a certain context (Meert, Struyf, and Blockeel 2008b). In these cases, a table based representation contains redundancies; an alternative representation that avoids these redundancies can yield more efficient inference.

Several methods have been proposed to exploit local structure. For instance, the contextual variable elimination (CVE) algorithm (Poole and Zhang 2003) uses a more compact representation if some of the CPDs can be represented by a decision trees or in general exhibit contextual independence. This reduces the tree-width of the network, thus allowing for more efficient inference. CVE is a generalization of probability trees and a comparison is made in (Poole and Zhang 2003). Another example is multiplicative factorization (MF) (Díez and Galán 2003) which can exploit local structures like independent causation (e.g. noisy-or/and). CVE and MF each exploit one particular type of structure, but cannot handle the other. CVE relies on contexts being mutually-exclusive and exhaustive (we will call this the *MEE-restriction*), which makes it unsuitable to combine it with MF. This is explained in more detail in Sec. 6.2.3.

There are also methods that utilize a preprocessing phase to compile the belief network into a different structure, which is optimized for answering multiple queries and allows efficient inference with particular types of local structure. Some known methods are the Arithmetic Circuits (AC) of (Chavira and Darwiche 2007) and the AND/OR-trees of (Mateescu and Dechter 2008). In PLL, for each query a new network is built (every time requiring a compilation), therefore, we need to take into account both the compilation time and the inference time. This makes that there is a trade-off between the time spend in optimizing a secondary structure and the inference phase itself.

The main contribution of this chapter is that we show how in CVE the MEE-restriction can be lifted. This leads to a new method, *CVE-OC*: CVE with overlapping constraints. Lifting the MEE-restriction has two important consequences: (a) contexts can be encoded more compactly, with increased efficiency as a result, and (b) factorizations like MF can also be handled, which means that CVE-OC can exploit all structures that CVE and MF can exploit. An additional contribution, is that CVE-OC handles constraints on the range of multi-valued variables. This is an extension of CVE for which no concrete method had been presented before.

### 6.1.1 Bibliographical Note

After realizing that inference for PLL posed some challenges we performed a comparison between known inference systems for CP-logic and some simple Bayesian network inference techniques. This work was first published at ILP 2009.

> W. Meert, J. Struyf, and H. Blockeel (2009a). "CP-logic theory inference with contextual variable elimination and comparison to BDD based inference methods". In: *Proceedings ofthe 19th International Conference on Inductive Logic Programming (ILP)*. (Leuven, Belgium, July 1–4, 2009). Volume 5989. Lecture Notes in Computer Science, pages 96–109.

The comparison showed that although contextual variable elimination could efficiently deal with determinism and contextual independence, the presence of causal independence in the Bayesian network caused inefficient inference. We investigated Bayesian network inference techniques that are able to handle causal independence techniques and proposed a new technique to combine all the previous at PGM 2010.

> W. Meert, J. Struyf, and H. Blockeel (2010a). "Contextual variable elimination with overlapping contexts". In: *Proceedings ofthe 5th European workshop on Probabilistic Graphical Models (PGM10)*. (Helsinki, Finland, Sept. 13–15, 2010), pages 193–210

### 6.1.2 Structure of this Chapter

This chapter is organized as follows. We start by providing background on CVE and MF. After this we present CVE-OC, this chapter's main contribution. Next, we discuss its usefulness in the context of statistical relational learning. We present an experimental evaluation in that context, and finally present our conclusions. For a reader not familiar with CVE, it is recommended to read Poole and Zhang (2003).

## 6.2 Existing inference techniques for CP-logic

Since CP-logic was introduced, several inference methods have been proposed for (a subset of) CP-logic. The efficiency of these methods is crucial for developing fast parameter and structure learning algorithms (Meert, Struyf, and Blockeel 2008b). In this section, we give an overview of some inference techniques for CP-logic and for equivalent Bayesian networks. Detailed descriptions about these methods can be found in the referenced papers.

**Example 6.1.** We will use the following CP-theory as a running example:

$$
\begin{aligned}
shops(john) &: 0.2. && (c_1)\\
shops(mary) &: 0.9. && (c_2)\\
(spaghetti : 0.5) \vee (steak : 0.5) &\leftarrow shops(john). && (c_3)\\
(spaghetti : 0.3) \vee (fish : 0.7) &\leftarrow shops(mary). && (c_4)
\end{aligned}
$$

This CP-theory models the situation that John and his partner Mary may independently decide to go out to buy food for dinner. ♢

### 6.2.1 BDD-based approaches

**ProbLog.** ProbLog (Kimmig et al. 2008) is a probabilistic logic programming language that can serve as a target language to which other probabilistic logic modeling languages can be compiled. In particular, acyclic CP-theories without negation[1] can be translated into ProbLog as explained in Section 4.7.

ProbLog's inference engine works as follows. Given a query, it first computes all proofs of the query and collects these in a DNF formula. Next, it converts this formula to a binary decision diagram (BDD). Relying on this BDD representation, ProbLog then computes the query's probability in one bottom-up pass through the BDD (using dynamic programming).

**cplint.** Inspired by ProbLog, Riguzzi (2007) proposes *cplint*, which is a CP-theory inference system that makes use of BDDs in a similar way as ProbLog. There are two differences with the transformation to ProbLog. First, cplint uses a different encoding to represent which head atom is caused by a CP-event.

[1]Recently, support for negation has been added to ProbLog (Kimmig 2010)

Second, cplint supports negation. When it encounters a negative body literal $\neg a$ in a proof, it computes all proofs of $a$ and includes the negation of the DNF resulting from all these proofs into the original DNF (note that this process is recursive).

### 6.2.2 Variable elimination

In Chap. 3 we presented a transformation that can transform any acyclic CP-theory with a finite Herbrand universe to an equivalent Bayesian network (EBN). Based on this transformation, CP-theory inference can be performed by applying the transformation on the given theory and then running a Bayesian network (BN) inference algorithm, such as variable elimination (VE), on the resulting EBN.

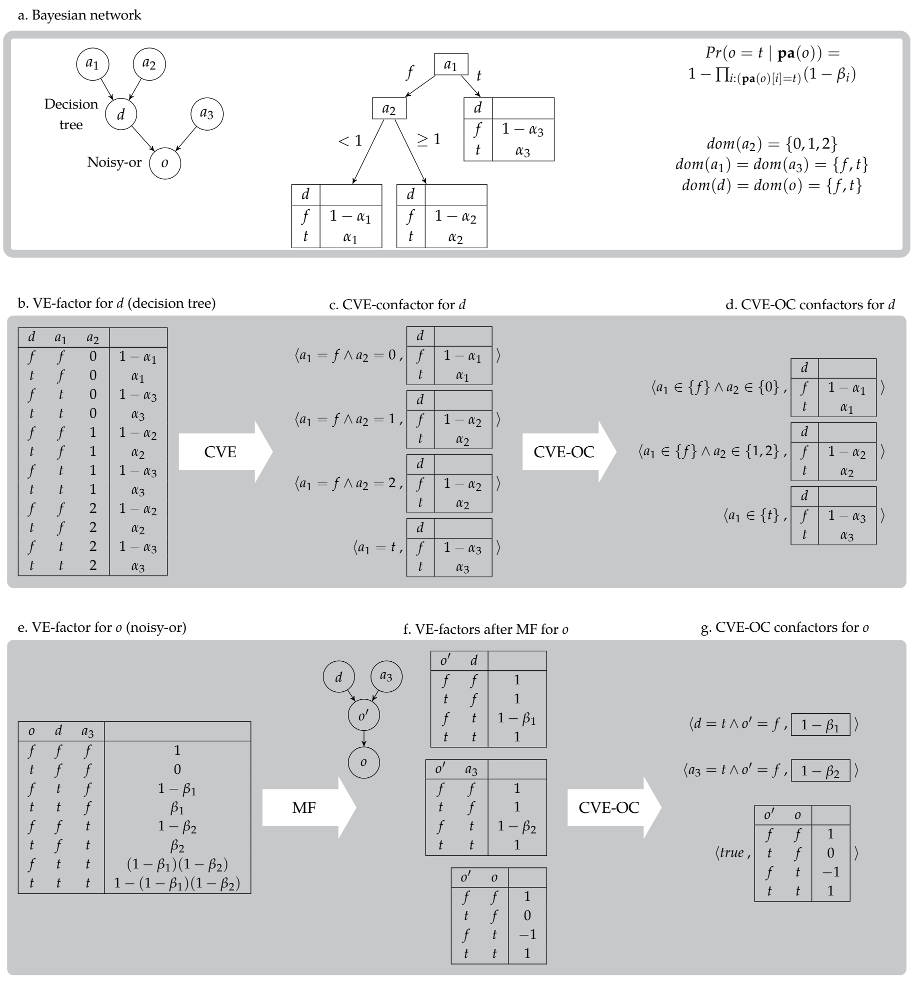
The shopping CP-theory can be transformed to an EBN which in fact is a set of factors. A CP-theory may contain more structural information than a BN, and this structural information has to be encoded numerically in the factors of the EBN. This can result in factors with redundant information (a factor may have many identical columns) and cause suboptimal inference. This effect can be seen in the top-left factor of Fig. 6.2.a, which represents that *spaghetti* is true if John or Mary buys spaghetti. This factor has many identical columns.

### 6.2.3 Contextual variable elimination (CVE)

To address the problem of redundant information in the factors, we propose to use contextual variable elimination (CVE) (Poole and Zhang 2003), which is an extension to VE that exploits contextual independence to speed up inference by representing the joint probability distribution as a set of *confactors* instead of factors. Confactors can be more compact than factors because they explicitly encode structural information by means of so-called contexts.

**Contextual variable elimination algorithm.** CVE makes use of a more specific form of conditional independence known as *contextual independence* (Boutilier et al. 1996; Poole and Zhang 2003).

**Definition 6.1** (Contextual Independence)**.** Assume that $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ and $\mathbf{c}$ are sets of variables. $\mathbf{x}$ and $\mathbf{y}$ are contextually independent given $\mathbf{z}$ and context $\mathbf{c} = \mathbf{v_c}$,


a. Bayesian network
Decision tree
Noisy-or
$Pr(o = t \mid \mathbf{pa}(o)) = 1 - \prod_{i:(\mathbf{pa}(o)[i]=t)}(1-\beta_i)$
$dom(a_2) = \{0,1,2\}$
$dom(a_1) = dom(a_3) = \{f,t\}$
$dom(d) = dom(o) = \{f,t\}$
b. VE-factor for $d$ (decision tree)
c. CVE-confactor for $d$
d. CVE-OC confactors for $d$
CVE
CVE-OC
e. VE-factor for $o$ (noisy-or)
f. VE-factors after MF for $o$
g. CVE-OC confactors for $o$
MF
CVE-OC
$\langle a_1 = f \wedge a_2 = 0,$
$\langle a_1 = f \wedge a_2 = 1,$
$\langle a_1 = f \wedge a_2 = 2,$
$\langle a_1 = t,$
$\langle a_1 \in \{f\} \wedge a_2 \in \{0\},$
$\langle a_1 \in \{f\} \wedge a_2 \in \{1,2\},$
$\langle a_1 \in \{t\},$
$\langle d = t \wedge o' = f, \boxed{1-\beta_1} \rangle$
$\langle a_3 = t \wedge o' = f, \boxed{1-\beta_2} \rangle$
$\langle true,$

| $d$ | $a_1$ | $a_2$ | |
|---|---|---|---|
| $f$ | $f$ | 0 | $1-\alpha_1$ |
| $t$ | $f$ | 0 | $\alpha_1$ |
| $f$ | $t$ | 0 | $1-\alpha_3$ |
| $t$ | $t$ | 0 | $\alpha_3$ |
| $f$ | $f$ | 1 | $1-\alpha_2$ |
| $t$ | $f$ | 1 | $\alpha_2$ |
| $f$ | $t$ | 1 | $1-\alpha_3$ |
| $t$ | $t$ | 1 | $\alpha_3$ |
| $f$ | $f$ | 2 | $1-\alpha_2$ |
| $t$ | $f$ | 2 | $\alpha_2$ |
| $f$ | $t$ | 2 | $1-\alpha_3$ |
| $t$ | $t$ | 2 | $\alpha_3$ |

| $o$ | $d$ | $a_3$ | |
|---|---|---|---|
| $f$ | $f$ | $f$ | 1 |
| $t$ | $f$ | $f$ | 0 |
| $f$ | $t$ | $f$ | $1-\beta_1$ |
| $t$ | $t$ | $f$ | $\beta_1$ |
| $f$ | $f$ | $t$ | $1-\beta_2$ |
| $t$ | $f$ | $t$ | $\beta_2$ |
| $f$ | $t$ | $t$ | $(1-\beta_1)(1-\beta_2)$ |
| $t$ | $t$ | $t$ | $1-(1-\beta_1)(1-\beta_2)$ |

| $o'$ | $d$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 1 |
| $f$ | $t$ | $1-\beta_1$ |
| $t$ | $t$ | 1 |

| $o'$ | $a_3$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 1 |
| $f$ | $t$ | $1-\beta_2$ |
| $t$ | $t$ | 1 |

| $o'$ | $o$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 0 |
| $f$ | $t$ | $-1$ |
| $t$ | $t$ | 1 |

Figure 6.1: (a) Belief network with local structure; on top the graphical model, then the decision tree for node $D$ and the definition of noisy-or for node $O$, and at the bottom the domains for the variables; (b) CPD for $D$ represented as a table like in VE; (c) CPD for $D$ represented by confactors for use in CVE; (d) CPD for $D$ represented by confactors for CVE-OC; (e) CPD for $O$; (f) CPD for $O$ factorized with MF for use in VE; (g) CPD for noisy-or represented by confactors for CVE-OC after MF.

with $\mathbf{v_c} \in \mathrm{dom}(\mathbf{c})$, iff

$$\mathrm{Pr}(\mathbf{x}|\mathbf{y} = \mathbf{v_y} \wedge \mathbf{z} = \mathbf{v_z} \wedge \mathbf{c} = \mathbf{v_c}) = \mathrm{Pr}(\mathbf{x}|\mathbf{z} = \mathbf{v_z} \wedge \mathbf{c} = \mathbf{v_c})$$

for all $\mathbf{v_y} \in \mathrm{dom}(\mathbf{y})$ and $\mathbf{v_z} \in \mathrm{dom}(\mathbf{z})$ such that $Pr(\mathbf{y} = \mathbf{v_y} \wedge \mathbf{z} = \mathbf{v_z} \wedge \mathbf{c} = \mathbf{v_c}) > 0$. We also say that $\mathbf{x}$ is *contextually independent* of $\mathbf{y}$ given $\mathbf{z}$ and context $\mathbf{c} = \mathbf{v_c}$ (if we drop $\mathbf{c} = \mathbf{v_c}$, we say $\mathbf{x}$ is conditionally independent from $\mathbf{y}$ given $\mathbf{z}$).

VE (Zhang and Poole 1996) represents the joint distribution as a product of factors, in which each factor is a conditional probability table. CVE (Poole and Zhang 2003) factorizes the joint distribution further by replacing each factor by a set of *contextual factors* or *confactors*. A confactor $r_i$ consists of two parts: a *context* and a *table*:

$$\langle \underbrace{v_1 = v_{1,i} \wedge \ldots \wedge v_{k-1} = v_{k-1,i}}_{\text{context}}, \underbrace{factor_i(v_k, \ldots, v_m)}_{\text{table}} \rangle$$

The context is a conjunction of variable-value tests ($v_j = v_{j,i}$), which indicates the condition under which the table is applicable (if the context is "*true*" the table is always applicable). The context is used to split up factors into confactors based on Def. 6.1. The table stores probabilities for all value assignments of a set of zero or more variables ($v_k, \ldots, v_m$).

The set of confactors that together represent the CPD of a variable $v$ (the confactors *for v*) is mutually-exclusive and exhaustive (MEE). This means that for each possible value assignment for a variable $V$ and its parents $\mathbf{pa}(v)$, there is precisely one confactor of which the table includes the parameter $Pr(v = v_i \mid \mathbf{pa}(v) = \mathbf{v_{pa}})$. These conditions ensure that confactors for the same variable do not overlap, therefore, the set of all confactors for a variable is identical to the original factor.

Fig. 6.1.c shows a confactor representation of a BN for which the CPD for $d$ can be represented by a decision tree. This CPD can be compactly represented as a set of confactors of which each context is the conjunction of variable-value tests on a path from the decision tree root to one of its leaves. (As a note, the converse is not true; a decision tree cannot always represent confactors equally compactly.)

We describe CVE at a high level (the complete algorithm can be found in (Poole and Zhang 2003)). Similar to VE, CVE eliminates the non-query, non-evidence variables one by one from the joint distribution. To eliminate a variable $e$, it relies on four basic operations:

1. $\langle c, t_1 \rangle \otimes \langle c, t_2 \rangle \equiv \langle c, t_1 \otimes t_2 \rangle$, multiplying two confactors with identical contexts $c$.
2. $\sum_e \langle c, t \rangle \equiv \langle c, \sum_E t \rangle$, summing out a variable $e$ that appears in the table of a confactor.
3. $\sum_e (\langle c \wedge e = e_1, t_1 \rangle, \ldots, \langle c \wedge e = e_k, t_k \rangle) \equiv \langle c, \sum t_i \rangle$, summing out a variable $e$, with domain $e_1, \ldots, e_k$, that appears in the contexts.
4. $\langle c, t \rangle \equiv \langle c \wedge x = x_1, t(x = x_1) \rangle, \ldots, \langle c \wedge x = x_k, t(x = x_k) \rangle$, *splitting* a factor; $t(x = x_i)$ is table $t$ but elements for which $x \neq x_i$ are removed.

The first three operations are only possible if the contexts are identical (indicated with $c$) except for the variable to eliminate ($e$). To make the contexts identical, CVE uses the fourth operator (splitting). Given two confactors, repeated splitting can be used to create two confactors with identical contexts. The order in which these operators are applied is chosen heuristically by the CVE algorithm. Splitting creates extra confactors, and therefore the heuristic is such that it avoids this operation as much as possible.

