Tractable Probabilistic Models for Causal Learning and Reasoning

Benjie Wang

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Summer 2023

Acknowledgements

Pursuing a DPhil, especially partially during a global pandemic, has been a challenging, but immensely rewarding experience. Here I would like to name just a few of the people who have made this journey special.

First and foremost, I would like to express my deepest gratitude to my wonderful supervisor, Marta Kwiatkowska. Since starting as a DPhil student in 2019, I have learned so much about research under your patient and insightful guidance. Thank you in particular for your encouragement and advice in pursuing important research questions without fear or expediency. This thesis, and my development as a independent researcher, would not have been possible otherwise.

Thanks are also due to my friends and colleagues who I have had the good fortune to work with over the past years: Emanuele La Malfa, Matthew Wicker, Clare Lyle, Xiyue Zhang, Hjalmar Wijk, Tom Rainforth, Daqian Shao, Siddhartha Datta, Artem Velikzhanin, Jon Vadillo, Luca Laurenti, Stefan Webb, Anthony Hartshorn, and Rui Yan. Emanuele, it has been a pleasure sharing the DPhil journey with you. Matthew, Clare, Xiyue, Hjalmar, Tom: I could not have asked for more enthusiastic and talented collaborators - I have learned much working with each of you. Thank you all for enriching my DPhil life, from the long Zoom discussions during the pandemic to the whiteboard brainstorms and coffee breaks after.

Finally, I am, and have always been, indebted to my Mum and Dad for their unconditional love and support. This thesis, and all of my achievements, I owe to you.

I gratefully acknowledge funding from the European Research Council (ERC) through the FUN2MODEL project (grant agreement No. 834115), under the European Union's Horizon 2020 research and innovation programme. The research outcomes in this thesis would not have been possible without this generous support.

Abstract

This thesis examines the application of tractable probabilistic modelling principles to causal learning and reasoning. Tractable probabilistic modelling is a promising paradigm that has emerged in recent years, which focuses on probabilistic models that enable *exact* and *efficient* probabilistic reasoning. In particular, the framework of probabilistic circuits provides a systematic language of the tractability of models for various inference queries based on their structural properties, with recent proposals pushing the boundaries of expressiveness and tractability. However, not all information about a system can be captured through a probability distribution over observed variables; for example, the causal direction between two variables can be indistinguishable from data alone. Formalizing this, Pearl's Causal Hierarchy (also known as the *information hierarchy*) delineates three levels of causal queries, namely, associational, interventional, and counterfactual, that require increasingly greater knowledge of the underlying causal system, represented by a structural causal model and associated causal diagram. Motivated by this, we investigate the possibility of tractable causal modelling; that is, exact and efficient reasoning with respect to classes of causal queries. In particular, we identify three scenarios, separated by the amount of knowledge available to the modeler: namely, when the full causal diagram/model is available, when only the observational distribution and identifiable causal estimand are available, and when there is additionally uncertainty over the causal diagram. In each of the scenarios, we propose probabilistic circuit representations, structural properties, and algorithms that enable efficient and exact causal reasoning. These models are distinguished from tractable probabilistic models in that they can not only answer different probabilistic inference queries, but also causal but also different interventions and even different causal diagrams. However, we also identify key limitations that cast

doubt on the existence of a fully general tractable causal model. Our contributions also extend the theory of probabilistic circuits by proposing new properties and circuit architectures, which enable the analysis of advanced inference queries including, but not limited to, causal inference estimands.

Contents

List of Symbols

$\boldsymbol{U}, \boldsymbol{u}, U, u$	Exogenous variables
$\boldsymbol{V}, \boldsymbol{v}, V, v$	Endogenous variables
$\boldsymbol{W}, \boldsymbol{w}$	Subset of variables
$\boldsymbol{Q}, \boldsymbol{q}$	Marginal determinism
$\boldsymbol{X}, \boldsymbol{x}, X, x$	Treatment variables (intervention targets)
$\boldsymbol{Y}, \boldsymbol{y}, Y, y$	Outcome variables
$\boldsymbol{Z}, \boldsymbol{z}, Z, z$	Adjustment variables
$\boldsymbol{E}, \boldsymbol{e}$	Event/Evidence variables
$\mathcal{D}, d, n$	Dataset, Variable dimension, Number of samples
G	Graph
$\mathrm{pa}, \mathrm{ch}, \mathrm{an}, \mathrm{desc}$	Parents, children, ancestors, descendants
σ	Order
p	Probability distribution
$\mathbb{E}$	Expectation
$\mathcal{P}, \boldsymbol{\mathcal{P}}$	Probabilistic model, Probabilistic model class
$\mathcal{N}, \mathcal{CBN}, Pr$	Bayesian network (BN), causal Bayesian network (CBN), conditional probability distributions (CPD)
$\boldsymbol{\lambda}, \lambda, \boldsymbol{\Theta}, \theta$	Indicator, parameter variables for BN
$\mathcal{M}, \boldsymbol{F}, \mathrm{Pr}$	Structural causal model (SCM), Functional mechanisms, Exogenous variables distribution
$\mathcal{C}, \boldsymbol{\mathcal{C}}$	Probabilistic circuit (PC), Probabilistic circuit class
$\mathcal{C}_{sub}$	Complete subcircuit
$\boldsymbol{\omega}, \boldsymbol{g}$	PC weights, leaf node functions
N, T, P, L, R	PC nodes (generic, sum, product, leaf, root)
ϕ, supp	Scope, support (for PC nodes)
v, w	Vtree, Md-vtree
M, E	Vtree nodes, edges
ψ	Decision variable/Vtree labelling function
f	BN/PC polynomial
$\mathbb{R}$	Set of real numbers
$\mathbb{1}$	Indicator/identity function

List of Figures

Chapter 1

Introduction

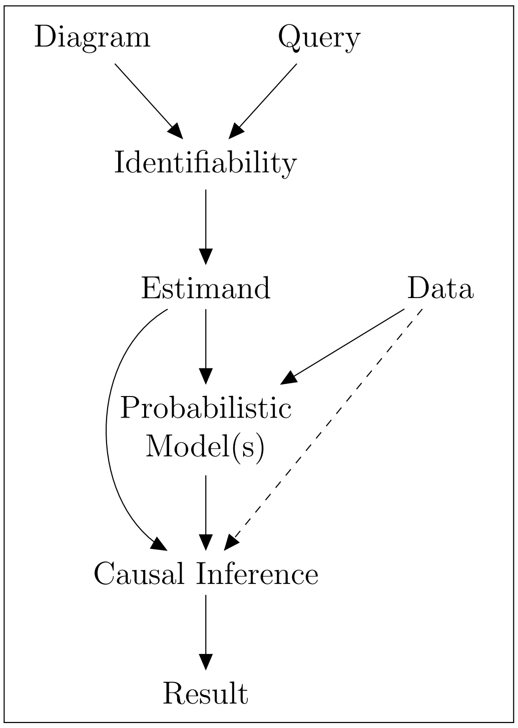
Machine learning, and particularly methods based on deep neural networks, have revolutionized the field of artificial intelligence, achieving impressive performance on many long-standing challenges, from image classification to playing Go, that would have been unimaginable just two decades ago. Learning algorithms implement the principle that computers can improve their performance on a task over time simply by observing examples, without being explicitly programmed [119]. This is particularly important in the many domains and tasks where handcrafting a program would be impractical for a human. With modern computational resources and the advent of improved learning algorithms and model architectures, machine learning is now commonly deployed in the real world, in areas from medical imaging to self-driving cars to computer graphics [58, 122].

Perhaps the most remarkable feature of these methods is in their autonomy and generality: many seemingly unconnected tasks have all been solved, essentially end-to-end, by applying powerful function approximation techniques from data. Furthermore, once a model is trained from data, performing inference (for example, predicting labels in a discriminative model, or sampling from a generative model) is often orders of magnitude faster. These attributes help to explain the popularity of machine learning methods. However, with greater popularity has come greater scrutiny, and the practical deficiencies of purely data-driven machine learning are by now well known. In particular, models based on deep learning exhibit issues such as poor *robustness* to input perturbations; lack of *explainability* to humans; and poor performance under *distribution shift*.