Confactors can represent CPDs more compactly than tables, but, as the previous discussion illustrates, at the cost of more complicated basic operations.

**Equivalent Bayesian network with confactors.** We repeat briefly the transformation explained in Section 3.6 for the example given before (see Fig. 6.2.b). This time however, we create confactors instead of factors:

1. For every CP-event, create a variable (called a *choice*) whose value indicates which head atom is chosen (e.g., $c_4$ indicates the fourth CP-event's choice). The probability distribution for this variable is represented by multiple confactors. The context of one confactor represents the case that the body is true. The other confactors constitute the case that the body is false, and make the set of confactors complete and mutually exclusive.

   For example, the first confactor for $c_4$ (Fig. 6.2, right) represents the case that the body *shops*(*mary*) is true and the event chooses to make one of the head atoms true ($c_4 = 1$ for spaghetti, $c_4 = 2$ for fish). The other $c_4$

confactor corresponds to the case that *shops*(*mary*) is false; in that case no head atom is caused ($c_4 = 0$).

2. For every atom in the theory, create a Boolean variable. The probability distribution of this variable is factorized in multiple confactors that together encode an OR-function (by means of the contexts). If at least one of the events where the atom is in the head has selected the atom, it becomes true; otherwise, it will be false.

   For example, the confactors for *spaghetti* represent an OR-function. The first confactor states that if choice node $c_3$ selects spaghetti it will be true with probability 1. If $c_3$ has not selected spaghetti, it can be choice node $c_4$ that selects spaghetti. This is represented by the second confactor. The last confactor expresses that if none of both parent choice nodes select spaghetti it will be false.

The transformation above can be extended to improve inference efficiency. For example, we represent CP-events that have the same atoms in the head and mutually exclusive bodies by a single choice variable. Also, a factor is not split up if the resulting confactors are not more compact (in terms of the number of parameters) than the original factor (e.g., $c_4$ is not split into two confactors like in Fig. 6.2, but kept as a single factor). Once the set of confactors representing the CP-theory is constructed, we use the CVE algorithm (Poole and Zhang 2003) to perform CP-theory inference.

### 6.2.4 Results

We evaluate the inference methods on the task of inferring the marginal distribution of one designated variable in four CP-theories of varying complexity. We always select the variable with the highest inference cost and do not include any evidence (i.e., we consider the most difficult case).

**Theory 1: Growing heads.** The first theory is called "growing heads" and is made up from events using atom $a_i$. There are two types of events. The first states that for every atom $a_i$ there is an event causing it with probability 50% without a reason. The second one says that if an atom $a_i$ is true, it causes one of

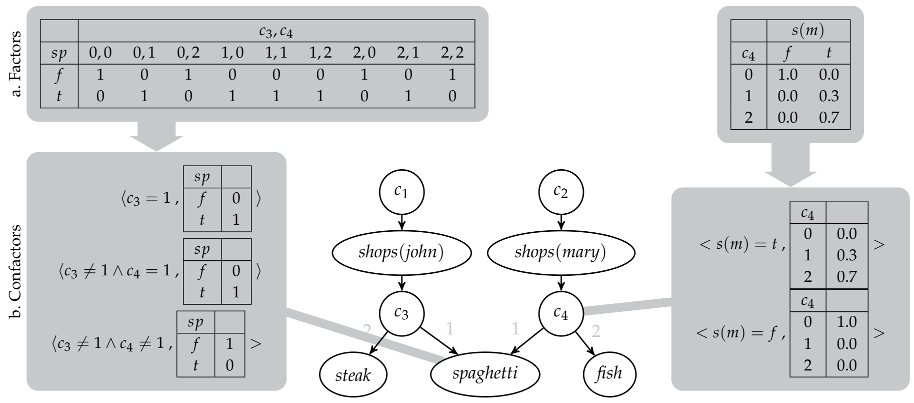

a. Factors
b. Confactors
$c_3, c_4$
$sp$
0,0 0,1 0,2 1,0 1,1 1,2 2,0 2,1 2,2
$f$ 1 0 1 0 0 0 1 0 1
$t$ 0 1 0 1 1 1 0 1 0
$s(m)$
$c_4$ $f$ $t$
0 1.0 0.0
1 0.0 0.3
2 0.0 0.7
$\langle c_3 = 1,$ $sp$ $f$ 0 $t$ 1 $\rangle$
$\langle c_3 \neq 1 \wedge c_4 = 1,$ $sp$ $f$ 0 $t$ 1 $\rangle$
$\langle c_3 \neq 1 \wedge c_4 \neq 1,$ $sp$ $f$ 1 $t$ 0 $>$
$c_1$
$c_2$
$shops(john)$
$shops(mary)$
$c_3$
$c_4$
steak
spaghetti
fish
2 1 1 2
$< s(m) = t,$ $c_4$ 0 0.0 1 0.3 2 0.7 $>$
$< s(m) = f,$ $c_4$ 0 1.0 1 0.0 2 0.0 $>$

Figure 6.2: Factor and confactor representation for node *spaghetti* (left) and $C_4$ (right)

the atoms $a_j$ with $j < i$ with equal probability. The theory for size 4 is

$$\begin{aligned}
a_0 &: 0.5. \\
a_0 &: 1.0 \leftarrow a_1. \\
a_1 &: 0.5. \\
a_0 : 0.5 \vee a_1 &: 0.5 \leftarrow a_2. \\
a_2 &: 0.5. \\
a_0 : 0.33 \vee a_1 : 0.33 \vee a_2 &: 0.33 \leftarrow a_3. \\
a_3 &: 0.5.
\end{aligned}$$

This theory expresses mainly noisy-or relations between the atoms. For example, atom $a_0$ can be caused for no reason, by $a_1$, $a_2$, or $a_3$.

**Theory 2: Growing bodies.** The second theory, named "growing bodies", has for every atom $a_i$ in the theory a set of CP-events that can cause it and whose bodies together make up a decision tree. For an atom $a_i$ there are events $a_i \leftarrow \neg a_k, \ldots, \neg k_l, k_{l+1}$ with $i < k$. Thus, the theory for size 4, consisting of both types of events is

$$\begin{aligned}
&a_0 : 0.5 \leftarrow a_1.\\
&a_0 : 0.5 \leftarrow \neg a_1, a_2.\\
&a_0 : 0.5 \leftarrow \neg a_1, \neg a_2, a_3.\\
&a_1 : 0.5 \leftarrow a_2.\\
&a_1 : 0.5 \leftarrow \neg a_2, a_3.\\
&a_2 : 0.5 \leftarrow a_3.\\
&a_3 : 0.5.
\end{aligned}$$

**Theory 3: Blood type.** The third theory is taken from Kersting and De Raedt (2007) and expresses the probability distribution over the possible blood types a person has given his ancestors. In this theory most atoms appear in the bodies of multiple events. For this reason some way of tabling is useful to avoid calculating the probability of a particular atom again for every event. The parameter indicates how far back we take ancestors into account. An extract of a theory looks as follows (with *pc* the father chromosome, *mc* the mothers, and *bt* the bloodtype):

$$\begin{aligned}
&pc(p,a) : 0.3 \vee pc(p,b) : 0.3 \vee pc(p,null) : 0.4.\\
&mc(p,a) : 0.3 \vee mc(p,b) : 0.3 \vee mc(p,null) : 0.4.\\
&bt(p,a) : 0.9 \vee bt(p,b) : 0.03 \vee bt(p,ab) : 0.03 \vee bt(p,null) : 0.04\\
&\qquad\qquad\qquad\qquad\qquad\qquad \leftarrow pc(p,a), mc(p,a).\\
&bt(p,a) : 0.03 \vee bt(p,b) : 0.03 \vee bt(p,ab) : 0.9 \vee bt(p,null) : 0.04\\
&\qquad\qquad\qquad\qquad\qquad\qquad \leftarrow pc(p,b), mc(p,a).
\end{aligned}$$

**Theory 4: UWCSE.** The fourth theory is learned from the UWCSE dataset and is not as well-structured as the artificial examples before. The theory contains disjuncts in the head as well as decision trees in the bodies of multiple events.

For theory (a), the BDD based inference methods (cplint and ProbLog) are faster than CVE and VE for large problem instances. CVE and VE are slower partly because they compute the probability that a variable is true, but also for the probability that it is false (separately). It is well known that for Noisy-and,

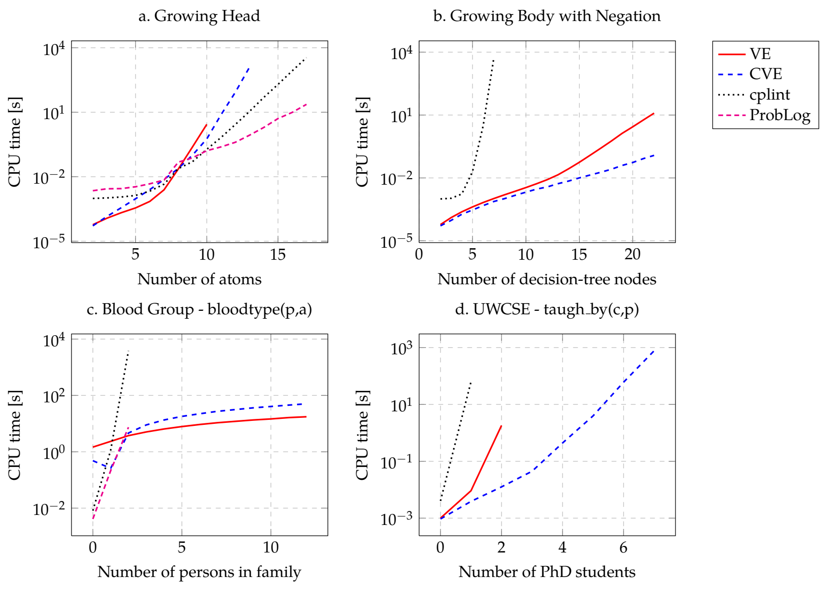


Figure 6.3: Experimental results (including example EBNs for small theories).

which occurs in the ProbLog program that theory (a) is transformed into[2], it is more efficient to only compute the probability $P_T$ that its outcome is true and to calculate the probability that it is false as $1 - P_T$. An advantage of the BDD based methods is that they only compute the probability that an atom is true.

For theories (b)-(d), CVE and VE outperform the BDD based methods. For theory (b) and (d), this is partly due to the complexity of cplint's method for handling negation. A second reason for the inferior performance of the BDD methods is the following: if the same atom is encountered multiple times in the proofs, then the DNF formula will contain an identical subexpression for each occurrence, and computing all these subexpressions will require repeatedly proving the same goal. Some of these redundant computations can be avoided by 'tabling' proofs (Riguzzi 2010; Mantadelis and Janssens 2009). The results for tabling are not included because the ProbLog implementation originally did not support tabling. Graphs (b) and (d) have no results for ProbLog since the ProbLog originally did not support negation.

[2]Noisy-or and noisy-and can all be generalized to the same noisy-max structure. Therefore, ProbLog can handle any of these structures in an efficient way.

CVE outperforms VE for large problem instances on theories (a), (b), and (d). This is due to the compact representation with confactors instead of factors. VE runs out of memory after size 10 in (a) and size 2 in (d).

## 6.3 Multiplicative factorization of noisy-max

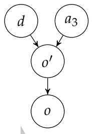
We repeat Figure 6.1.e as Figure 6.4.e to show how noisy-or can be represented in terms of a table that is used by VE. Noisy-or is a causal indepence structure in a Bayesian network where the inputs independently cause the output. Noisy-or is thus a decomposable dependence relation (Lucas 2005). The tabular representation has the disadvantage that, while the inputs independently cause the output, this independence is not reflected in the factorization. Confactors do not offer a solution for this type of local structure, so another technique should be used.

e. VE-factor for $o$ (noisy-or)

| $o$ | $d$ | $a_3$ | |
|---|---|---|---|
| $f$ | $f$ | $f$ | 1 |
| $t$ | $f$ | $f$ | 0 |
| $f$ | $t$ | $f$ | $1-\beta_1$ |
| $t$ | $t$ | $f$ | $\beta_1$ |
| $f$ | $f$ | $t$ | $1-\beta_2$ |
| $t$ | $f$ | $t$ | $\beta_2$ |
| $f$ | $t$ | $t$ | $(1-\beta_1)(1-\beta_2)$ |
| $t$ | $t$ | $t$ | $1-(1-\beta_1)(1-\beta_2)$ |

MF

f. VE-factors after MF for $o$



| $o'$ | $d$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 1 |
| $f$ | $t$ | $1-\beta_1$ |
| $t$ | $t$ | 1 |

| $o'$ | $a_3$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 1 |
| $f$ | $t$ | $1-\beta_2$ |
| $t$ | $t$ | 1 |

| $o'$ | $o$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 0 |
| $f$ | $t$ | $-1$ |
| $t$ | $t$ | 1 |

CVE-OC

g. CVE-OC confactors for $o$

$\langle d = t \wedge o' = f\,, \boxed{1-\beta_1} \rangle$

$\langle a_3 = t \wedge o' = f\,, \boxed{1-\beta_2} \rangle$

$\langle true\,,$

| $o'$ | $o$ | |
|---|---|---|
| $f$ | $f$ | 1 |
| $t$ | $f$ | 0 |
| $f$ | $t$ | $-1$ |
| $t$ | $t$ | 1 |

$\rangle$

Figure 6.4: Repetition of Figure 6.1.e,f,g

Díez and Galán (2003) propose a state-of-the-art multiplicative[3] factorization (MF) for a factor representing noisy-or (and its generalization noisy-max). In this new set of factors, each factor only involves one of the inputs. In general, this leads to faster inference with VE. It is not necessary for this chapter to understand the MF method into depth, but important to note is that this method uses multiple factors to represent the CPD for a variable (for example

[3]The term 'multiplicative' is used because the summation that is typical for noisy-or is transformed into a multiplication causing further factorization.

$o'$ in Figure 6.4.f). Because of the MEE-restriction, these two factors cannot be represented by confactors. In the following we will propose an extension of the CVE algorithm that keeps the advantages but extends it such that methods like multiplicative factorization can be mixed with CVE.

## 6.4 CVE with overlapping contexts

Our main contribution is the CVE-OC algorithm. The CVE-OC algorithm removes the restrictions on the confactors imposed by the CVE algorithm.

First, CVE expects that the confactors for a variable $V$ are MEE. This condition ensures that the parameters in the confactors are identical to those in $V$'s original conditional probability table. It is also a pre-condition for the algorithm that CVE uses to combine confactors while eliminating a variable (the absorption algorithm). As mentioned in the introduction, this pre-condition has certain disadvantages (e.g., it is incompatible with MF (Díez and Galán 2003), and it may make expressing logical constraints more complicated). Therefore, we improve the algorithm so that the pre-condition is no longer necessary. As a result, a parameter in the original table is not guaranteed to be equal to a parameter in a single confactor (like in CVE) but is equal to the multiplication of different parameters found in different confactors with non-mutually-exclusive contexts. As a consequence the absorption algorithm can no longer be used and we need a new technique to decide which confactors to combine when.

Second, the equality tests in the contexts can be replaced with set membership tests. This allows for a more compact representation in domains with multi-valued variables. This representation was already proposed by Poole and Zhang (Poole and Zhang 2003), but not supported in their algorithm and implementation as it requires one to extend the splitting operation.

A confactor $r_i$ now has the following form:

$$\langle v_1 \in \mathbf{v}_{1,i} \wedge \ldots \wedge v_k \in \mathbf{v}_{k,i} \wedge \ldots \wedge v_n \in \mathbf{v}_{n,i}\ , \quad \mathit{factor}_i(v_k, \ldots, v_n, \ldots, v_m) \rangle$$

The context is a conjunction of set membership tests ($v_j \in \mathbf{v}_{j,i}, \mathbf{v}_{j,i} \subseteq \mathrm{dom}(v_j)$, with $1 \leq j \leq n$), which indicates the condition under which the table is applicable. The table stores parameters for given value assignments for a set of zero or more variables ($v_k \ldots v_m$). Note that a variable can now appear both in the context and in the table.

The interpretation of a set of confactors with overlapping contexts for a variable $v$ can be given in terms of the multiplication of their parameters. Given a value assignment for a variable and its parents, it is now possible that multiple confactors for $v$ have contexts that are applicable and each of these confactors has a parameter in its table that is consistent with the value assignment. The product of these parameters is equal to the parameter in the original table representing the CPD. If the set is not exhaustive, we assume that the value assignments not covered by a context correspond to parameters that are equal to 1.0, which are irrelevant in a multiplication.

Based on the above modifications, we can convert the CPDs in Fig. 6.1.c and 6.1.f into the more compact representation in Fig. 6.1.d and 6.1.g, which is the input to CVE-OC. This shows three new uses of confactors: (a) it is possible to use set membership to express value ranges of a variable (e.g., the conditions on $A_2$); (b) noisy-or can be represented more efficiently by using MF (possibly combined with set membership in case of multi-valued variables); (c) logical constraints can be more compactly expressed and will be exploited during variable elimination.

## 6.4.1 The CVE-OC algorithm

Recall from the explanation of CVE that its core operations are not only multiplication and sum-out, but also *compatibility checking* and *splitting*. As explained in Poole and Zhang (2003), compatibility checking and splitting must be performed very often and may therefore be computationally expensive.

Since CVE-OC allows overlapping contexts, it cannot use heuristics based on mutual exclusivity like in the CVE algorithm to reduce the number of compatibility checks. Moreover, the use of set membership tests requires even more types of splitting. To improve the efficiency of compatibility checking and splitting we propose a temporary tree-based index structure to represent the set $R_e$ of all confactors that contain the variable $e$ that is being eliminated.

Fig. 6.5 shows an example of this index structure. Each internal node contains a variable that appears in a context of a confactor in $R_e$ and the outgoing edges of the node are labeled with subsets of the variable's domain. The leaf nodes contain the tables.