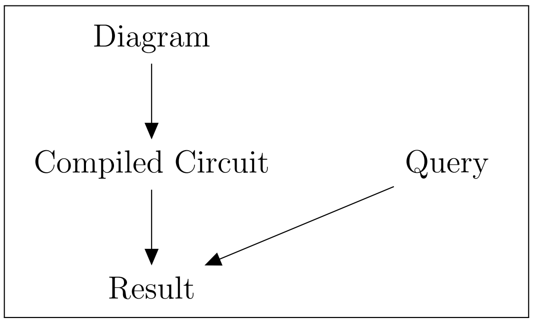
One criticism that could be levelled at end-to-end machine learning models is that, in contrast to traditional approaches to AI, such as expert systems or probabilistic graphical models, they typically lack a consistent knowledge base (model), and associated reasoning algorithms. For example, probabilistic generative models can claim to encode a knowledge base through the joint probability distribution they express, but reasoning about the distribution in modern large-scale models is typically intractable, meaning that we have to resort to approximations, for example by sampling or by using heuristics (e.g. prompting in autoregressive language models). Obtaining guarantees on or reasoning over the output of such models typically requires significant additional computational effort [86, 100, 195]. The downside to traditional knowledge-based systems, however, is that they often involve significant hand-coding of knowledge, and require reasoning algorithms that are generally far slower than inference in machine learning models, limiting their scalability to high-dimensional problems.

To tackle this apparent tradeoff, the emerging field of tractable probabilistic modelling (TPM) is concerned with studying families of probabilistic models, and classes of probabilistic inference queries, for which reasoning about the queries is exact (or approximable with guarantees) and computationally efficient. Tractable probabilistic models have the desirable property that they can be used as a transparent world model; that is, we can efficiently and exactly inspect any aspect of the probability distribution represented by the model that can be captured by a tractable query class. For example, in a latent variable model, MAP (maximum a posteriori) queries allow us to inspect the most likely values of the latent variables given an observed sequence, which can be interpreted as an explanation of the underlying system dynamics. Recent advances in TPMs have shown the ability to scale tractable models far beyond what was previously thought possible [146, 137, 107].

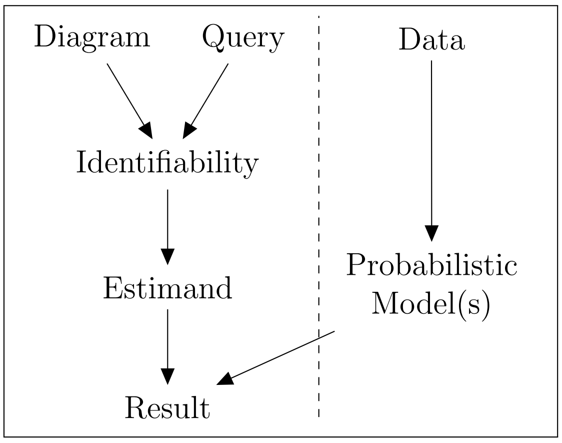
However, even tractable models of the observed probability distribution cannot reliably express causality in the underlying system. Understanding cause-effect relationships is arguably the bedrock of scientific inquiry, and indeed the way in which humans reason about the world. However, it has long been understood that inferring causality from associations in data alone is not possible. The modern treatment of causality, starting from the seminal work of Pearl [132], sheds further light on this by



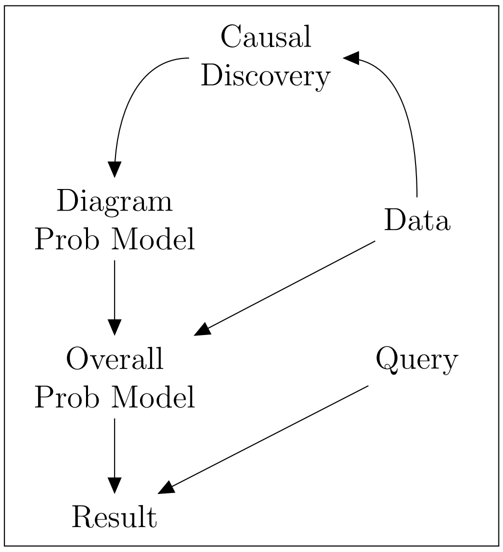
(a) Causal Inference



(b) Compilation



(c) Model + Estimand



(d) Reasoning with diagram uncertainty

Figure 1.1: Conceptual overview of causal reasoning frameworks/workflows

proposing an *information hierarchy*, in which observed data is provably insufficient to make causal statements without assumptions about the underlying domain [9]. Such assumptions can be specified in the form of a causal diagram, which is a directed acyclic graph over the domain variables specifying the causal relationships. Causal inference then provides a principled schema for combining graphical assumptions with data in order to answer causal queries, as depicted in Figure 1.1a. Correspondingly, the space of causal queries is larger and more expressive than probabilistic queries, due to the addition of *interventional semantics* which depend on the causal diagram. For example, this more expressive language has given rise to new causality-aware notions of robustness [176] and fairness [98, 145] of models.

In this thesis, we examine the synthesis of causal reasoning and tractable probabilistic modelling. The application of tractable modelling principles to causality is especially appealing as causal models can be viewed, in some sense, to be closer to the underlying physical reality of the data-generating system [158]; in this view, causal models are viewed as a useful abstraction of the underlying physics of the system, while probabilistic models are an abstraction in turn of the causal model. As such, the development and deployment of a "tractable causal model" would constitute the ability to efficiently inspect the causal knowledge behind the model that is being used, which is highly desirable for developing reliable artificial intelligence. Correspondingly, developing classes of *causal queries* that can be answered efficiently on these models would significantly simplify the application of causal reasoning, as the models could then form an efficient and modular component of a larger intelligent system.

However, there are a couple of key challenges towards developing tractable causal models. The first is conceptual in nature, relating to what information should be contained inside such a model. Unlike probabilistic modelling, where the answers to all possible queries are present in the joint distribution, causal models have differing levels of abstraction that differ in terms of the queries that they can answer, as well as their learnability (identifiability) from data. The second is computational in nature, and relates to identifying model representations that enable tractable computation of causal queries (and their corresponding algorithms). This, too, becomes more challenging due to the larger space of possible causal queries compared to probabilistic queries, induced by the interventional semantics of causal models.

1.1 Contributions

In this thesis, we develop theory and methodology for tractable causal modelling. The contributions of this thesis can be summarized as 1) introducing new settings for tractable causal reasoning, beyond fully specified graphical models; 2) identifying tractability conditions and algorithms for causal reasoning in each of these settings; 3) developing new TPM (specifically, probabilistic circuit [33]) architectures, which satisfy the requirements for tractable causal reasoning by design and can be learned from data; and 4) introducing a new language for analyzing tractability in probabilistic circuits. We explain our contributions in more detail in the following itemized description:

- We introduce the concept of *tractable causal modelling*, which broadly refers to probabilistic models that can be used to (computationally) tractably answer many different causal queries (for example, with respect to different causal interventions, or different causal graphs). In Figure 1.1a, we depict the typical approach to causal reasoning. Here, one first decides on a causal query, and then fits a probabilistic model in order to aid in estimating that query. In Figures 1.1b, 1.1c, and 1.1d, we propose cases where a single probabilistic model can be used to answer many causal queries.

- We analyze tractable causal modelling in the existing setting of compiled representations of causal graphs (Figure 1.1b). In this setting, one compiles a causal graph into a tractable probabilistic circuit, that then enables reasoning over a broad class of causal queries (interventional marginals) in polynomial time. We show conditions on the compiled circuit (specifically, the elimination order) that enable efficient computation of two advanced classes of causal queries, namely, interventional robustness, and counterfactual queries. We further demonstrate the application of interventional robustness to analyzing the robustness of classifiers to causal distribution shifts.

- We introduce two new settings for tractable causal modelling, when knowledge of the fully-specified causal diagram is not available (or compilation is too expensive). The first (Figure 1.1c) concerns performing exact causal inference when the observational distribution is given as a general probabilistic circuit, rather than a circuit compiled from a causal graph. This allows for a conceptual separation between the *causal* quantities, on the left hand side of the Figure, and the *probabilistic* model, given on the right hand side of the Figure. The second (Figure 1.1d) concerns exact inference in tractable causal models expressing the data distribution jointly with uncertainty over the causal graph. In both cases, we develop a novel tractable probabilistic circuit model (MDNet and OrderSPN respectively) that are tractable for the relevant causal queries.

- In the first new setting (Figure 1.1c), we prove that causal inference is intractable for most classes of probabilistic circuits. We then identify subclasses of circuits (and corresponding algorithms) that enable tractable computation of interventional distributions when the distribution is identified through either the backdoor, frontdoor, or napkin causal formulae. These constitute the first non-compiled circuits for which exact causal inference is known to be tractable, and the corresponding algorithms constitute the first poly-time algorithms for causal inference on probabilistic circuits that do not rely on a compilation assumption. On the other hand, we also discuss why it is likely not possible to design circuits which are tractable for causal inference entirely independently of the causal graph, without losing expressivity.

- In the second new setting (Figure 1.1d), we propose a new approximate representation (OrderSPNs) that specifies a joint distribution over causal graphs and variables. We show that, for this model, it is possible to tractably evaluate interventional likelihoods, and in the common case of linear Gaussian causal models, evaluate causal effects averaged over the uncertainty. We further show empirically that the tractability and compactness of OrderSPNs can have practical benefits with regards to accuracy compared to other approximate inference methods.