Before explaining the index construction procedure, we define the *rank* of a variable. Each variable is given a different rank. $\text{rank}(e) = 0$, and the ranks of the other variables are positive and increase monotonically with the number of confactors they appear in (ties are broken at random). The index will have the property that variables with a higher rank occur higher up in the tree. By placing frequent occurring variables close to the root the tree can be more compact since these variables appear in more branches and branches represent contexts.

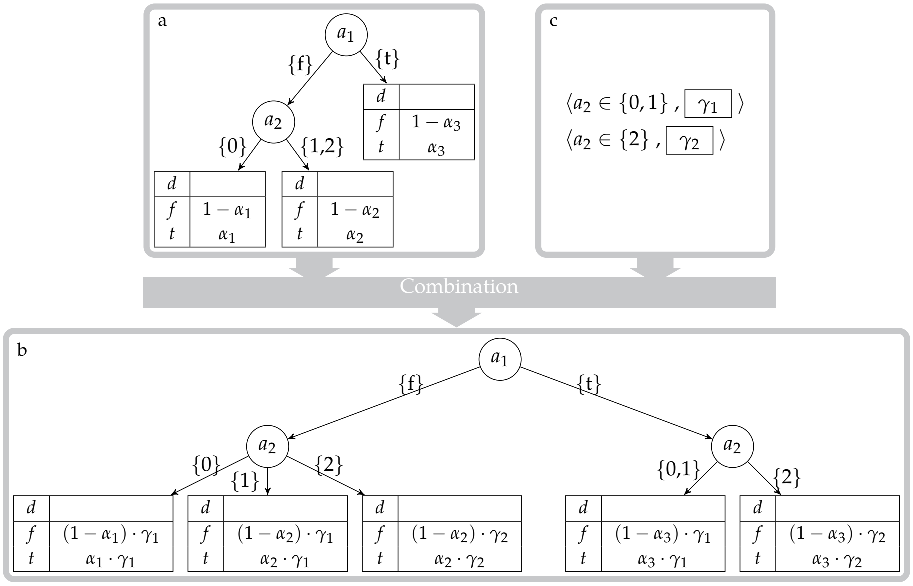

a
$a_1$
{f}
{t}
$a_2$
{0}
{1,2}
$d$ | $f$: $1-\alpha_1$ | $t$: $\alpha_1$
$d$ | $f$: $1-\alpha_2$ | $t$: $\alpha_2$
$d$ | $f$: $1-\alpha_3$ | $t$: $\alpha_3$
c
$\langle a_2 \in \{0,1\}, \boxed{\gamma_1} \rangle$
$\langle a_2 \in \{2\}, \boxed{\gamma_2} \rangle$
Combination
b
$a_1$
{f}
{t}
$a_2$
{0}
{1}
{2}
$a_2$
{0,1}
{2}
$d$ | $f$: $(1-\alpha_1)\cdot\gamma_1$ | $t$: $\alpha_1\cdot\gamma_1$
$d$ | $f$: $(1-\alpha_2)\cdot\gamma_1$ | $t$: $\alpha_2\cdot\gamma_1$
$d$ | $f$: $(1-\alpha_2)\cdot\gamma_2$ | $t$: $\alpha_2\cdot\gamma_2$
$d$ | $f$: $(1-\alpha_3)\cdot\gamma_1$ | $t$: $\alpha_3\cdot\gamma_1$
$d$ | $f$: $(1-\alpha_3)\cdot\gamma_2$ | $t$: $\alpha_3\cdot\gamma_2$

Figure 6.5: (a) Tree index structure used by CVE-OC to eliminate $a_2$, after adding the confactors from Fig. 6.1.d. (b) The tree structure after adding the confactors shown in (c) to (a).

To construct the index, CVE-OC starts with a tree that consists of a single leaf that contains a table $t = 1.0$. Then it absorbs all confactors from $R_e$ one by one into the tree. In the end, the tree will contain for any context a correct CPD. To absorb a confactor $r = \langle c, t \rangle$, it moves the confactor down the tree. When it encounters an internal node containing variable $n$, one of the following five cases may occur (CVE-OC acts according to the first case that applies).

1. $c$'s top-ranked variable $v$ has a higher rank than rank$(n)$, i.e., $v$ should appear above $n$ in the tree. Let $v \in \mathbf{v_v}$ be the term about $v$ in the context $c$. CVE-OC then replaces node $n$ by a new node labeled $v$ with two outgoing

edges: one labeled $\mathbf{v_v}$, and one labeled with its complement $(\text{dom}(v) - \mathbf{v_v})$. The original subtree rooted at $n$ is duplicated and becomes both the left and right subtree of the new node $v$. CVE-OC removes variable $v$ from $r$'s context $c$ and sorts the resulting confactor down both subtrees.

2. $n$ appears in $r$'s context, i.e., $(n \in \mathbf{v_n}) \in c$. If one of the outgoing edges from $n$ is labeled $\mathbf{v_n}$, then CVE-OC sorts $r$ down that branch. If not, then it must perform the splitting operation. To this end, it processes all $n$'s outgoing edges in turn. For a given edge $i$, if its label $\mathbf{v_{n,i}}$ is disjoint from $\mathbf{v_n}$, the corresponding subtree is incompatible with $r$ and no further computation is required. Otherwise, $\mathbf{v_{n,i}}$ must be split. CVE-OC removes the edge labeled $n\mathbf{v_{n,i}}$ from $n$ and replaces it with two new edges, one labeled $\mathbf{v_{n,i}}^+ = \mathbf{v_{n,i}} \cap \mathbf{v_n}$ and one labeled $\mathbf{v_{n,i}}^- = \mathbf{v_{n,i}} - \mathbf{v_n}$. The original subtree rooted at $\mathbf{v_{n,i}}$ is duplicated below these two new edges. Next, CVE-OC removes $n$ from $r$'s context and projects its table on the condition $n \in \mathbf{v_{n,i}}^+$. Then it sorts the resulting confactor down edge $\mathbf{v_{n,i}}^+$.

3. $n$ appears in $r$'s table. Let $\mathbf{v_{n,1}}, \ldots, \mathbf{v_{n,k}}$ be the labels of $n$'s outgoing edges. CVE-OC sorts $r$ down the tree via each $\mathbf{v_{n,i}}$ after projecting $r$'s table on the condition $n \in \mathbf{v_{n,i}}$.

4. $r$ represents a logical constraint ($t = 0$) and $c = \emptyset$. The context is empty, and the table consists of the single value 0, hence applying the confactor means multiplication by 0. In this case, CVE-OC replaces node $n$ by a new leaf and initializes the table in that leaf to zero. We call this step pruning the tree based on a constraint.

5. If $n$ does neither appear in the context $c$ nor in the table $t$, then the confactor $\langle c, t \rangle$ is sorted down all edges of the internal node.

If $r$ reaches a leaf, which stores a table $t_l$, then the following happens. If $t_l = 0$, then the value remains 0 (multiplication by 0) and no further computation is required (this leaf represents a constraint). If $t_l \neq 0$, then there are two cases: either $c = \emptyset$ or $c \neq \emptyset$. In the former case, CVE-OC replaces the table in the leaf by the factor multiplication of $t_l$ and $t$. In the latter case, it must introduce a new node for the top ranked variable in $c$ into the tree. This is done in precisely the same way as in step one above.

After all confactors in $R_e$ are absorbed into the index, CVE-OC can sum-out the variable $e$. This is done in two steps. In the first step, $e$ is summed out from all the tables in the leaves of the index. The next step applies to all internal nodes that are labeled with $e$. Because $\text{rank}(e) = 0$, all children of such nodes are leaves. To sum-out $e$, a node that contains $e$ is simply replaced by a leaf with a table equal to the factor sum of the tables in $e$'s children.

After $e$ is eliminated, the tree is converted back into a set of confactors, by creating one confactor for every leaf. Together with the confactors not in $R_e$, these form the set of confactors for the following elimination step. Note that we cannot reuse the same index for eliminating a different variable because the index is specific to the variable that is being eliminated. Also, converting all confactors into one single tree is to be avoided as a tree is in general not the most compact representation for a set of confactors. Therefore, we include as few confactors as possible in the tree (by means of the variable ordering), and we only use the tree to make compatibility checking and splitting more straightforward.

## 6.5 Experiments

We evaluate the inference methods on the task of inferring the marginal distribution of one designated variable in four types of BNs of varying complexity. We again select the variable with the highest inference cost and do not include any evidence (i.e., we consider the most difficult case). The software and BNs are available online (dtai.cs.kuleuven.be/corporal). We compare five algorithm/input combinations: VE, VE with multiplicative factorization (VE+MF), CVE$^{\in}$, CVE-OC, and CVE-OC with multiplicative factorization (CVE-OC+MF). CVE$^{\in}$ is our own C++ implementation of CVE; it is the second algorithm described in (Poole and Zhang 2003) (which uses absorption), extended with set membership tests (so we can accurately assess the contribution of the overlapping confactors). CVE-OC uses exactly the same input (confactors) as CVE$^{\in}$; CVE-OC+MF has additional confactors for noisy-or/and structures. We use the minimum deficiency elimination ordering (Bertelè and Brioschi 1972), which is a simple greedy heuristic that performs well for VE. Fig. 6.6 presents the results. Additionally, we have added in the graphs results obtained using the ACE system (Chavira and Darwiche 2007), which is a representative and state-of-the-art compilation-based approach. Version 2.0 is used, with default settings. The input consists of conditional probability tables except for noisy-or/and nodes which are encoded using the noisy-max syntax.

The BNs in (a) and (b) are constructed from artificial CP-logic theories (more details were given in Section 6.2.4). The BNs in (a) only include interconnected noisy-or and noisy-and nodes with a linearly increasing number of parents. VE runs out of memory when the number of noisy-or nodes is larger than 8, while CVE can handle larger BNs because of its compact confactor representation. VE+MF and CVE-OC+MF are much faster than the other methods because they efficiently factorize the noisy-or nodes. For experimental comparison of MF with other methods for noisy-or networks we refer to (Díez and Galán 2003; Savicky and Vomlel 2007). The ACE system also exploits the presence of noisy-or/and nodes but is a factor 100 slower because of the compilation where it optimizes for all variables. After crossing the curves for non-MF methods, the ACE-curve stops because the tables used as input become too large to generate. The network in (b) contains many CPDs structured as decision trees with a linearly increasing number of parents. CVE and CVE-OC are well suited to handle such a representation and are therefore faster than VE (MF has no influence). The ACE system seems to be unable to fully exploit the interconnected decision tree structures in the CPDs and is also slower because it tries to optimize the structure for all variables.

The BNs in (c) are constructed from a CP-theory that was learned from the UW-CSE dataset [4]. The UW-CSE BNs include all structures also included in (a) and (b). For such networks CVE-OC+MF excels as it can efficiently represent all these structures. The other methods run out of memory because they cannot represent one of the local structures efficiently. The noisy-or/and nodes are no problem for the ACE system and when the decision tree structures do not interact too heavily it can handle them efficiently. The curve for CVE-OC+MF fluctuates because the heuristic used for the elimination ordering is agnostic about local structure when combining different structures. A heuristic that takes local structure better into account would be an interesting future research topic.

The networks in (d) are the randomized networks used in Fig. 11 of (Poole and Zhang 2003) and created with the original Java code available from the author. This experiment compares CVE$^{\in}$ with the new CVE-OC algorithm on the same data as Poole and Zhang used. This shows that CVE-OC, although more generally applicable, is about as fast as CVE$^{\in}$ even when there are no additional structures to be exploited.

[4]`http://alchemy.cs.washington.edu/data/uw-cse/`

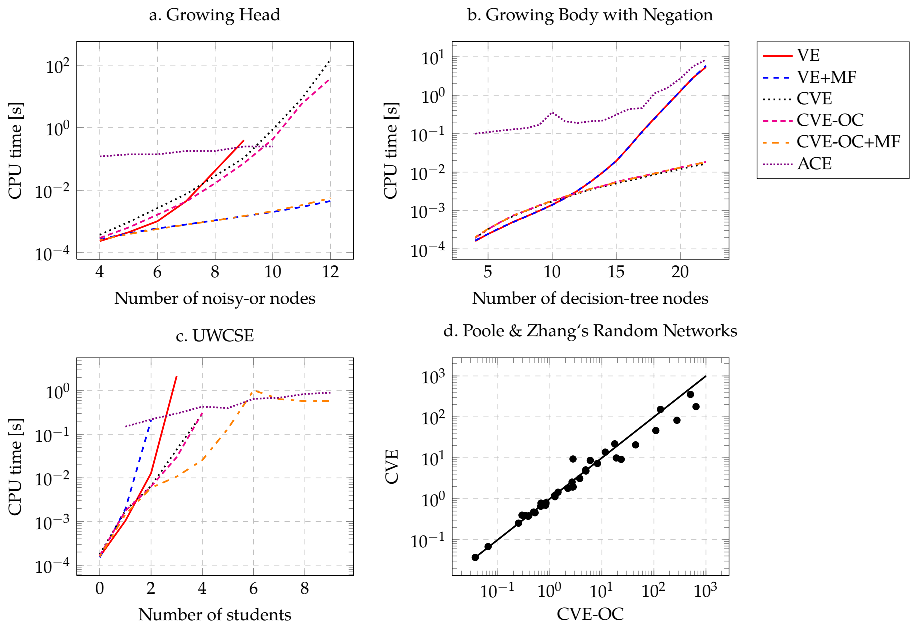


Figure 6.6: Inference times for networks originating from CP-theories and random networks. The horizontal axis of (a)-(c) indicates BN complexity.

## 6.6 Conclusions

We have applied CVE as an inference method for CP-logic and compared it to other CP-logic inference methods that are based on BDDs. Depending on the theory, CVE may be faster than the current BDD based methods. CVE performed poorly on local structures like noisy-or which occur often in probabilistic logic models. Therefore, we improved the CVE algorithm to also cope with such structures.

We presented the algorithm CVE-OC (CVE with overlapping contexts), which extends contextual variable elimination (CVE). The introduction of overlapping contexts is a simple but powerful step. From the representation point of view, it offers an elegant combination of deterministic and probabilistic knowledge. From the computational point of view, the need for equality testing is reduced, invalid combinations of values are pruned, and the loosened restrictions allow for better optimizations. CVE-OC generalizes over both CVE and MF, and provide optimization opportunities beyond the union of what those methods offer.

The experiments show that CVE-OC, while more generally applicable, can handle input for CVE without any loss in efficiency. Because of its generality, the input for CVE-OC can be more compact than for CVE and other known optimization methods for VE can be integrated. For example, the integration of MF is shown to be faster and more compact than only VE, CVE or MF. The ACE systems has an overhead because of the compilation optimizes more than necessary (e.g., for all possible queries). This makes the ACE system slower for small theories, but for larger theories the time for inference becomes larger than that of the compilation and inference becomes the main issue for efficiency. For the UWCSE experiment (Figure 6.6.c) ACE and CVE-OC+MF perform equally well. For the growing bodies theories, ACE has difficulties as the number of decision trees increases. Since ACE does not allow to input decision trees directly, it needs to detect these local structures itself and this turns out to be difficult causing ACE to run slower than CVE-OC+MF.

# 7 Learning

## 7.1 Introduction

This chapter investigates the learning of CP-theories from training data. It makes the following two contributions: (a) it proposes a new method for learning CP-theories, and (b) it presents a theoretical and experimental comparison between CP-theory and BN learning.

The first contribution of this chapter is SEM-CP-Logic, an algorithm for learning CP-theories. SEM-CP-Logic uses BN learning techniques to learn a BN that is equivalent to a CP-theory and then applies the inverse transformation to obtain the resulting CP-theory. More specifically, it is a modified version of Friedman's structural EM algorithm (SEM) (Friedman 1997) for learning BNs. This idea of using greedy structure search combined with a parameter learning method has also been used for learning other probabilistic logics, including Bayesian Logic Programs (Kersting and De Raedt 2008) and Markov Logic Networks (Richardson and Domingos 2006). The differences are that SEM-CP-Logic uses a parameter learning method that is specialized to learn CP-theory parameters, and that it modifies the structure search such that any BN encountered during the search corresponds to a valid CP-theory. To this end, SEM-CP-Logic uses a refinement operator that is specific to CP-theories.

The second contribution is a comparison between CP-theory and BN learning. We already compared CP-theories and BNs at a theoretical level in Chapter 4.2. With this theoretical comparison in mind, we present experiments comparing CP-theory and BN structure and parameter learning in a controlled artificial domain. BNs may need to learn more complex structures with more parameters, which is detrimental to the interpretability of the resulting model and to the efficiency of the learning. This is supported by our evaluation, which shows that CP-theory learning requires less training data compared to BN learning in such cases. Next to these controlled experiments, we illustrate the applicability of SEM-CP-logic by testing it on an application in medical research targeting the Human Immunodeficiency Virus (HIV).

### 7.1.1 Bibliographical note

Riguzzi (2004)[1] presented a method that was able to learn a subset of CP-logic. This subset was later extended to any non-recursive CP-theory with a finite Herbrand universe by introducing a transformation from CP-theories to BNs in

> H. Blockeel and W. Meert (2006). "Towards learning non-recursive LPADs by transforming them into Bayesian networks". In: *Proceedings ofthe 15th International Conference on Inductive Logic Programming (ILP)*. (Bonn, Germany, Aug. 10–13, 2005). Volume 3625. Lecture Notes in Computer Science, pages 94–108

In this paper, we mainly focused on the transformation itself and only briefly discussed how a CP-theory learning algorithm can be constructed based on this transformation. This work was extended with a concrete approach to learning propositional CP-logic theories in two follow-up papers on which this chapter is mainly based:

> W. Meert, J. Struyf, and H. Blockeel (2007). "Learning ground CP-logic theories by means of Bayesian network techniques". In: *Proceedings ofthe 6th workshop on Multi-Relational Data Mining (MRDM)*. (Warsaw, Poland, Sept. 17, 2007), pages 93–104

[1]Note that CP-logic theories are equivalent to Logic Programs with Annotated Disjunctions (LPADs) (Vennekens, Verbaeten, and Bruynooghe 2004). The research about LPADs has evolved into CP-logic.

W. Meert, J. Struyf, and H. Blockeel (2008b). “Learning ground CP-logic theories by leveraging Bayesian network learning techniques.” In: *Fundamenta Informaticae* 89(1), pages 131–160

### 7.1.2 Structure of this chapter

In Chapter 7.2 we introduce a method to learn the parameters of a CP-theory. In Chapter 7.3 we then focus on the learning of the structure. Given these two components we apply structure learning for CP-logic on artificial and real datasets and compare the results with structure learning for Bayesian networks in Chapter 7.4.

## 7.2 Parameter learning

Once a given CP-theory is transformed into its EBN, we can use known BN parameter learning methods to learn the parameters of this network and the CP-theory parameters can then be trivially extracted from the learned CPTs.

Note that the parameter learning algorithm does not need to learn values for all the parameters in the CPTs of the EBN. Most parameters are fixed to certain values (either 0.0 or 1.0) by the transformation (Section 3.6). If the parameter learning algorithm would change these values, then the network would no longer be equivalent to the given CP-theory. Therefore, only the CP-theory parameters, which appear in the CPTs of the choice nodes, need to be learned. Thus, the CPTs contain two types of parameters: the parameters in the original CP-theory and the parameters that are set to 0.0 or 1.0 to encode the extra structure in a CP-theory.

Each training example only assigns values to the atom nodes. The choice nodes are unobserved. We therefore use expectation maximization (EM) (Dempster, Laird, and Rubin 1977; Baum et al. 1970), which is especially designed to handle unobserved nodes, to learn the CP-theory parameters in the choice nodes. An additional advantage of using EM is that this also allows for missing values in the training data.

To force the CPTs to have ones and zeros in the right positions after learning, it suffices to initialize these parameters to 0.0 or 1.0. The Bayesian update rule in the expectation step of the EM algorithm can only update values strictly between

0.0 and 1.0: if the prior probability is set to 0.0 or 1.0 then the posterior probability will also be 0.0 or 1.0. Hence, the standard EM parameter learning algorithm learns the correct parameters for our CP-theory after proper initialization. We initialize the CP-theory parameters to $1/(n+1)$, with $n$ the number of head atoms. The particular parameter learning algorithm that we use is the one implemented in the Open Bayes toolbox (Gaitanis 2007).

### 7.2.1 Complexity

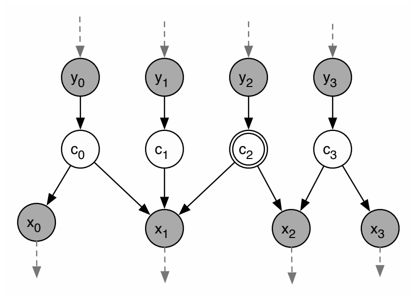
In this section, we motivate why we chose to use a general BN parameter learning method and discuss the complexity of CP-theory parameter learning.

Since noisy-OR is a popular combining rule in other probabilistic logic formalisms, such as BLPs, parameter learning for noisy-OR as a combining rule has already been studied. For example, Jaeger (2007) and Natarajan et al. (2006) propose specific methods to learn noisy-OR parameters that are based on EM or gradient descent and Vomlel (2006) proves convergence of noisy-or parameter learning for EM. These particular methods are not directly applicable to CP-logic. Here, the situation is more complex because of the mutually exclusive consequences (we illustrate this below with an example). This makes the use of gradient descent not straightforward, and for this reason we chose to use an EM algorithm to learn the parameters. Because the inference formulas that appear in the expectation step are complex and depend heavily on the structure of the CP-theory (as we will show below), we have not yet investigated possible optimizations, and resort to a general EM algorithm for BN parameter learning (Gaitanis 2007).

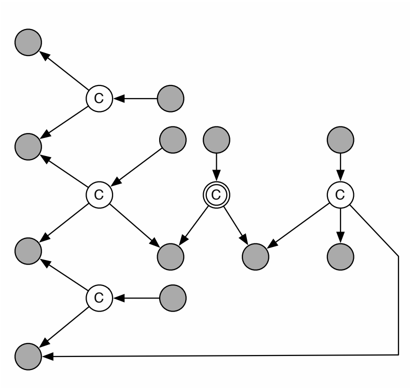
We illustrate the complexity of the inference that occurs during CP-theory parameter learning with the example shown in Fig. 7.1.a. This network has a specific structure; we do not consider the general case, but it illustrates some of the issues with inference. In the given structure, every choice node $c_i$ has at most one common consequence $x_i$ with another rule, and the causes $y_i$ are not related.

Suppose that we want to learn the parameters for the CP-event represented by choice node $c_2$, given a data set $D$ containing $N$ data points $d_i$ in which all atom nodes have particular values assigned. That is,

$$D = \{d_i \mid 0 \leq i < N\} = \{\{x_0^{d_i}, \ldots, x_3^{d_i}, y_0^{d_i}, \ldots, y_3^{d_i}\} \mid 0 \leq i < N\}\,. \tag{7.1}$$



(a) EBN structure.



(b) Inference example where the Markov blanket contains a cycle.

Figure 7.1: Illustration of the inference during parameter learning (the gray nodes are the observed atom nodes, and the double circled choice node is the query node).

To estimate the missing values of $c_2$, we need to perform inference for every data point. This means that we have to calculate the probability $Pr(c_2|y_2, d_i)$, which is proportional to $Pr(c_2, y_2|d_i)$. To do so, it is sufficient to know the partial network shown in Fig. 7.1.a because that includes the Markov blanket (Neapolitan 2003). The inference formula for this case can be found by applying the rules of probability theory, and is the following:

$$\begin{aligned}
Pr(c_2, y_2 \mid d_i) \quad = \quad & \frac{1}{Z} \cdot Pr(y_2 \mid y_2^{d_i}) \\
\cdot \quad & \sum_{c_0,c_1} Pr(x_1^{d_i} \mid c_0, c_1, c_2) \cdot Pr(x_0^{d_i} \mid c_0) \cdot Pr(c_0 \mid y_0^{d_i}) \cdot Pr(c_1 \mid y_1^{d_i}) \\
\cdot \quad & \sum_{c_3} Pr(x_2^{d_i} \mid c_2, c_3) \cdot Pr(x_3^{d_i} \mid c_3) \cdot Pr(c_3 \mid y_3^{d_i}) \\
\cdot \quad & Pr(c_2 \mid y_2^{d_i})
\end{aligned} \tag{7.2}$$

Because the $x_i$ variables are observed in the data, $c_i$ nodes that influence the same variable $x_i$ are dependent. This results in a chain of dependencies between choice nodes that share consequences. In Eq. 7.2, it can be seen that to calculate $Pr(c_2, y_2|d_i)$, we also need to take $c_0$, $c_1$ and $c_3$ into account. This leads to the multiplication of summations; if a consequence is shared with multiple choice nodes, we need to sum over all the possible combinations of the values for those choice nodes. This results in a complexity that increases exponentially with the number of choice nodes that share a common consequence. A similar effect occurs in the case of noisy-OR nodes and optimizations that have been proposed in that context, such as by Vomlel (2006), are also applicable for this special case.

The example in Fig. 7.1.a only considers a specific CP-theory. In general, the structure of a CP-theory may lead to more complex dependencies between the choice nodes because of the shared consequences. This leads to more complex inference. For example, in Fig 7.1.b, the Markov blanket contains a non-directed cycle, which makes inference harder (Neapolitan 2003). Therefore, we do not try to infer specific inference formulas for CP-theory parameter learning, but apply a general BN inference procedure instead (Gaitanis 2007).

Such a general inference procedure can be further optimized by exploiting the specific structure of the CPTs that appear in the EBN of a CP-theory. The CPTs of the atom nodes, which only contain the values 0.0 and 1.0, do not describe a true probability distribution but a *functional dependency*: for a given input, there is only one possible output and this output is a deterministic function of the input. Because the EBN contains many such functional dependencies, optimizations such as CVE-OC (see Chapter 6) prove to be useful in such cases.

As pointed out above, we use the BN inference algorithm implemented in the Open Bayes toolbox (Gaitanis 2007), which does not include such optimizations. With this implementation, CP-logic parameter learning is 50 (in *Shop1*, see Section 7.4.1) to 1000 (in *Shop2*) times slower than computing the parameters of the corresponding fully observed (and almost fully connected) BN. In the latter experiment, the total execution time of the CP-theory parameter learning was about one minute for a data set with 1000 examples on a 2.6GHz Intel Core 2 Duo processor with 2GB RAM.

## 7.3 Structure learning

Besides learning the parameters, we also need to learn the structure of the CP-theory. This involves a search over possible CP-theories.

### 7.3.1 Structural EM and the BIC Score

The learning algorithm that we propose is based on *structural EM* (SEM), which was introduced by Friedman (Friedman 1997) to learn the structure of a BN in the presence of unobserved nodes or missing values. SEM performs a greedy hill-climbing search through the space of possible network structures and finds the network that (locally) maximizes a given evaluation score, such as the *Bayesian Information Criterion* (BIC) (Neapolitan 2003). This section briefly describes SEM and the BIC score and the next sections adapt these to learn CP-theories.

The greedy search implemented by SEM (Algorithm 4.a) starts from an initial network, which is either a network without any edges or a network with a random edge structure. In each main loop iteration it computes all possible refinements of the current network and then selects the refinement that maximizes the evaluation score. A refinement is obtained either by adding a new edge to the current network, or by deleting one of the edges from the current network. The algorithm ends and returns the current network if no refinement improves the score. To avoid local maxima, the search algorithm can be restarted from different random networks and the result is then the best network found during the different runs.

**Algorithm 4** (a) SEM learns the structure of a BN from a given data set $D$. (b) A similar algorithm can be used to learn the structure of a CP-theory. The main difference is that the initial network represents a valid CP-theory and that the refinement operator 'Refinements-CP-Theory' only generates networks that are valid CP-theories.

```
procedure SEM($D$)
1: $i := 0$
2: $\text{BN}_i :=$ initial BN
3: repeat
4:     $i := i + 1$
5:     $R :=$ Refinements$(\text{BN}_{i-1})$
6:     $R' :=$ Update-params$(R, D)$
7:     $\text{BN}_i := \text{argmax}_{\text{bn} \in R'}$ Eval$(\text{bn}, D)$
8: until Eval$(\text{BN}_i, D) <$ Eval$(\text{BN}_{i-1}, D) + \epsilon$
9: return $\text{BN}_{i-1}$

procedure SEM-CP-logic($D$)
1: $i := 0$
2: $\text{BN}_i :=$ BN(initial CP-theory)
3: repeat
4:     $i := i + 1$
5:     $R :=$ Refinements-CP-Theory$(\text{BN}_{i-1})$
6:     $R' :=$ Update-params$(R, D)$
7:     $\text{BN}_i := \text{argmax}_{\text{bn} \in R'}$ Eval$(\text{bn}, D)$
8: until Eval$(\text{BN}_i, D) <$ Eval$(\text{BN}_{i-1}, D) + \epsilon$
9: return CP-theory$(\text{BN}_{i-1})$
```

The evaluation score is usually based on the likelihood of the data given the network structure and the parameters in the CPTs. Therefore, in order to evaluate a given candidate structure, its parameters must be learned first. Recall from Section 7.2 that parameter learning in the presence of unobserved nodes is implemented as an EM algorithm. This computationally expensive EM run must be repeated for each candidate network. The strength of the SEM algorithm is that it avoids this repeated parameter learning by making the parameter learning incremental: to learn the parameters for a given refinement it copies the parameter values from the original network and only relearns the parameters of the modified part. Details of this optimization can be found in (Friedman 1997).

A frequently used evaluation score for BN structure learning is the *Bayesian Information Criterion* (BIC) (Neapolitan 2003), which is defined as

$$\mathrm{BIC} = \ln(L) - \frac{d}{2} \cdot \ln(N)\,, \tag{7.3}$$

where $L$ is the likelihood of the data given the network (and learned parameters), $d$ the number of free parameters in the CPTs of the network, and $N$ the number of training examples.

BIC trades off the fit to the data (the likelihood) versus the complexity of the network. That is, it biases the search to simple network structures, or, when applied to CP-theory learning, to simple CP-theories. Given that the syntax of CP-logic allows for a compact representation of causal processes, we can apply Occam's Razor and assume that, among different CP-theories with equal likelihood, the smallest theory is the one that correctly represents the structure of the causal process that generated the data (this assumption is also made by Pearl (Pearl 2000)). BIC's bias for small theories will therefore result in a bias in favor of discovering the correct causal structure. Note that this is only a heuristic; it does not guarantee to produce the correct structure. In fact, Pearl (Pearl 2000) has shown that for some causal processes it is impossible to discover the correct causal structure from a data set.

### 7.3.2 Structure Search for CP-Theories

Algorithm 4.b shows how we modify the greedy search of SEM to learn CP-theories. The search still occurs in BN space, but the SEM-CP-logic algorithm ensures that each network encountered during the search represents a valid CP-theory. This requires the following three modifications:

1. The initialization is such that the initial BN represents a valid CP-theory. In particular, SEM-CP-logic starts from the CP-theory that has one rule for each atom in the domain with the atom in the head and an empty body. This ensures that each atom has a cause (atoms without a cause cannot appear in the data). The corresponding BN is shown in Fig. 7.2.b on the left. Another possibility would be to repeatedly start from random initial CP-theories. We here use the former strategy because it is computationally more efficient and proved to be sufficient in our experiments.

2. The refinement operator only generates refinements that represent valid CP-theories. We discuss the refinement operator that we use to this end in detail in Section 7.3.3.

3. SEM-CP-logic initializes the parameters of the networks as described in Section 7.2 before starting the EM part of the algorithm. This ensures that the CPTs of the atom and choice nodes have the required structure.

We also modify the BIC score. We do not set $d$ to the total number of parameters in the CPTs of the EBN because most of these parameters are fixed by the transformation. Instead, we set $d$ to the number of edges in the EBN. Note that the number of edges is equal to the number of CP-theory parameters plus the number of conditions in the rule bodies. One can argue that this is the *true* number of parameters of the model.

### 7.3.3 Refinement Operator

The refinement operator that we propose is similar to that of SEM in the sense that it also adds edges to or removes edges from the current network. In addition, it also needs to be able to introduce new choice nodes to create new rules. It also takes the following constraints on the structure of the network into account to ensure that it represents a valid CP-theory (see Section 3.5.2 for the equivalent Bayesian network representing a CP-theory):

- Each node is either an atom node or a choice node (as defined previously).
- Each edge is annotated to be either positive or negative (representing a negated body atom). They are both regular edges in a Bayesian network, but this extra knowledge is used for learning.
- Edges are only allowed between an atom node and a choice node.
- Negative edges go from an atom node to a choice node.
- Each atom node has at least one incoming edge.
- The CPTs are structured as described in Section 3.5.2.

Algorithm 5 shows the resulting refinement operator. It performs five basic actions: delete a literal from a rule, add an atom to a rule head, add a literal to

**Algorithm 5** The CP-logic refinement operator. The structure of a CP-theory is represented as a BN $(A, C, E)$, with $A$ the atom nodes, $C$ the choice nodes, and $E$ the directed edges. An edge to a negative body atom is indicated with a dotted arrow '$\dashleftarrow$'.

```
procedure Refinements-CP-Theory(BN)
 1: $(A, C, E) :=$ BN; $R := \emptyset$
 2: for each edge $e \in E$ do
 3:     $R := R \cup \{\text{DeleteEdge}(e, A, C, E)\}$
 4: for each atom node $a_1 \in A$ do
 5:     for each choice node $c \in C$ do
 6:         $R := R \cup \{(A, C, E \cup \{a_1 \leftarrow c\})\}$            // Add atom to rule head
 7:         $R := R \cup \{(A, C, E \cup \{c \leftarrow a_1\})\}$       // Add pos. atom to rule body
 8:         $R := R \cup \{(A, C, E \cup \{c \dashleftarrow a_1\})\}$      // Add neg. atom to rule body
 9:         $(c', E') = \text{CloneRule}(c, E)$          // Clone rule and negate condition
10:         $R := R \cup \{(A, C \cup \{c'\}, E' \cup \{c \leftarrow a_1, c' \dashleftarrow a_1\})\}$
11:     for each atom node $a_2 \in A$ do
12:         Create four new choice nodes $c_1, \ldots, c_4$
13:                                         // New rule with pos. body atom
14:         $R := R \cup \{(A, C \cup \{c_1\}, E \cup \{a_1 \leftarrow c_1, c_1 \leftarrow a_2\})\}$
15:                                         // New rule with neg. body atom
16:         $R := R \cup \{(A, C \cup \{c_2\}, E \cup \{a_1 \leftarrow c_2, c_2 \dashleftarrow a_2\})\}$
17:         $R := R \cup \{(A, C \cup \{c_3, c_4\},$
18:                 $E \cup \{a_1 \leftarrow c_3, c_3 \leftarrow a_2, a_1 \leftarrow c_4, c_4 \dashleftarrow a_2\})\}$
19: for each choice node $c \in C$ do
20:     $R := R \cup \{\text{InvertRule}(c, A, C, E)\}$
21: return $R$
```

a rule body, create a new rule, and invert a rule. In the following we describe these actions in more detail.

## Delete a Literal from a Rule

Deleting an edge either corresponds to deleting an atom from a rule head or to deleting a literal from a rule body. It is implemented by removing the edge between the corresponding atom node and choice node. If the choice node has no outgoing edges left after removing the edge, the choice node itself is also deleted. The refinement operator never removes the last incoming edge of an atom node.

Fig. 7.2.a illustrates this operation. The first step in the figure shows how a rule can be eliminated by removing a choice node and its edges. The second step removes a literal from the body of a rule by deleting an incoming edge of the choice node that represents the rule.

### Add a Literal to a Rule

Adding an edge between a choice node and an atom node corresponds to adding a literal to the corresponding rule of the CP-theory. To add an atom to the head of a rule, an edge is created departing from the choice node representing that rule to the atom node representing the added atom. Adding a literal to the body of a rule is accomplished by adding an edge from the literal's atom node to the choice node. The new edge can be either a positive edge (represented with '$\longleftarrow$'), or it can be a negative edge (represented with '$\dashleftarrow$') if the new body atom is negated.