- We introduce new theory and methodology for tractable probabilistic modelling. In particular, we introduce a new framework, md-vtrees, for analyzing support properties in probabilistic circuits, together with an architecture, MDNets, for enforcing these properties independently of the scope properties. We show separation results that show these new circuits can be exponentially more succinct than previously proposed classes of circuits. We then show how this framework can be used to extend the compositional approach to inference in TPMs, to derive tractability conditions and algorithms for a wider range of inference queries.

1.2 Thesis Outline

We now outline the structure of this thesis. In Chapter 2, we introduce necessary background on causality and tractable probabilistic modelling. In Chapter 3, we then survey the literature on tractable modelling and computational aspects of causality. Following this, in Chapter 4, we present results on tractability of causal reasoning using compiled circuits. In Chapter 5, we present results on the tractability of causal inference for general circuits, as well as our new framework for support properties in probabilistic circuits. In Chapter 6, we then describe causal reasoning under diagram uncertainty and introduce OrderSPNs. Finally, in Chapter 7 we conclude by summarizing the findings of this thesis, and suggesting promising directions for future work. Proofs not included in the main text and other details can be found in the Appendix.

1.3 Publications

This thesis is based upon several papers [191, 196, 193, 190] which were published during my studies. Additionally, during my DPhil I contributed to a number of other works which are not included in this thesis [100, 192, 184, 208]. In this section I detail my contributions to joint-authored work in the context of this thesis.

Chapter 4 is primarily based on two papers [191, 196]. Firstly, in [191] (published at IJCAI 2021), the interventional robustness problem, its complexity in relation to marginal MAP, and the bounding methodology was introduced. I formalized the interventional robustness problem, proposed the use of arithmetic circuits, developed and proved the properties and correctness of the bounding algorithm, and performed the majority of the experimental evaluations. The initial motivation behind the work and some other results not included in the chapter are due to other authors. In [196] (published at IJCAI 2022), the extension to credal sets was proposed, and simplifications of the results in the first paper were made. I conceived the initial idea of extending to parameter bounds, assisted with the proofs, and contributed to the experimental methodology. The idea of the simplifications, and experimental evaluations comparing to credal inference methods were due to other authors. The part of the chapter on computing counterfactuals was done independently and has not appeared in published work to date.

Chapter 5 is based upon the paper [190] (published at AISTATS 2023) and parts of a preceding workshop version [189], together with some new results regarding succinctness of md-vtrees. The idea of analyzing exact computation of causal inference queries, the md-vtree framework for circuits, the MDNet architecture, theoretical results, and the methodology and experiments were my own.

Chapter 6 is based upon the paper [193] (published at ICML 2022). The idea of applying tractable models to Bayesian structure learning, the design of OrderSPNs, reasoning algorithms, and execution of experiments were my own. The learning procedure and design of experiments was developed over time, primarily by myself but in conjunction with other authors, and some illustrations in the experiments are due to other authors.

Chapter 2

Preliminaries

Contents

In this chapter, we cover the essential technical material underpinning the work in this thesis. The chapter is divided into two broad sections: the first concerns foundations in causality, while the second covers probabilistic inference with a particular focus on probabilistic circuit models.

Notation We use uppercase letters to denote a random variable (e.g. V) and lowercase letters for an instantiation of a variable (e.g. v). Sets of variables (and their assignments) are denoted using bold font (e.g. $\boldsymbol{V}, \boldsymbol{v}$), and we use val to denote the set of all instantiations of a set of variables (e.g. $\text{val}(\boldsymbol{V})$). In a slight abuse of notation, we use lowercase to indicate instantiations of arbitrary sets; for example, $\boldsymbol{v} \setminus \boldsymbol{w}$ is an instantiation of $\boldsymbol{V} \setminus \boldsymbol{W}$. We use calligraphic letters (e.g. $\mathcal{P}, \mathcal{C}$) to denote probabilistic/causal models, and $p_{\mathcal{P}}$ to denote the distributions represented by those models. For ease of reference, a full list and explanation of notation used can be found in the preamble to this Thesis.

2.1 Causality

In this section, we cover the basics of modern treatments of causality, focusing on Pearl's graphical framework [133]. In this framework, the underlying reality of a data-generating system is assumed to be given by a *structural causal model* (SCM). We explain the information hierarchy of causal queries, from associational to interventional to counterfactual queries. Finally, we explain the problem of causal inference; that is, computing causal quantities given partial information on the underlying SCM.

2.1.1 Structural Causal Models

The science of causality seeks to draw conclusions about the causal relationships underlying a data-generating system. Unfortunately, it has been observed through many scientific fields, from economics to population genetics to clinical experimental design, that such conclusions are not easy or even possible to draw purely from data, as epitomized in the common refrain *correlation is not causation.* The fundamental issue, observed in [130], is that probability theory is insufficient as a language to express the experimental conditions or context under which data is generated, precluding the direct application of statistical inference to draw causal conclusions. For example, the observation of a correlation between smokers and developing lung cancer does not distinguish between a world in which smoking causes the development of cancer, and one in which there exists a gene that causes people to be predisposed to both smoking and developing lung cancer, a possibility which was exploited by cigarette manufacturers [60, 68] to cast doubt on the interpretation of observational studies.

The first component towards tackling causal questions is to formally define models that can distinguish between these different underlying realities. To this end, structural causal models (SCMs) [132] provide a framework for specifying the underlying causal mechanisms in a data-generating process.

Definition 2.1 (Structural Causal Model)**.** *A SCM $\mathcal{M}$ is a tuple $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, Pr)$ where:*

- $\boldsymbol{U}$ *is a set of exogenous (i.e. outside the model) random variables, which are typically unobserved;*
- $\boldsymbol{V} = \{V_1, .., V_d\}$ *is a set of endogenous (i.e. inside the model) random variables;*
- $\boldsymbol{F} = \{F_1, ..., F_d\}$ *is a set of causal mechanisms (functions). Each endogenous variable* V_i *is associated with a set of endogenous parents* $\boldsymbol{PA}_i \subseteq \boldsymbol{V} \setminus \{V_i\}$ *and a set of exogenous parents* $\boldsymbol{U}_i \subseteq \boldsymbol{U}$*. Then,* F_i *is a mapping from the domain of* $\boldsymbol{U}_i \cup \boldsymbol{PA}_i$ *to the domain of* V_i*, i.e.*

$$V_i \leftarrow F_i(\boldsymbol{U}_i, \boldsymbol{PA}_i) \tag{2.1}$$

- $Pr(\boldsymbol{U})$ *is a probability distribution over the exogenous variables.*

In this definition, the endogenous variables are quantities that we wish to model causally (e.g. smoking, cancer), whose parents can be viewed as direct causes of the variable, as given by the causal mechanisms. The exogenous variables can be interpreted as external sources of noise whose causal relationships are not being explicitly modelled, besides their effect on the endogenous variables (e.g. genetics). A SCM represents a modular data-generating process, where the state of the external world is sampled (exogenous variables), and then the value of each endogenous variable is generated according to its causal mechanism deterministically.

In practice, it is often helpful to use a graphical abstraction of a SCM known as a *causal diagram* (or causal graph). Causal diagrams represent the essential graphical structure of the causal relations between endogenous variables in an SCM.

Definition 2.2 (Causal Diagram)**.** *The causal diagram* $\mathcal{G}$ *induced by a SCM* $\mathcal{M} = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, Pr)$ *is a directed graph consisting of:*

- *A node for every endogenous variable* $V_i \in \boldsymbol{V}$*;*
- *Directed edges* $V_j \rightarrow V_i$ *whenever* $V_j \in \boldsymbol{PA}_i$*, i.e.* V_j *appears as an argument of* F_i*;*
- *Bidirected edges* $V_j \longleftrightarrow V_i$ *whenever* $\boldsymbol{U}_i$ *and* $\boldsymbol{U}_j$ *share a variable, or they are correlated in* $Pr(\boldsymbol{U})$*.*

Though SCMs with cyclic dependencies are also possible [14], in this thesis we focus on acyclic SCMs, also known as semi-Markovian SCMs.

Definition 2.3 (Semi-Markovian SCM)**.** *A SCM $\mathcal{M}$ is said to be semi-Markovian if the causal diagram induced by the SCM does not contain directed cycles. In this case, for each variable $V \in \boldsymbol{V}$ we write $V(\boldsymbol{U})$ to denote the value of V as a function of the exogenous variables $\boldsymbol{U}$.*

In semi-Markovian SCMs, any value $\boldsymbol{u}$ of the exogenous values generates unique values for each of the endogenous variables, by applying the causal mechanisms in any topological order of the causal diagram. As a result, the SCM defines a distribution over the d endogenous variables $\boldsymbol{V}$, given by:

$$p_{\mathcal{M}}(\boldsymbol{V}) = \sum_{\boldsymbol{U}} \Pr(\boldsymbol{U}) \prod_{i=1}^{d} \mathbb{1}_{V_i = F_i(\boldsymbol{U}_i, \boldsymbol{PA}_i)} \tag{2.2}$$

We call this the *observational* distribution, as it reflects the distribution over the endogenous (observed) variables $\boldsymbol{V}$ that the data generated from a system given by the SCM follows.