The first step of Fig. 7.2.b adds the literal $x$ to the body of the second rule. The equivalent step in BN space is adding an edge from the atom node representing $x$ to the choice node representing the second rule.

Adding a literal to the body of a rule may result in a CP-theory with likelihood zero. Before we discuss how this can be avoided, we first explain in more detail why it happens. An atom in a CP-theory can only become true if it has a *cause* to become true, i.e., if the body of a rule where the atom appears in the head is true. Suppose that a given atom is only present in the head of one rule. This is true for all atoms in the CP-theory on the left of Fig. 7.2.b. Let us focus on the second rule $y : \alpha_2$. Atom $y$ can become true independent of the other atoms (because the rule's body is always true). After adding a literal to the rule body ($x$ after step 1), the head can only become true if the body $x$ is true. Suppose that in the target theory, $y$ can also be true if $x$ is false ($y$ has multiple causes). The current theory will then have zero likelihood, because the data will contain cases where $y$ is true and $x$ false. The new body of the rule constrains the head too much. On the other hand, we may want to have a relation between $x$ and $y$ with $x$ being one of the possible causes for $y$.

To check if the new body is a cause of the head, but not the only one, we perform a trivial *lookahead* step: we generate a refinement that is identical to the previous one, but includes in addition a copy of the given rule with the newly added atom negated (Line 10 of Fig. 5). Copying the rule is implemented by the CloneRule

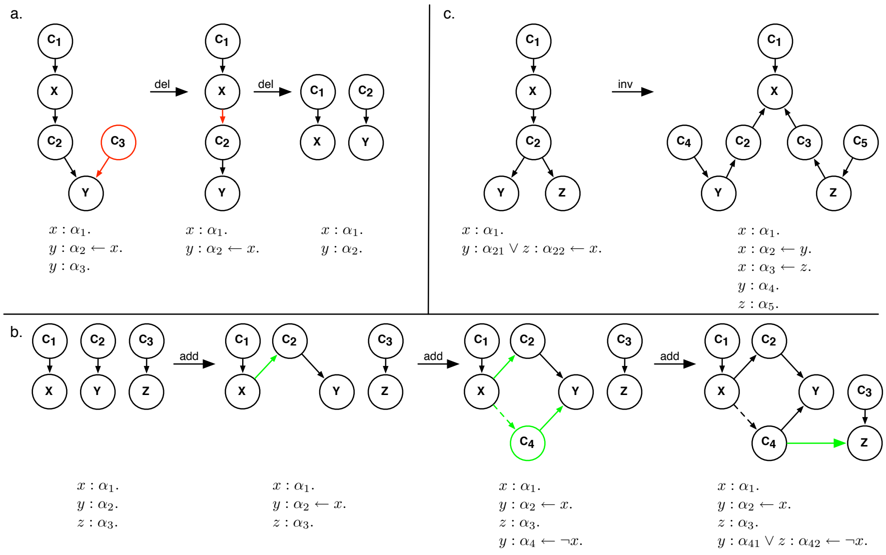

a.
del
del
c.
inv
b.
add
add
add
$x : \alpha_1.$
$y : \alpha_2 \leftarrow x.$
$y : \alpha_3.$
$x : \alpha_1.$
$y : \alpha_2 \leftarrow x.$
$x : \alpha_1.$
$y : \alpha_2.$
$x : \alpha_1.$
$y : \alpha_{21} \vee z : \alpha_{22} \leftarrow x.$
$x : \alpha_1.$
$x : \alpha_2 \leftarrow y.$
$x : \alpha_3 \leftarrow z.$
$y : \alpha_4.$
$z : \alpha_5.$
$x : \alpha_1.$
$y : \alpha_2.$
$z : \alpha_3.$
$x : \alpha_1.$
$y : \alpha_2 \leftarrow x.$
$z : \alpha_3.$
$x : \alpha_1.$
$y : \alpha_2 \leftarrow x.$
$z : \alpha_3.$
$y : \alpha_4 \leftarrow \neg x.$
$x : \alpha_1.$
$y : \alpha_2 \leftarrow x.$
$z : \alpha_3.$
$y : \alpha_{41} \vee z : \alpha_{42} \leftarrow \neg x.$

Figure 7.2: Examples of the refinement operator. The dotted arrow represents a rule where that particular atom is negated in the body of the rule.

function in Algorithm 5, which creates a copy of the given choice node and of the edges in which it participates. The new rule with the negated condition covers the causes not yet discovered, be it in a rudimentary way. In a subsequent step, the algorithm can find another cause for the head and add this to the rule body with the negated atom. It can then optionally remove this negated atom later on. Adding the copy of the rule with the negated body is illustrated in step 2 of Fig. 7.2.b. Note that the algorithm considers step 1 and step 2 together as one single refinement.

Although the refinement generated by the lookahead step is equivalent to the original rule from a logical point of view, this is not the case if we consider the attached probabilities. Take for example these two theories:

| 1 | 2 |
|---|---|
| $y : \alpha \leftarrow .$ | $y : \beta_1 \leftarrow x$ |
| | $y : \beta_2 \leftarrow \neg x$ |

Seen as purely logic theories, both are equivalent: whatever the truth value of

$x$ is, $y$ will be true. However, from a probabilistic point of view, both theories are different and may have a different likelihood. Suppose that we have a data set where if $x$ is true, there is a high probability that $y$ is also true, but when $x$ is false, this probability is much lower. (We assume that there is also a rule $x : \gamma \leftarrow .$ that can make $x$ true.) To calculate the likelihood we multiply the probabilities of examples. Since we know that there are more examples where $x$ and $y$ are true than there are examples where only $y$ is true, we want the first kind to have a high probability and the second kind to have a low probability. This diversification is not possible in the first theory, but is possible in the second one. Therefore, in this case, the likelihood of the second theory will be better than that of the first one.

#### Create a New Rule

To be able to create CP-theories with more rules than atoms in the domain, the refinement operator needs to be able to create new rules. The new rules represent simple causes with one atom in the head and one literal in the body. The refinement operator considers creating such a rule for each pair of atoms. In addition to adding a single rule, it also considers adding two rules at once: one with a positive body atom and one with the body atom negated. The rationale behind this lookahead step is the same as the one for adding cloned rules with the added condition negated.

#### Invert a Rule

Inverting a rule corresponds to switching the direction of the causation. It is sometimes possible that the algorithm initially learns the relationship between atoms with the wrong direction of causation. Because the algorithm builds further upon already learned networks, the incorrect direction of causation may persist when subsequent relationships are added. And, although the initial relationship by coincidence had a good likelihood, the following steps may be suboptimal. The 'invert rule' operator can detect such situations and reverse them.

The direction of causation goes from the body towards the head. To invert this direction we must somehow switch head and body. Consequently, the edges of the corresponding choice node have to be inverted. It is, however, not possible to just invert all the edges. Only inverting the edges would turn the disjunction in the head of the rule into a conjunction, which would result in a low likelihood since a disjunction was learned from the training data and not a conjunction. Therefore, if a rule has multiple head atoms, it is necessary to split the rule into a different rule for every atom in the rule head. This way the head atoms stay in a disjunction and are not converted into a conjunction. Fig. 7.2.c illustrates this operation. Similar reasoning applies to rules with multiple conditions in the body.

### 7.3.4 Complexity

In this section, we compare the number of refinements generated by SEM and SEM-CP-logic, and discuss the computational cost of both approaches. The input to both methods is the training data and the set of atoms.

SEM generates, during each iteration, precisely $|A|^2$ refinements, with $A$ the set of atom nodes: for each pair of atoms it either generates a refinement by adding a new edge between the nodes representing the atoms or by removing the edge if it already existed in the network. Note that SEM does not automatically introduce new unobserved nodes.

SEM-CP-logic generates $|E| + 4 \cdot |A| \cdot |C| + 3 \cdot |A|^2 + |C|$ refinements, with $E$ the set of edges, $A$ the set of atom nodes, and $C$ the set of choice nodes. This is strictly more than for the SEM algorithm. Moreover, the number of generated refinements may grow as more edges and choice nodes are added to the network. Ignoring the 'invert rule' operation, the number of edges added by a given refinement step is at most 4 and the number of choice nodes added is at most 2, that is, in refinement step $i$, $|E| = O(i)$, and $|C| = O(i)$. So, the total number of refinements generated during $m$ iterations is of the order $\sum_{i=1}^{m} O(i \cdot |A| + |A|^2) = O(m^2|A| + m|A|^2)$. For SEM this is $m|A|^2$. Therefore, the total number of refinements that SEM-CP-logic generates grows quadratically in the number of iterations, while it only grows linearly in the case of SEM (unless one specifies an upper-bound on the number of rules).

Note that comparing the number of refinement steps as a function of the number of iterations does not provide all information required to infer the induction time of both approaches. There are two reasons for this.

1. Evaluating a random refinement has a different computational cost in both methods. SEM-CP-logic runs the costly EM algorithm to learn the

parameters (Section 7.2.1), while this is not necessary in the plain BN case (the parameters can be learned by simple counting if there are no missing values and all domain atoms are observed). On the other hand, the number of parameters in the BNs generated by SEM grows exponentially with the number of edges, while the number of learned parameters grows only linearly with the number of edges in the CP-logic case.

2. The number of iterations required to learn a particular theory may be different in both cases. For example, to represent a CP-theory rule with many head atoms (Section 4.2.3), the BN constructed by SEM will be fully connected, requiring many iterations, while the corresponding CP-theory can be modeled with a fairly simple network with only one choice node and a limited number of edges.

It follows from the above observations that, depending on the problem at hand, either SEM or SEM-CP-logic may be the most efficient method. However, given that the number of parameters of the BN that SEM needs to learn may grow exponentially with the size of the CP-theory, it is not difficult to find cases where SEM-CP-logic will be much faster than SEM. (This requires that SEM-CP-logic does not explicitly store the CPTs of the EBN, which may also grow exponentially with the size of the CP-theory. This is explained in more detail in Chapter 6)

## 7.4 Experiments

The goal of our experiments is to test if SEM-CP-logic is able to learn CP-theories from training data. We present two sets of experiments: (a) experiments in a controlled artificial domain, where the target theory is a CP-theory (Section 7.4.1), and (b) experiments in a real world application where the goal is to discover causal relations between the occurrence of mutations in HIV (Section 7.4.2).

### 7.4.1 Experiments in an Artificial Domain

In this experiment we test how much training data is required to accurately learn a CP-theory with SEM-CP-logic. We also compare the resulting CP-theory to a regular BN learned with SEM from the same training data. We hypothesize that, in this case, SEM will need more training data than SEM-CP-logic to learn a given theory because, as we saw in Section 4.2, it needs to learn exponentially more parameters and a more complex network structure to be able to represent the same theory.

## Experimental Setup

We use the following software implementations. SEM-CP-logic is implemented on top of the Open Bayes toolbox (Gaitanis 2007) for the Python programming language. We use the SEM algorithm from the Structure Learning Package (Leray and Francois 2004), which is an extension of the Bayesian Network Toolbox (Murphy 2007) for Matlab.

We run two sets of experiments, one set with a simple and a second set with a more complex CP-theory as target theory. Both theories are based on the shopping example (Section 4.2.4). The first theory *Shop1* is precisely the theory of Fig. 3.5. It has 5 atoms, 4 rules, 6 parameters, and each rule has at most 2 head atoms. The second theory *Shop2* is a more complex version of the shopping example with 8 atoms, 8 rules, 13 parameters, and at most 3 head atoms in a given rule.

We construct the training and test data by sampling interpretations from the target theory (*Shop1* or *Shop2*). The test set is fixed and consists of 1000 examples for *Shop1* and 2000 examples for *Shop2*. In each trial, we sample a training set of the given size. The learning algorithm trains on this sample, and the resulting models (CP-theory or BN) are tested on the fixed test set. As evaluation measure we use the log likelihood, which is defined as follows:

$$L(D|BN) = \sum_{i=1}^{N} \log Pr(d_i|BN)\,, \tag{7.4}$$

with $D$ the set of $N$ data points $d_i \in D$, and $BN$ the structure and parameters of the BN or EBN.

We report, for each method and training set size, the test set log likelihood averaged over 10 trials, together with the 90% confidence interval for the average, and the minimum and maximum likelihood obtained over the different trails.

When a model assigns probability zero to a particular test example, the log likelihood for that example is $\log(0) = -\infty$, and as a consequence, also the total test set log likelihood will be $-\infty$. This makes it difficult to compare different models. Therefore, we add a penalty of $-700$ to the log likelihood for each example with probability 0 (instead of $-\infty$). The value $-700$ is also used by the SEM software (Murphy 2007) that we use to learn BNs.

### Shopping Example: Structure Learning

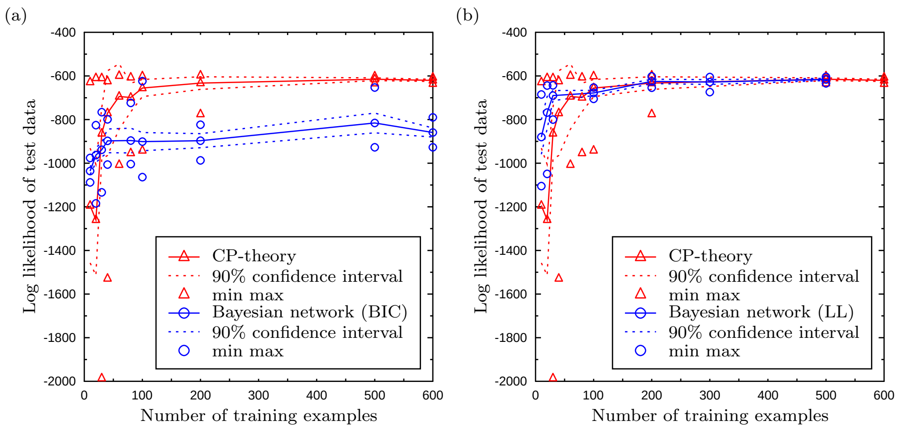


Figure 7.3: Comparing structure learning of CP-theories and BNs for target theory *Shop1*. (a) SEM optimizes BIC score. (b) SEM optimizes log likelihood.

Fig. 7.3.a presents the results for target *Shop1*. Let us first focus on the CP-theory results. For a small number of training examples, there is a large variation in the test set log likelihood, but given more training data, the test set log likelihood converges to the test set log likelihood of the target theory (-603). This shows that SEM-CP-logic is able to accurately learn the target theory given a reasonable amount of training data.

The large variation in test set log likelihood for small training sets can be explained as follows. Causality imposes a strong connection between the atoms' truth values in the data set. When an incorrect causal relationship is inferred from the training data, this may result in a theory that assigns zero likelihood to some of the test examples, yielding a very low test set log likelihood (because of the penalty mentioned above). Suppose that in a given training set atom $y$ is only true when $x$ is true. This may cause SEM-CP-logic to learn the rule

$y : \alpha \leftarrow x$. If the target theory contains alternative causes for $y$, which are not sufficiently represented in the training data, then the test set may contain cases where $y$ is true and $x$ is false. Such a case is not covered by the learned theory, which will assign a likelihood of zero to it. BNs suffer to a lesser extent from this problem because the SEM algorithm starts from a non-zero prior over the CPT elements. This guarantees that the learned conditional probabilities are always non-zero.

For training sets with more than 100 examples, the average test set log likelihood of the BNs is worse than that of the CP-theories. This is because BNs have difficulties to represent the causal structure present in the target theory. As shown in Section 4.2.4, representing the *Shop1* target requires an almost fully connected BN. Given that the BIC score used by SEM prefers networks with a small number of edges, SEM is unable to learn the required structure in most cases. We tried increasing the training set size to 1500 examples, but this did not improve test set log likelihood.

Fig. 7.3.b shows the same results, but now SEM's evaluation score is set to the training set log likelihood instead of the BIC score. So, it no longer includes a penalty for the complexity of the BN. With this modification, the BNs are able to approximate the target theory. However, the learned BNs are almost always fully connected (for the $> 100$ example training sets, the BN is fully connected in all of the trails). Hence, the BNs are much less interpretable than the CP-theories, and do not indicate the causal structure of the problem.

### Shopping Example: Parameter Learning

*Shop1*'s target theory is relatively simple, and without BIC's network complexity penalty, SEM had no difficulty to accurately learn the probability distribution from the given training data. Assuming a fully connected network, it had to learn $2^5 - 1 = 31$ parameters compared to 6 for SEM-CP-logic. *Shop2*'s target theory is more complex. Now there are $2^8 - 1 = 255$ parameters in a fully connected BN, which is much more than the 13 parameters in the target CP-theory. We therefore hypothesize that more training data are needed for SEM than for SEM-CP-logic to accurately learn *Shop2* when there are more parameters in the theory.

Fig. 7.4 presents parameter learning results for *Shop2*. We show results for the CP-theory, and for two different BN structures: one is a fully connected BN, and

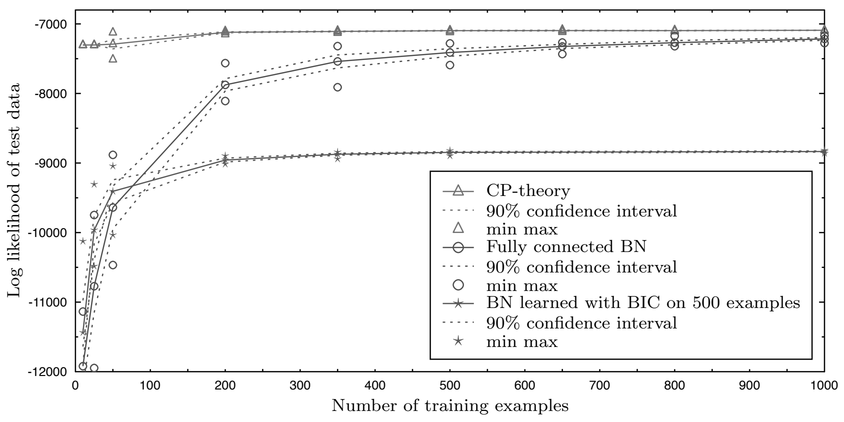


Figure 7.4: Comparing parameter learning for CP-theories and BNs for target theory *Shop2*.

the other is a BN structure that was learned by SEM using the BIC score on a training set consisting of 500 examples.

Since the learned BN has fewer parameters than the fully connected BN, EM can learn its parameters more accurately from small training sets ($< 50$ examples). However, given more training data, the log likelihood of the learned BN converges to a sub-optimal value. Because of its overly simple structure, the learned BN cannot represent the probability distribution of the target theory.

Both the CP-theory and the fully connected BN can represent the target theory. This can be seen from their log likelihood values, which converge to the same value. However, because the CP-theory has much fewer parameters than the fully connected BN, CP-theory parameter learning requires much less training data. (It requires about 200 examples to converge; the log likelihood of the fully connected BN is still below that of the CP-theory after 1000 training examples.)

In this case structure learning will also be unable to learn the target theory exactly. Thus making it even more difficult to learn the structure of *Shop2* than that of *Shop1*.

### 7.4.2 Discovering Causal Relations Between HIV Mutations

To test SEM-CP-logic in a real world application we apply it in the context of medical research targeting the Human Immunodeficiency Virus (HIV). In particular, we are interested in using SEM-CP-logic to discover causal relations between the occurrence of mutations in the virus. Before doing so, we briefly introduce the application domain and the data set that we use.

Multiple drugs have been developed that inhibit the replication of HIV, but in spite of that, current therapies are of limited success. A major reason why the drugs fail is the virus' ability to escape from drug pressure by developing drug resistance. This escape mechanism is based on HIV's high rates of replication and mutation, that is, the presence of the drugs oppresses certain mutations but the ratios will turn around since those mutations in the genetic material of the virus that are not oppressed take over the population. These mutations may in turn replicate into other mutations that improve the virus' fitness. Note that CP-logic is particularly suited to this application domain because of its implicit causality: the fact that one mutation is present, may cause a particular other mutation to become the majority population. It is interesting to model these causal relations between mutations to gain more insight in HIV, and ultimately to find better drug combinations for HIV therapy.

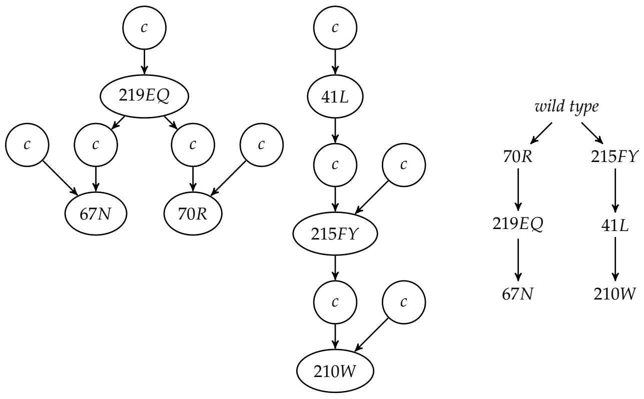
We use the data set from Beerenwinkel et al. (Beerenwinkel et al. 2005). This data set records mutations in HIV's reverse transcriptase gene in patients that are treated with the drug zidovudine. It contains 364 samples. No resistance associated mutations other than six classical zidovudine mutations are present in the data set. Each of these mutations is denoted with a CP-theory atom: 41L, 67N, 70R, 210W, 215F/Y, and 219E/Q. These atoms indicate the location where the mutation occurred (e.g., 41) and the amino acid to which the position mutated (e.g., 'L' for Leucine).

The CP-theory resulting from running SEM-CP-logic on this data set is shown in Fig. 7.5.a and Fig. 7.5.b shows the corresponding EBN. Medical literature states that 41L, 215F/Y and 210W tend to occur together, and that 70R and 219E/Q tend to occur together as well. This can also be seen from the CP-theory.

Beerenwinkel et al. (Beerenwinkel et al. 2005) have applied their 'Mtreemix' software to this data set. Mtreemix builds a probabilistic model that consists of a mixture of so-called mutagenic trees (Fig. 7.5.c). Each node of such a tree is a mutation and the edges represent hypothesized causal relations between

$$\begin{aligned}
219EQ &: 0.19 \leftarrow .\\
67N &: 0.10 \leftarrow .\\
70R &: 0.31 \leftarrow .\\
41L &: 0.27 \leftarrow .\\
215FY &: 0.20 \leftarrow .\\
210W &: 0.01 \leftarrow .\\
67N &: 0.67 \leftarrow 219EQ.\\
70R &: 0.30 \leftarrow 219EQ.\\
215FY &: 0.90 \leftarrow 41L.\\
210W &: 0.01 \leftarrow 215FY.
\end{aligned}$$

a. CP-theory



b. EBN

c. Mutagenetic tree

Figure 7.5: CP-theory, corresponding EBN, and mutagenic tree describing the HIV mutation pathway in patients treated with the drug zidovudine.

the mutations ratios. We have compared the likelihood of Mtreemix's output with two trees in the mixture (one degenerate tree modeling external causes for the mutations, and the tree from Fig. 7.5.c) to that of our CP-theory. Mtreemix obtained a log likelihood of -994.5, while the CP-theory has a log likelihood of -986.5. This shows that SEM-CP-logic is able to model the probability distribution of this causal process at least as accurately as the state-of-the-art.

Mixtures of mutagenic trees are more difficult to interpret than CP-theories. We therefore compare our CP-theory to the single mutagenic tree from Fig. 7.5.c. The CP-event 67N ← 219E/Q is also present in the mutagenic tree. The tree also contains the relation that 41L causes 210W, but then as a direct relation between the two mutations. The relations 70R ← 219E/Q and 215F/Y ← 41L are also present in the tree, but with the direction of causation reversed.

Given the structure of the causal process in this application, it is impossible to deduce the correct causal direction of the relations from a cross-sectional data set (i.e., the data is from different patients and measured at unknown time points) (Pearl 2000). As a result, one has to rely on heuristics. Unfortunately, also domain experts do not know the correct direction of causation in this application (only correlations are known), so we do not have a ground truth to compare the heuristics to.

SEM-CP-logic and Mtreemix use different heuristics and may therefore end up with a different direction of causation. SEM-CP-logic assumes that the most compact theory is the correct causal theory (the one with the highest BIC score), while Mtreemix uses the relative number of occurrences of the mutations to decide on the direction of causation (Beerenwinkel et al. 2005). Because SEM-CP-logic's compactness heuristic is the BIC score, it cannot distinguish between theories with the same size and likelihood. This occurs, for example, for the theories $(x : \alpha_1) \leftarrow y$ and $(y : \alpha_2) \leftarrow x$ if both theories, in addition to the rule, also include external causes for $x$ and $y$. This also applies to the direction of causation in the rules in Fig. 7.5: a theory with the direction of causation reversed in one of the rules would have the same size and likelihood (but different parameter values). To distinguish between such theories, an additional heuristic is required. Such a heuristic could, for example, extend BIC's bias for few external causes, in such a way that it prefers theories with extreme probabilities for the external causes (either close to 1.0 or close to 0.0).

## 7.5 Conclusions

In this chapter, we compared learning Causal Probabilistic Logic (CP-logic) theories to learning Bayesian networks (BNs). Our first contribution is such a comparison at a theoretical level. In particular, we have shown that CP-theories in which the rules have precisely one head atom can be represented by a BN with one edge for each CP-theory rule with a non-empty body. The conditional probability distributions of the nodes in this BN correspond to a combination of noisy-OR and deterministic AND. CP-theories with rules with multiple head atoms, on the other hand, cannot be compactly represented by a BN over the domain atoms. Here, a compact representation is only possible after the introduction of unobserved nodes in the BN.

By relying on a transformation from CP-theories to BNs, it becomes possible to adapt known BN learning techniques to learn CP-theories. Adapting these techniques is the second contribution of this chapter. In particular, based on Friedman's structural EM (SEM) algorithm, we proposed SEM-CP-logic, an algorithm for learning CP-theories from training data. The main advantage of this transformation based approach is that it becomes possible to rely on the large amount of the research available on learning BNs (Heckerman and Brees 1994; Neapolitan 2003) and to reuse this in the context of CP-logic.

We have compared SEM-CP-logic to the SEM algorithm for learning the structure of BNs. Experiments in an artificial domain show that CP-theories better approximate the target theory than BNs if it contains causal relations. SEM needs to learn more edges and more parameters in the CPTs of the BNs than SEM-CP-logic to be able to approximate the target theory; this is detrimental to the efficiency of the learning and to the interpretability of the resulting model. Moreover, CP-theories explicitate the causal relations, which BNs do not.

To test the applicability of SEM-CP-logic, we have used it to discover causal relations between mutations in the HIV. SEM-CP-logic is able to model the probability distribution of this causal process as accurately as a state-of-the-art method (Mtreemix). Because both methods use heuristics to infer the direction of causation, this direction is sometimes different in the theories found by SEM-CP-logic and Mtreemix. Further research is required to assess which heuristic works best.

# 8 Conclusion

In this chapter, the main contributions and conclusions are summarized. Some directions for future work are also provided.

## 8.1 Summary of contributions and conclusions

The goal of this dissertation was to *improve the inference algorithms for directed probabilistic logic models and improve probabilistic logic learning*. The contributions can be summarized as follows:

- Any work on (directed) probabilistic logic models requires a particular formalism to represent the model. In this dissertation we used *CP-logic* and related this formalism to a variety of other known languages in the field of probabilistic logical learning and subfields. Our work on CP-logic can be seen as an extension to the original CP-logic research located in the field of knowledge representation (Vennekens 2007). In this dissertation we extend the knowledge about CP-logic (and directed PLMs in general) to the field of machine learning by giving insights in inference, parameter learning, and structure learning.

CP-logic's main ambition is to be intuitive, modular and compact from a knowledge representation point of view. The syntax and semantics are as simple as possible to make it easy to learn the language. On the other hand, this makes that the CP-logic syntax is not as extensive as some of the other formalisms that focus on versatility and have numerous syntax constructs to express special structures compactly.

- We have presented *first-order Bayes ball*, an algorithm that finds the minimum requisite part of a CP-theory necessary to calculate the probability of a query given some evidence. Because it reasons on the first-order level, it finds the ground CP-theory needed for inference faster than current methods. But more importantly, the resulting requisite CP-theory can be first order, permitting it to be used as input to lifted inference methods. Such methods capitalize the symmetries present in probabilistic logic models to improve the efficiency of inference. The disadvantage of lifted inference methods is that the operations they perform are more expensive since they also need to detect the type of symmetry. By limiting the CP-theory to the requisite part we can avoid doing unnecessary operations and speed up lifted inference. Also since it is not possible to compile first order theories for lifted inference like the junction tree for probabilistic models, it makes sense to perform a query based pruning first with first-order Bayes ball. This is also shown in the experiments where inference was performed on some theories already known in the lifted inference literature and where inference was performed by inference after grounding as well as by lifted inference.

  An additional advantage of first-order Bayes ball is the connection to first-order graphical presentations like parametrized (equivalent) Bayesian networks. This makes it easier to inspect the model and reason about effects propagate through the model. The fact that the structure of such a graph is meaningful without looking at the parameters is thanks to the semantics of CP-logic. If one would like to extend CP-logic with meta predicates such as *forall*, first-order Bayes ball will need to be extended.

- Inference for CP-logic and PLMs is a mixture between probabilistic and logic inference. This requires for specialized algorithms to cope with different types of structures. The *contextual variable elimination with overlapping contexts* method extends probabilistic inference such that it better handles (partly) deterministic relations between variables. Such relations appear for example in contextual and causal independence structures, two types that appear frequently in probabilistic logic models. Contextual variable elimination with overlapping contexts allows for a more compact formalization of the models which results in a reduction of the computational cost. Experiments show that our approach outperforms the original contextual variable elimination and variable elimination on multiple problems.

An important reason to use confactors for CP-logic inference is that they allow us to directly encode the structures present in the CP-theory into confactors. Also more advanced encodings like the multiplicative factorization for causal independence structures are now supported resulting in an optimal encoding. An important aspect of inference in probabilistic models is the order in which variables are handled. For contextual variable elimination and variable elimination in general this is the elimination ordering. Advanced heuristics exists for variable elimination but for contextual variable elimination it is not yet clear how the contexts can be used optimally to decide on the ordering. For contextual variable elimination with overlapping contexts there is an extra issue in that even more general confactors are allowed. For example, when multiplicative ordering is used this has particular influence on the elimination ordering.

- The last contribution is in learning (ground) CP-theories. We proposed an adaptation of the *Structural EM algorithm* for Bayesian networks called SEM-CP-logic. This new approach makes use of the two-way transformation from a CP-theory to an equivalent Bayesian network. The refinement operators are modified such that the method searches heuristically over possible CP-theories instead of Bayesian networks. Experiments in a controlled artificial domain show that for learning CP-theories, learning with SEM-CP-logic requires fewer training data than Bayesian network learning

  Situations where there are causal processes at work can be represented more compactly with a CP-theory than with a Bayesian network. This language bias is used to learn a model of HIV mutations where the interactions can be considered to be causal. Experiments show that this heuristic delivers results that are comparable to current methods used in HIV research.

  Structure learning is a challenging task for probabilistic models as well as logic models. Combing both fields results in a more expressive language which leads to a larger search space for structural learning. Structure learning for probabilistic logic models is therefore still a difficult task. As an example, the refinement operators used in SEM-CP-logic are more specific than those used for Bayesian networks. Since this method learns ground theories, an extra operator is needed to introduce logic variables into the CP-events in order to fully exploit the expressiveness of probabilistic logic models.

## 8.2 Future work

Combining probability theory and logic has proven to be a useful but highly non-trivial. Although there were some earlier ideas, the field of probabilistic logic learning (or statistical relational learning) formed itself in the nineties and is still growing. By now, we are getting past the initial discovery phase of the field and hitting upon the more fundamental issues. More importantly, the research is more focussed on particular techniques and not anymore on the formalism itself. By looking at the underlying principles it is possible to discover methods that apply to multiple formalisms and also to better understand the relationship betweens formalisms (Jaeger, Kersting, and De Raedt 2006). In this dissertation, we investigated some of these issues but, of course, many are still open.

### 8.2.1 Probabilistic loops

In Chapter 3, we briefly touched upon probabilistic loops in a directed probabilistic logic model. Logic programming and CP-logic allow for such loops and have semantics for them. Performing inference, however, is in general not efficient. Transforming the CP-theory to a Bayesian network does not solve the problem since directed loops are not allowed. We proposed a transformation to a loop free CP-theory, but this transformation is naive and the resulting theory is often more complex than needed causing less efficient inference. More research is necessary to find efficient inference algorithms that know how to handle directed cycles.

Probabilistic loops are specific to the field of directed probabilistic logic models as it does not appear as such in probability theory or logic. They are closely related to recursive loops in logic programming where loops are handled through caching. This solution, however, is not applicable because of the interactions with probabilities.

### 8.2.2 Lifted inference

The concept of lifted inference is to perform inference in such a way that possible optimizations from both probability theory and logic theory are taken into account. Results up to now have shown that it is not always possible to simply take the available optimizations from one field and apply them to the field of probabilistic logic learning. These new types of interactions demand new types of optimizations.

The initial approaches to lifted inference (Poole 2003; Salvo Braz, Amir, and Roth 2005; Milch et al. 2008) showed that inference efficiency can be improved by orders of magnitude, but they also hint towards a complicated set of operations necessary to perform lifted inference. Since then, numerous people have contributed insights into aspects of the problem but in general lifted inference methods can only deal with specific models and is not yet usable for real world problems.

With FOBB we presented a valuable companion to existing lifted inference methods to handle more comprehensive and real-life models. A more thorough investigation however is necessary to see how the shattering step in FOBB compares to the shattering in lifted inference algorithms like C-FOVE. Another interesting future step would be caching the outcome of FOBB. Can the requisite part of a theory for a particular query and certain observations be used for another query with other observations? Some initial attempts in this direction have been made for MLNs but there are still some open questions (Aniruddh and Domingos 2010).

### 8.2.3 Learning

We proposed an algorithm to learn ground CP-theories from data. The next step from this point is to extend the operators such that they also introduce first-order predicates. An ideal starting point would be to combine with the relational extension of the FOIL algorithm, *ProbFoil* (De Raedt and Thon 2010).

A completely different direction would be to focus on the causal aspects of CP-logic also for learning. The heuristic method used is based on the natural bias of CP-logic towards logic processes. A more thorough method could be to use the causal theory of (Pearl 2000) to devise a constraints based method for CP-logic based on the IC*-algorithm. From a knowledge representation view this is already being investigated by Vennekens, Bruynooghe, and Denecker (2010).

### 8.2.4 Continuous variables

Although CP-logic can handle multi-valued variables in the first-order setting, it is not clear how to cope with continuous variables. This is in general true for SRL and in lesser degree for Bayesian networks. Therefore, an interesting further path is to investigate how to deal with continuous variables in SRL which is already gaining attention (Gutmann, Jaeger, and De Raedt 2010). This is not only interesting from a research point of view but also for practical applications where continuous variables are a common feature.

### 8.2.5 Inhibitory events

CP-logic natively supports a noisy-OR like behaviour, where different causes enforce the common consequence. Often, however, there is a need for something like *inhibitory events* similar to inhibitory connections in neural networks. Basically they decrease the activity of the target, which for CP-logic means that if the body of such an event is true, it lessens the probability that the consequence is true. Although we can model such an inhibitory effect in CP-logic, the abundance of such events in, for example, diagnostics applications, make that it would be advantageous to support this type of event by means of a syntax construct.