Semi-Markovian SCMs allow for the presence of *latent confounding*; that is, correlations between endogenous variables which are caused by common or correlated exogenous parent(s). On the other hand, Markovian SCMs make a more stringent assumption that there is no such exogenous confounding, and as such do not have any bidirected edges. As a result, correlations between variables can always be attributed either to a direct causal relation, or an endogenous (observed) common parent.

Definition 2.4 (Markovian SCM)**.** *A semi-Markovian SCM $\mathcal{M}$ is said to be Markovian if the following conditions hold:*

1. *$Pr(\boldsymbol{U}) = \prod_{U \in \boldsymbol{U}} Pr_U(U)$, i.e. the exogenous variables are independent;*
2. *Each exogenous variable is a parent of exactly one endogenous variable.*

2.1.2 The Causal Hierarchy of Queries

The utility of the SCM framework is not purely in its ability to model causal relationships in the underlying system, but also in how it makes predictions about the behaviour of the system. For example, the distribution of the endogenous (observed) variables in a semi-Markovian SCM is given by marginalizing out the exogenous variables as in Equation 2.2. However, SCMs can answer a much wider range of causal queries that go beyond the observational distribution. These queries are organized in a *causal hierarchy*, spanning from *associational* to *interventional* to *counterfactual*. We begin by giving an intuitive account:

1. **Associational**: Associational queries relate to observation; e.g. "is smoking correlated with lung cancer incidence?"

2. **Interventional**: Interventional queries relate to actions, or *doing*; e.g. "if we made everyone stop smoking, would this decrease lung cancer incidence?"

3. **Counterfactual**: Counterfactual queries relate to imagining, or retrospecting; e.g. "if my father had not smoked, would he still be alive today?"

These query classes form an information hierarchy, in that the higher levels of the hierarchy are more general and require more information on the SCM to answer. For example, associational queries are a special case of interventional queries where we "do nothing", while interventional queries are a special case of counterfactual queries where there is no counterfactual condition (in this case, the condition is the fact that the speaker's father did smoke, contracted lung cancer, and died). In order to define these queries more formally, we first need to define the concept of an intervention on a SCM:

Definition 2.5 (SCM Intervention)**.** *Given a SCM $\mathcal{M} = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, Pr)$, where $\boldsymbol{V} = \{V_1, ...V_d\}$, an intervention is a set of causal mechanisms $\boldsymbol{F'} = \{F'_1, ...F'_d\}$.*

We define the set of intervened endogenous variables $\boldsymbol{V_{F'}}$, where $V_{\boldsymbol{F'},i}$ is defined by the structural equation:

$$V_{\boldsymbol{F'},i} = F'_i(\boldsymbol{U}_i, \boldsymbol{PA}_{\boldsymbol{F'},i}) \tag{2.3}$$

In other words, a SCM intervention $\boldsymbol{F}'$ changes the causal mechanisms for (a subset of) the endogenous variables $\boldsymbol{V}$, defining a new set of intervened variables $\boldsymbol{V}_{\boldsymbol{F}'}$. The most common type of intervention is the *hard* intervention, where a subset $\boldsymbol{X} \subseteq \boldsymbol{V}$ of variables are set to a specific value $\boldsymbol{x}$. This can be expressed by setting $F'_i \equiv v_i$ when $V_i \in \boldsymbol{X}$, and $F'_i = F_i$ otherwise. This can be interpreted as performing the corresponding action in the system (for example, eliminating smoking). In such cases, we notate $\boldsymbol{V}_{\boldsymbol{x}}$ to mean $\boldsymbol{V}_{\boldsymbol{F}'}$.

With the definition of interventions in hand, we can now define interventional and counterfactual distributions. While there is only a single observational distribution for an SCM, there is a different interventional distribution for every intervention, and different counterfactual distributions for any combination of interventions.