The independence of events offers great flexibility but makes it sometimes slightly more difficult to express particular situations where an inhibitory event would fit. For example, if there is already a general rule modeled that declares that cars can break and we want to add a rule for a specific brand of cars, it overrules the general rule to make it less likely. Simply adding a specific rule will enlarge the probability of breaking because the probabilities of the general rule and the specific one are combined.

$$\begin{aligned}&\mathit{broken}(\mathit{Car}) : 0.3.\\&\mathit{broken}(\mathit{Car}) : 0.2 \leftarrow \mathit{isParticularBrand}(\mathit{Car}).\end{aligned}$$

In case the specific rules needs to express a higher probability of breaking, the probability of the specific rule can be adjusted such that the combination represents the real probability. But if it is lower, the modularity is lost since it is necessary to overrule the general rule and adjust it such that the specific case is excluded. For example,

$$\begin{aligned}&\mathit{broken}(\mathit{Car}) : 0.3 \leftarrow \neg \mathit{isParticularBrand}(\mathit{Car}).\\&\mathit{broken}(\mathit{Car}) : 0.2 \leftarrow \mathit{isParticularBrand}(\mathit{Car}).\end{aligned}$$

The question is whether the CP-logic syntax or semantics can be extended in an intuitive way such that inhibitory events can be handled as modularly as regular events. The modularity of the CP-events can be linked to the decomposability of noisy-or and is categorized as an *positive qualitative influence* by Lucas (2005). Inhibitory events as discussed here can be seen as *negative qualitative influences* since there is always a general rule that also applies. In that case, a modular approach seems to be possible (for example by negating the causal probability). More research, however, is necessary to see if an inhibitory event is indeed monotonic in behaviour and thus not an *ambiguous qualitative influence* in which could be a hint that there is no guaranteed efficient inference method.

# Bibliography

AFNOR (2008). *NF E01-010: Mécatronique - Vocabulaire* (cited on page 5).

Aji, S. M. and R. J. Mceliece (2000). “The generalized distributive law”. In: *IEEE Transactions on Information Theory* 46(2), pages 325–343 (cited on page 23).

Aniruddh, N. and P. Domingos (2010). “Efficient lifting for online probabilistic inference”. In: *Proceedings ofthe 24th AAAI Conference on Artificial Intelligence*. (Atlanta, Georgia, USA, July 11–15, 2010), pages 1193–1198 (cited on page 167).

Bancilhon, F., D. Maier, Y. Sagiv, and J. D. Ullman (1986). “Magic sets and other strange ways to implement logic programs (extended abstract)”. In: *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*. (Cambridge, Massachusetts, Mar. 24–26, 1986), pages 1–15 (cited on page 5).

Baum, L. E., T. Petrie, G. Soules, and N. Weiss (1970). “A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains”. In: *Annals of Mathematical Statistics* (41), pages 164–171 (cited on page 141).

Beerenwinkel, N., J. Rahnenfuhrer, M. Daumer, D. Hoffmann, R. Kaiser, J. Selbig, and T. Lengauer (2005). “Learning multiple evolutionary pathways from cross-sectional data." In: *Journal of Computational Biology* 12(6), pages 584–598 (cited on pages 159, 161).

Bertelè, U. and F. Brioschi (1972). *Nonserial Dynamic Programming*. New York: Academic Press, Inc (cited on pages 23, 134).

Blockeel, H. and W. Meert (2006). "Towards learning non-recursive LPADs by transforming them into Bayesian networks". In: *Proceedings ofthe 15th International Conference on Inductive Logic Programming (ILP)*. (Bonn, Germany, Aug. 10–13, 2005). Volume 3625. Lecture Notes in Computer Science, pages 94–108 (cited on pages 36, 140, 187).

Boutilier, C., N. Friedman, M. Goldszmidt, and D. Koller (1996). "Context-specific independence in Bayesian networks". In: *Proceedings ofthe 12th Conference on Uncertainty in Artificial Intelligence (UAI)*. (Portland, Oregon, USA, Aug. 1–4, 1996), pages 115–123 (cited on page 121).

Breese, J. (1992). "Construction of belief and decision networks". In: *Computational Intelligence* 8(4), pages 624–647 (cited on pages 76, 78, 92).

Breese, J., R. Goldman, and M. Wellman (1994). "Introduction to the special section on knowledge-based construction of probabilistic and decision models". In: *IEEE Transactions on Systems, Man, and Cybernetics* 24(11), pages 1577–1579 (cited on pages 65, 84).

Broek, B. Van de (2010). "Branched gold nanoparticles for specific photothermal therapy of cancer". PhD thesis. K.U.Leuven (cited on page ii).

Bruynooghe, M., 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: *Proceedings of the International Workshop on Statistical Relational Learning (SRL)*. (Leuven, Belgium, July 2–4, 2009). (Cited on pages 66, 186).

Chavira, M. and A. Darwiche (2007). "Compiling Bayesian networks using variable elimination". In: *Proceedings ofthe 20th International Joint Conference on Artificial Intelligence (IJCAI)*. (Hyderabad, India, Jan. 6–12, 2007), pages 2443–2449 (cited on pages 118, 134).

Colmerauer, A. and P. Roussel (1993). "The birth of prolog". In: *SIGPLAN Notices* 28 (3), pages 37–52. (Cited on page 31).

Cozman, F. G. (2004). "Axiomatizing noisy-or". In: *Proceedings ofthe 16th European Conference on Artificial Intelligence (ECAI)*. (Valencia, Spain, Aug. 22–27, 2004), pages 979–980 (cited on page 118).

Darwiche, A. (2003). “A differential approach to inference in bayesian networks”. In: *Journal of the ACM* 50 (3), pages 280–305 (cited on page 23).

De Raedt, L. and K. Kersting (2003). “Probabilistic logic learning”. In: *ACM SIGKDD Explorations Newsletter* 5(1), pages 31–48 (cited on pages 4, 35).

De Raedt, L., A. Kimmig, and H. Toivonen (2007). “ProbLog: a probabilistic Prolog and its application in link discovery”. In: *Proceedings ofthe 20th International Joint Conference on Artificial Intelligence (IJCAI)*. (Hyderabad, India, Jan. 6–12, 2007), pages 2468–2473 (cited on pages 63, 65, 82, 93).

De Raedt, L. and I. Thon (2010). “Probabilistic rule learning”. In: *Proceedings ofthe 20th International Conference on Inductive Logic Programming (ILP)*. (Florence, Italy, June 27–30, 2010) (cited on page 167).

De Raedt, L., B. Demoen, D. Fierens, B. Gutmann, G. Janssens, A. Kimmig, N. Landwehr, T. Mantadelis, W. Meert, R. Rocha, V. Santos Costa, I. Thon, and J. Vennekens (2008). “Towards digesting the alphabet-soup of statistical relational learning”. NIPS 2008 Workshop on Probabilistic Programming. (Whistler, Canada, Dec. 13, 2008). (Cited on pages 63, 83, 187).

Dechter, R. (1999). “Bucket elimination: a unifying framework for reasoning”. In: *Journal of Artificial Intelligence* 113(1), pages 41–85. (Cited on page 23).

Dempster, N., A. Laird, and D. Rubin (1977). “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of Royal Statistical Society B* 39, pages 185–197 (cited on pages 41, 141).

Dewdney, A. K. (1993). *The New Turing Omnibus: Sixty-Six Excursions in Computer Science*. New York, USA: Computer Science Press/Freeman (cited on page 26).

Díez, F. J. and S. F. Galán (2003). “Efficient computation for the noisy max”. In: *International Journal of Intelligent Systems* 18(2), pages 165–177 (cited on pages 111, 118, 129, 130, 135).

Driessens, K. and W. Meert (2010). “Automatic smart diagnostics”. FMTC Symposium. (Gent, Belgium, Dec. 1, 2010) (cited on page 187).

Fierens, D., H. Blockeel, J. Ramon, and M. Bruynooghe (2004). “Logical Bayesian networks”. In: *Proceedings ofthe 3rd International Workshop on Multi-Relational Data Mining (MRDM)*. (Seattle, USA, Aug. 22, 2004), pages 19–30 (cited on pages 65, 85).

Fierens, D., H. Blockeel, M. Bruynooghe, and J. Ramon (2006). “Logical bayesian networks and their relation to other probabilistic logical models”. In: *Proceedings ofthe 15th International Conference on Inductive Logic Programming*

*(ILP)*. (Bonn, Germany, Aug. 10–13, 2005). Volume 3625. Lecture Notes in Computer Science, pages 40–42 (cited on page 85).

Friedman, N. (1997). "Learning belief networks in the presence of missing values and hidden variables". In: *Proceedings ofthe 14th International Conference on Machine Learning (ICML)*. (Nashville, Tennessee, USA, July 8–12, 1997), pages 125–133 (cited on pages 139, 145, 146).

Friedman, N., M. Goldszmidt, and A. Wyner (1999). "Data analysis with bayesian networks: a bootstrap approach". In: *Proceedings ofthe 15th Conference on Uncertainty in Artificial Intelligence (UAI)*. (Stockholm, Sweden, July 30–Aug. 1, 1999), pages 206–215 (cited on page 65).

Frühwirth, T. (2009). *Constraint Handling Rules*. Cambridge University Press (cited on page 79).

Gaitanis, K. (2007). *Open Bayes for Python*. URL: `http://www.openbayes.org/` (cited on pages 142, 144, 145, 155).

Geiger, D., T. S. Verma, and J. Pearl (1990). *Identifying independence in Bayesian networks*. Technical report CSD-890028 (R-116). UCLA Cognitive Systems Laboratory (cited on page 25).

Gelder, A. V., K. A. Ross, and J. S. Schlipf (1991). "The well-founded semantics for general logic programs". In: *ACM* 38(3), pages 620–650 (cited on page 45).

Getoor, L. and B. Taskar, editors (2007). *Statistical Relational Learning*. MIT Press (cited on pages 4, 5, 35).

Gutmann, B., M. Jaeger, and L. De Raedt (2010). "Extending ProbLog with continuous distributions". In: *Proceedings ofthe 20th International Conference on Inductive Logic Programming (ILP)*. (Florence, Italy, June 27–30, 2010). (Cited on page 168).

Haddawy, P. (1994). "Generating bayesian networks from probability logic knowledge bases". In: *Proceedings ofthe 10th Conference on Uncertainty in Artificial Intelligence (UAI)*. (Seattle, USA, July 29–31, 1994), pages 262–269 (cited on page 4).

Halpern, J. Y. (1990). "An analysis of first-order logics of probability". In: *Journal of Artificial Intelligence* 46(3), pages 311 –350. (Cited on page 87).

Heckerman, D. and J. S. Brees (1994). "Causal independence for probability assessment and inference using Bayesian networks". In: *IEEE Transactions on Systems, Man & Cybernetics, Part A (Systems & Humans)* 26(6), pages 826–831 (cited on pages 67, 71, 75, 161).

Hilbert, D. and W. Ackermann (1950). *Principles of mathematical logic*. New York: Chelsea Publishing Company (cited on page 3).

Hofstadter, D. R. (1979). *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books (cited on page 26).

Huth, M. and M. Ryan (2004). *Logic in Computer Science: Modelling and Reasoning about Systems, Second Edition*. Cambridge University Press (cited on page 26).

Jaeger, M. (1997). "Relational Bayesian networks". In: *Proceedings ofthe 13th Conference on Uncertainty in Artificial Intelligence (UAI)*. (Providence, Rhode Island, USA, Aug. 1–3, 1997) (cited on page 86).

Jaeger, M. (2007). "Parameter learning for relational bayesian networks". In: *Proceedings ofthe 24th International Conference on Machine Learning (ICML)*. (Corvallis, USA, June 20–24, 2007), pages 369–376 (cited on pages 84, 142).

Jaeger, M., K. Kersting, and L. De Raedt (2006). "Expressivity analysis for pl-languages". In: *Working Notes of the ICML Workshop "Open Problems in Statistical Relational Learning" (SRL)*. (Pittsburgh, USA) (cited on page 166).

Kersting, K. and L. De Raedt (2000). "Bayesian logic programs". In: *Proceedings ofthe Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming (ILP)*, pages 138–155 (cited on page 92).

Kersting, K. and L. De Raedt (2007). "An Introduction to Statistical Relational Learning". In: MIT Press. Chapter Bayesian Logic Programming: Theory and Tool, pages 291–322. (Cited on pages 4, 93, 96, 107, 127).

Kersting, K. and L. De Raedt (2008). "Basic principles of learning bayesian logic programs". In: *Probabilistic Inductive Logic Programming*. Volume 4911. Lecture Notes in Computer Science, pages 189–221 (cited on pages 5, 65, 84, 139).

Kimmig, A. (2010). "A Probabilistic Prolog and its Applications". PhD thesis. Department of Computer Science, Faculty of Engineering, K.U.Leuven. (Cited on page 120).

Kimmig, A., V. Santos Costa, R. Rocha, B. Demoen, and L. De Raedt (2008). "On the efficient execution of ProbLog programs". In: *Proceedings ofthe 24th International Conference on Logic Programming (ICLP)*. (Udine, Italy, Dec. 9–13, 2008). Volume 5366. Lecture Notes in Computer Science, pages 175–189 (cited on pages 63, 120).

Kindermann, R. and J. L. Snell (1980). *Markov random fields and their applications*. Volume 1. Contemporary Mathematics. Providence, R.I.: American Mathematical Society, pages ix+142. (Cited on page 2).

Kisynski, J. (2010). "Aggregation and Constraint Processing in Lifted Probabilistic Inference". PhD thesis. Vancouver, Canada: University of British Columbia (cited on pages 91, 97).

Kisynski, J. and D. Poole (2009a). "Constraint processing in lifted probabilistic inference". In: *Proceedings ofthe 25th Conference on Uncertainty in Artificial Intelligence (UAI)*. (Montreal, Canada, July 18–21, 2009) (cited on page 93).

Kisynski, J. and D. Poole (2009b). "Lifted aggregation in directed first-order probabilistic models". In: *Proceedings ofthe 21th International Joint Conference on Artificial Intelligence (IJCAI)*. (Pasadena, California, USA, July 11–17, 2009), pages 1922–1929 (cited on pages 112, 114).

Kok, S. and P. Domingos (2009). "Learning Markov Logic Network structure via hypergraph lifting". In: *Proceedings ofthe 26th International Conference on Machine Learning (ICML)*. (Montreal, Canada, July 14–18, 2009) (cited on page 5).

Kolmogorov, A. (1933). *Grundbegriffe der wahrscheinlichkeitsrechnung*. Berlin: Julius springer (cited on pages 2, 11).

Leray, P. and O. Francois (2004). *BNT Structure Learning Package: Documentation and Experiments*. Technical report. Laboratoire PSI (cited on page 155).

Lloyd, J. (1987). *Foundations of Logic Programming*. 2nd. SV (cited on pages 3, 26).

Lobo, J., J. Minker, and A. Rajasekar (1992). *Foundations of disjunctive logic programming*. MIT Press (cited on page 57).

Lucas, P. J. (2001). "Certainty-factor-like structures in Bayesian belief networks". In: *Knowledge-based systems* 14(7), pages 327–335 (cited on page 70).

Lucas, P. J. (2005). "Bayesian network modelling through qualitative patterns". In: *Journal of Artificial Intelligence* 163(2), pages 233–263 (cited on pages 67, 129, 169).

Mantadelis, T. and G. Janssens (2009). "Tabling relevant parts of SLD proofs for ground goals in a probabilistic setting". In: *International Colloquium on Implementation of Constraint and LOgic Programming Systems (CICLOPS)*. (Pasadena, California, July 14–17, 2009) (cited on page 128).

Mateescu, R. and R. Dechter (2008). *Mixed deterministic and probabilistic networks*. ICS Technical Report. UCI (cited on page 118).

Meert, W. (2007a). "Learning CP-logic theories". Computational Intelligence and Learning Doctoral School contact day (Leuven, Belgium, Aug. 29, 2007) (cited on page 188).

Meert, W. (2007b). "Towards learning non-recursive LPADs by transforming them into Bayesian networks". Former Freiburg, Leuven and Friends Workshop on Machine Learning (Heer, Massembre, Belgium, Mar. 21–23, 2007). (Cited on page 188).

Meert, W. (2008a). “Learning causal relationships”. Spring Workshop on Mining and Learning (Traben-Trarbach, Germany, Apr. 21–25, 2008) (cited on page 188).

Meert, W. (2008b). “PLL: Learning for CP-logic”. Seminar on Logic and Computation (Oostduinkerke, Belgium, Apr. 29–30, 2008). (Cited on page 188).

Meert, W. (2008c). “PLL: Learning for CP-logic”. Computational Intelligence and Learning Doctoral School contact day (at ECML) (Antwerp, Belgium, Sept. 15, 2008) (cited on page 188).

Meert, W., J. Struyf, and H. Blockeel (2007). “Learning ground CP-logic theories by means of Bayesian network techniques”. In: *Proceedings ofthe 6th workshop on Multi-Relational Data Mining (MRDM)*. (Warsaw, Poland, Sept. 17, 2007), pages 93–104 (cited on pages 36, 140, 187).

Meert, W., J. Struyf, and H. Blockeel (2008a). “CP-logic theory inference with contextual variable elimination”. NIPS 2008 Workshop on Probabilistic Programming. (Whistler, Canada, Dec. 13, 2008). (Cited on page 187).

Meert, W., J. Struyf, and H. Blockeel (2008b). “Learning ground CP-logic theories by leveraging Bayesian network learning techniques.” In: *Fundamenta Informaticae* 89(1), pages 131–160 (cited on pages 66, 107, 118, 120, 141, 185).

Meert, W., J. Struyf, and H. Blockeel (2009a). “CP-logic theory inference with contextual variable elimination and comparison to BDD based inference methods”. In: *Proceedings ofthe 19th International Conference on Inductive Logic Programming (ILP)*. (Leuven, Belgium, July 1–4, 2009). Volume 5989. Lecture Notes in Computer Science, pages 96–109. (Cited on pages 119, 187).

Meert, W., J. Struyf, and H. Blockeel (2009b). “Experimental comparison of CP-logic theory inference methods”. In: *Proceedings ofthe 18th Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*. (Tilburg, The Netherlands, May 18–19, 2009), pages 101–102 (cited on page 186).

Meert, W., J. Struyf, and H. Blockeel (2010a). “Contextual variable elimination with overlapping contexts”. In: *Proceedings ofthe 5th European workshop on Probabilistic Graphical Models (PGM10)*. (Helsinki, Finland, Sept. 13–15, 2010), pages 193–210 (cited on pages 119, 186).

Meert, W., J. Struyf, and H. Blockeel (2010b). “Contextual variable elimination with overlapping contexts”. In: *Proceedings ofthe 19th Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*. (Leuven, Belgium, May 27–28, 2010) (cited on page 186).

Meert, W., N. Taghipour, and H. Blockeel (2010). "First-order Bayes-ball". In: *Proceedings ofthe 21th European Conference on Machine Learning (ECML)*. (Barcelona, Spain, Sept. 20–23, 2010). Volume 6322. Lecture Notes in Computer Science, pages 369–384. (Cited on pages 93, 186).

Milch, B., L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling (2008). "Lifted probabilistic inference with counting formulas". In: *Proceedings ofthe 21st AAAI Conference on Artificial Intelligence*. (Boston, Massachusetts, USA, July 16–20, 2006), pages 1062–1608 (cited on pages 91, 93, 108, 112, 167).

Mitchell, T. (1997). *Machine learning*. McGrawHill (cited on page 3).

Muggleton, S. (2000). "Learning stochastic logic programs". In: *Proceedings ofthe AAAI 2000 Workshop on Learning Statistical Models from Relational Data*. (Austin, Texas, USA, July 31, 2000) (cited on pages 65, 87).

Muggleton, S. and W. Buntine (1988). "Machine invention of first-order predicates by inverting resolution." In: *Proceedings ofthe 5th International Conference on Machine Learning (ICML)*. (Ann Arbor, Michigan, USA, July 12–14, 1988), pages 339–352 (cited on pages 79, 85).

Murphy, K. P. (2007). *Bayesian Network Toolbox*. URL: `http://bnt.sourceforge.net` (cited on pages 155, 156).

Natarajan, S., P. Tadepalli, E. Altendorf, T. Dietterich, A. Fern, and A. Restificar (2006). "Learning first-order probabilistic models with combining rules". In: *Proceedings ofthe 23nd International Conference on Machine Learning (ICML)*. (Pittsburgh, USA, June 25–29, 2006), pages 609–616 (cited on pages 84, 142).

Neapolitan, E. R. (2003). *Learning Bayesian Networks*. Prentice Hall (cited on pages 11, 15, 68, 143–145, 147, 161).

Nikiforakis, N., W. Meert, Y. Younan, M. Johns, and W. Joosen (2011). "SessionShield: lightweight protection against session hijacking". In: *Proceedings ofthe International Symposium on Engineering Secure Software and Systems (ESSoS10)*. (Madrid, Spain, Feb. 9–10, 2011) (cited on page 186).

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann (cited on pages 2, 11, 23, 25, 67, 117).

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press (cited on pages 147, 160, 168).

Poole, D. and N. L. Zhang (2003). "Exploiting contextual independence in probabilistic inference". In: *Journal of Artificial Intelligence Research* 18, pages 263–313 (cited on pages 118, 119, 121, 123, 125, 130, 131, 134, 135).

Poole, D. (1993). "Probabilistic horn abduction and bayesian networks". In: *Journal of Artificial Intelligence* 64(1), pages 81–129 (cited on page 78).

Poole, D. (1997). “The independent choice logic for modelling multiple agents under uncertainty”. In: *Journal of Artificial Intelligence* 94(1-2), pages 7–56 (cited on pages 65, 78, 93).

Poole, D. (2003). “First-order probabilistic inference.” In: *Proceedings ofthe 18th International Joint Conference on Artificial Intelligence (IJCAI)*. (Acapulco, Mexico, Aug. 9–15, 2003), pages 985–991 (cited on pages 91, 96, 97, 100, 101, 167).

Ramon, J., T. Croonenborghs, D. Fierens, H. Blockeel, and M. Bruynooghe (2008). “Generalized ordering-search for learning directed probabilistic logical models”. In: *Machine Learning* 70(2-3), pages 169–188 (cited on page 117).

Richardson, M. and P. Domingos (2006). “Markov logic networks”. In: *Machine Learning* 62(1), pages 107–136 (cited on pages 65, 87, 139).

Riguzzi, F. (2004). “Learning logic programs with annotated disjunctions”. In: *Proceedings ofthe 14th International Conference on Inductive Logic Programming (ILP)*. (Porto, Portugal, Sept. 6–8, 2004). Volume 4455. Lecture Notes in Computer Science, pages 270–287 (cited on pages 47, 48, 140).

Riguzzi, F. (2007). “A top down interpreter for LPAD and CP-logic”. In: *Proceedings ofthe 10th Congress of the Italian Association for Artificial Intelligence (AI*IA)*. (Rome, Italy, Sept. 10–13, 2007). Volume 4733. Lecture Notes in Computer Science, pages 109–120 (cited on pages 63, 120).

Riguzzi, F. (2010). “SLGAD resolution for inference on Logic Programs with Annotated Disjunctions”. In: *Fundamenta Informaticae* 102(3–4), pages 429–466 (cited on page 128).

Robinson, J. A. (1965). “A machine-oriented logic based on the resolution principle”. In: *Journal of ACM* 12(1), pages 23–41. (Cited on page 29).

Salvo Braz, R. de, E. Amir, and D. Roth (2005). “Lifted first-order probabilistic inference”. In: *Proceedings ofthe 19th International Joint Conference on Artificial Intelligence (IJCAI)*. (Edinburgh, Scotland, UK, July 30–Aug. 5, 2005), page 1319 (cited on pages 91, 167).

Sato, T. (1995). “A statistical learning method for logic programs with distribution semantics”. In: *Proceedings ofthe 12th International Conference on Logic Programming (ICLP)*. (Tokyo, Japan, June 13–16, 1995), pages 715–729 (cited on page 83).

Sato, T. and Y. Kameya (2008). “New advances in logic-based probabilistic modeling by prism”. In: *Proceedings ofthe 18th International Conference on Inductive Logic Programming (ILP)*. (Prague, Czech Republic, Sept. 10–12, 2008). Volume 5194. Lecture Notes in Computer Science (cited on pages 65, 79).

Savicky, P. and J. Vomlel (2007). “Exploiting tensor rank-one decomposition in probabilistic inference”. In: *Kybernetika* 43(5), pages 747–764 (cited on page 135).

Shachter, R. D. (1999). “Bayes-ball: the rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams)”. In: *Proceedings ofthe 15th Conference on Uncertainty in Artificial Intelligence (UAI)*. (Stockholm, Sweden, July 30–Aug. 1, 1999) (cited on pages 91, 92, 94, 100, 108).

Shafer, G. (1996). *The Art of Causal Conjecture*. Cambridge, MA, USA: MIT Press (cited on page 42).

Shafer, G. and V. Vladimir (2006). “The sources of Kolmogorov’s *Grundbegriffe*”. In: *Statistical Science* 21, pages 70–98 (cited on page 2).

Singla, P. and P. Domingos (2008). “Lifted first-order belief propagation”. In: *Proceedings ofthe 21st AAAI Conference on Artificial Intelligence*. (Boston, Massachusetts, USA, July 16–20, 2006) (cited on pages 91, 113).

Sneyers, J., W. Meert, and J. Vennekens (2009). “CHRiSM: CHance Rules induce Statistical Models”. In: *Proceedings ofthe 6th International Workshop on Constraint Handling Rules (CHR)*. (Pasadena, California, USA, July 15, 2009), pages 62–76. (Cited on pages 66, 80, 186).

Sneyers, J., P. Van Weert, T. Schrijvers, and L. De Koninck (2010a). “As time goes by: constraint handling rules – a survey of CHR research between 1998 and 2007”. In: *Theory and Practice of Logic Programming* 10(1), pages 1–47 (cited on page 79).

Sneyers, J., W. Meert, J. Vennekens, Y. Kameya, and T. Sato (2010b). “CHR(PRISM)-based probabilistic logic learning”. In: *Theory and Practice of Logic Programming* 10(Special Issue 4-6), pages 433–447. (Cited on pages 66, 80, 185).

Stoev, J., A. Bartic, S. Gillijns, and W. Symens (2010). “Badminton playing robot - a multidisciplinary test case in mechatronics”. In: *Proceedings ofthe 5th IFAC Symposium on Mechatronic Systems*. (Cambridge, Massachusetts, USA, Sept. 13–15, 2010) (cited on page 5).

Taghipour, N., W. Meert, and H. Blockeel (2010). “First-order Bayes-ball”. Former Freiburg, Leuven and Friends Workshop on Machine Learning (Boppard, Germany, Mar. 15–17, 2010) (cited on page 188).

Taghipour, N., W. Meert, J. Struyf, and H. Blockeel (2009). "First-order Bayes-ball for CP-logic". In: *Proceedings ofthe International Workshop on Statistical Relational Learning (SRL)*. (Leuven, Belgium, July 2–4, 2009), pages 1–3. (Cited on pages 93, 187).

Thon, I., N. Landwehr, and L. De Raedt (2008). "Cpt-l: an efficient model for relational stochastic processes". In: *In Proceedings of the 6th International Workshop on Mining and Learning with Graphs (MLG)*. (Helsinki, Finland, July 4–5, 2008) (cited on page 87).

Van Emden, M. and R. Kowalski (1976). "The semantics of predicate logic as a programming language". In: *Journal of the ACM* 23(4), pages 733–742 (cited on page 31).

Vennekens, J. (2007). "Algebraic and Logical Study of Constructive Processes in Knowledge Representation". PhD thesis. Department of Computer Science, K.U.Leuven (cited on pages 35, 38, 43, 60, 78, 163).

Vennekens, J., M. Bruynooghe, and M. Denecker (2010). "Embracing events in causal modelling: interventions and counterfactuals in CP-logic". In: *Proceedings ofthe 12th European Conference on Logics in Artificial Intelligence (JELIA)*. (Helsinki, Finland, Sept. 13–15, 2010). Volume 6341. Lecture Notes in Computer Science, pages 313–325 (cited on pages 38, 168).

Vennekens, J., M. Denecker, and M. Bruynooghe (2009a). "CP-logic: a language of causal probabilistic events and its relation to logic programming". In: *Theory and Practice of Logic Programming* 9(3), pages 245–308 (cited on pages 35–37, 42, 87).

Vennekens, J., M. Denecker, and M. Bruynooghe (2009b). "CP-logic: a language of causal probabilistic events and its relation to logic programming". In: *CoRR* abs/0904.1672 (cited on page 67).

Vennekens, J., S. Verbaeten, and M. Bruynooghe (2004). *Logic Programs with Annotated Disjunctions*. Technical report. Department of Computer Science, K.U.Leuven (cited on pages 35, 43, 45, 46, 140).

Vomlel, J. (2006). "Noisy-or classifier". In: *International Journal of Intelligent Systems* 21(3), pages 381–398 (cited on pages 67, 71, 142, 144).

Wellman, M. P., J. Breese, and R. Goldman (1992). "From knowledge bases to decision models". In: *Knowledge Engineering Review* 7, pages 35–53 (cited on page 4).

Zhang, N. L. and D. Poole (1994). "A simple approach to Bayesian network computations". In: *Proceedings ofthe 10th Canadian Conference on Artificial*

*Intelligence*. (Banff, Alberta, Canada, May 1994), pages 171–178 (cited on page 23).

Zhang, N. L. and D. Poole (1996). "Exploiting causal independence in Bayesian network inference". In: *Journal of Artificial Intelligence Research* 5, pages 301–328 (cited on pages 117, 123).

# Curriculum Vitae

Wannes Meert was born on March 4th 1982, in Leuven, Belgium. After finishing high school at the Pius X College in Tessenderlo, he started studying engineering at the Katholieke Universiteit Leuven in 2000. In 2005, he received the degree of Master in Electrical Engineering (option Micro-electronics), followed one year later by the degree of Master in Artificial Intelligence (option Computer Science and Engineering). In September 2006, he joined the DTAI (Declaratieve Talen en Artificiële Intelligentie) group of the Department of Computer Science to pursue a Ph.D. on probabilistic logic learning under the supervision of prof. Hendrik Blockeel. Since January 2007, his research was funded by a personal grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen).

# List of publications

Available on http://www.cs.kuleuven.be/publicaties/lirias/mypubs.php?unum=U0052683.

## Journal Articles

- W. Meert, J. Struyf, and H. Blockeel (2008b). "Learning ground CP-logic theories by leveraging Bayesian network learning techniques." In: *Fundamenta Informaticae* 89(1), pages 131–160
- J. Sneyers, W. Meert, J. Vennekens, Y. Kameya, and T. Sato (2010b). "CHR(PRISM)-based probabilistic logic learning". In: *Theory and Practice of Logic Programming* 10(Special Issue 4-6), pages 433–447.

## Peer-reviewed Conference and Workshop Articles

- N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen (2011). "SessionShield: lightweight protection against session hijacking". In: *Proceedings ofthe International Symposium on Engineering Secure Software and Systems (ESSoS10)*. (Madrid, Spain, Feb. 9–10, 2011)
- W. Meert, N. Taghipour, and H. Blockeel (2010). "First-order Bayes-ball". In: *Proceedings ofthe 21th European Conference on Machine Learning (ECML)*. (Barcelona, Spain, Sept. 20–23, 2010). Volume 6322. Lecture Notes in Computer Science, pages 369–384.
- W. Meert, J. Struyf, and H. Blockeel (2010a). "Contextual variable elimination with overlapping contexts". In: *Proceedings ofthe 5th European workshop on Probabilistic Graphical Models (PGM10)*. (Helsinki, Finland, Sept. 13–15, 2010), pages 193–210
- W. Meert, J. Struyf, and H. Blockeel (2010b). "Contextual variable elimination with overlapping contexts". In: *Proceedings ofthe 19th Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*. (Leuven, Belgium, May 27–28, 2010)
- W. Meert, J. Struyf, and H. Blockeel (2009b). "Experimental comparison of CP-logic theory inference methods". In: *Proceedings ofthe 18th Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*. (Tilburg, The Netherlands, May 18–19, 2009), pages 101–102
- J. Sneyers, W. Meert, and J. Vennekens (2009). "CHRiSM: CHance Rules induce Statistical Models". In: *Proceedings ofthe 6th International Workshop on Constraint Handling Rules (CHR)*. (Pasadena, California, USA, July 15, 2009), pages 62–76.
- 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: *Proceedings of the International Workshop on Statistical Relational Learning (SRL)*. (Leuven, Belgium, July 2–4, 2009).

- N. Taghipour, W. Meert, J. Struyf, and H. Blockeel (2009). "First-order Bayes-ball for CP-logic". In: *Proceedings ofthe International Workshop on Statistical Relational Learning (SRL)*. (Leuven, Belgium, July 2–4, 2009), pages 1–3.
- W. Meert, J. Struyf, and H. Blockeel (2009a). "CP-logic theory inference with contextual variable elimination and comparison to BDD based inference methods". In: *Proceedings ofthe 19th International Conference on Inductive Logic Programming (ILP)*. (Leuven, Belgium, July 1–4, 2009). Volume 5989. Lecture Notes in Computer Science, pages 96–109.
- W. Meert, J. Struyf, and H. Blockeel (2007). "Learning ground CP-logic theories by means of Bayesian network techniques". In: *Proceedings ofthe 6th workshop on Multi-Relational Data Mining (MRDM)*. (Warsaw, Poland, Sept. 17, 2007), pages 93–104
- H. Blockeel and W. Meert (2006). "Towards learning non-recursive LPADs by transforming them into Bayesian networks". In: *Proceedings ofthe 15th International Conference on Inductive Logic Programming (ILP)*. (Bonn, Germany, Aug. 10–13, 2005). Volume 3625. Lecture Notes in Computer Science, pages 94–108

## Peer-reviewed Abstracts and Posters

- W. Meert, J. Struyf, and H. Blockeel (2008a). "CP-logic theory inference with contextual variable elimination". NIPS 2008 Workshop on Probabilistic Programming. (Whistler, Canada, Dec. 13, 2008).
- L. De Raedt, B. Demoen, D. Fierens, B. Gutmann, G. Janssens, A. Kimmig, N. Landwehr, T. Mantadelis, W. Meert, R. Rocha, V. Santos Costa, I. Thon, and J. Vennekens (2008). "Towards digesting the alphabet-soup of statistical relational learning". NIPS 2008 Workshop on Probabilistic Programming. (Whistler, Canada, Dec. 13, 2008).

## Posters and Presentations at miscellaneous events

- K. Driessens and W. Meert (2010). "Automatic smart diagnostics". FMTC Symposium. (Gent, Belgium, Dec. 1, 2010)

- N. Taghipour, W. Meert, and H. Blockeel (2010). “First-order Bayes-ball”. Former Freiburg, Leuven and Friends Workshop on Machine Learning (Boppard, Germany, Mar. 15–17, 2010)
- W. Meert (2008b). “PLL: Learning for CP-logic”. Seminar on Logic and Computation (Oostduinkerke, Belgium, Apr. 29–30, 2008).
- W. Meert (2008a). “Learning causal relationships”. Spring Workshop on Mining and Learning (Traben-Trarbach, Germany, Apr. 21–25, 2008)
- W. Meert (2008c). “PLL: Learning for CP-logic”. Computational Intelligence and Learning Doctoral School contact day (at ECML) (Antwerp, Belgium, Sept. 15, 2008)
- W. Meert (2007a). “Learning CP-logic theories”. Computational Intelligence and Learning Doctoral School contact day (Leuven, Belgium, Aug. 29, 2007)
- W. Meert (2007b). “Towards learning non-recursive LPADs by transforming them into Bayesian networks”. Former Freiburg, Leuven and Friends Workshop on Machine Learning (Heer, Massembre, Belgium, Mar. 21–23, 2007).

**Arenberg Doctoral School of Science, Engineering & Technology**
Faculty of Engineering
Department of Computer Science
Declarative Languages and Artificial Intelligence
Kasteelpark Arenberg 1
B-3001 Heverlee