Definition 2.6 (Causal Distributions)**.** *Given an SCM $\mathcal{M}$, observational ($\mathcal{L}_1$), interventional ($\mathcal{L}_2$), and counterfactual ($\mathcal{L}_3$) distributions are defined by:*

1. ***Associational/Observational**: The observational distribution is given by:*

$$p_{\mathcal{M}}(\boldsymbol{V}) = \sum_{\boldsymbol{U}} Pr(\boldsymbol{U}) \prod_{i=1}^{d} \mathbb{1}_{V_i = F_i(\boldsymbol{U}_i, \boldsymbol{PA}_i)} \tag{2.4}$$

2. ***Interventional**: Given any intervention $\boldsymbol{F}'$, the interventional distribution is given by:*

$$p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}'}) = \sum_{\boldsymbol{U}} Pr(\boldsymbol{U}) \prod_{i=1}^{d} \mathbb{1}_{V_{\boldsymbol{F}',i} = F'_i(\boldsymbol{U}_i, \boldsymbol{PA}_{\boldsymbol{F}',i})} \tag{2.5}$$

3. ***Counterfactual**: Given any set of interventions $\boldsymbol{F}^{(1)}, ..., \boldsymbol{F}^{(n)}$, the counterfactual distribution is given by:*

$$p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{V}_{\boldsymbol{F}^{(n)}}) = \sum_{\boldsymbol{U}} Pr(\boldsymbol{U}) \prod_{j=1}^{n} \left(\prod_{i=1}^{d} \mathbb{1}_{V_{\boldsymbol{F}^{(j)},i} = F_i^{(j)}(\boldsymbol{U}_i, \boldsymbol{PA}_{\boldsymbol{F}^{(j)},i})} \right) \tag{2.6}$$

In practice, rather than the distributions themselves, we are often interested in functions of these distributions corresponding to interpretable quantities. For example, for the observational distribution, we may want to compute the marginal probability of a specific variable $Y \in \boldsymbol{V}$, i.e. $p_{\mathcal{M}}(Y = y) = \sum_{\boldsymbol{V} \setminus \{Y\}} p_{\mathcal{M}}(\boldsymbol{V} \setminus \{Y\}, Y = y)$. We call such functions associational ($\mathcal{L}_1$), interventional ($\mathcal{L}_2$), and counterfactual ($\mathcal{L}_3$) *queries* respectively.

These queries, or predictions of the SCM, are important for two reasons. Firstly, from a *learning* perspective, observing data generated from a system allows us to deduce properties of the underlying SCM. For example, if we observe data from an intervened system where $\boldsymbol{X}$ has been set to $\boldsymbol{x}$, then the SCM should have the property that $p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{x}})$ (approximately) matches the empirical data distribution. Secondly, from a *reasoning* perspective, when we know (part of) the underlying SCM, we would like to use queries to gain insight into the behaviour of the system. For example, for a given intervention target $\boldsymbol{X}$ and two values $\boldsymbol{x}, \boldsymbol{x}'$ of $\boldsymbol{X}$, the average causal effect on Y, given by $\mathbb{E}_{p_{\mathcal{M}}}[Y_{\boldsymbol{x}'}] - \mathbb{E}_{p_{\mathcal{M}}}[Y_{\boldsymbol{x}}]$, expresses the effect of acting on the system. Note that this is an interventional query, as each expectation term is a function of the probability distribution of a single intervened SCM.

Given that all of these query types are defined as (functions of) probabilistic expressions on the underlying SCM, one might wonder, why do we need to explicitly distinguish between them? One answer is the distinction between observation, doing, and retrospecting that we established earlier, which humans understand on an intuitive level and use everyday in commonsense reasoning. However, armed with the formalism of structural causal models and interventions, it can be shown that there is an *information gap* between the layers of the causal hierarchy.

Definition 2.7 (Similarity of SCMs)**.** *Let $\mathcal{M}, \mathcal{M}'$ be two SCMs. We say that $\mathcal{M}, \mathcal{M}'$ are similar at layer i, written $\mathcal{M} \sim_i \mathcal{M}'$, if the models agree on every $\mathcal{L}_i$ query.*

Theorem 2.1 (Causal Hierarchy Theorem (Informal) [9])**.** *For almost all SCMs $\mathcal{M}$, and levels $i < j$, there exists a $\mathcal{M}'$ such that $\mathcal{M} \sim_i \mathcal{M}'$ but $\mathcal{M} \not\sim_j \mathcal{M}'$.*

That is, data corresponding to queries at level i is almost never sufficient for us to draw conclusions about queries at higher levels. The immediate consequence is that, unfortunately, observational data is not enough on its own for us to be able to answer causal (interventional and counterfactual) queries. This is problematic, as statistical inference is reliant on the principle that all information necessary to answer a query of interest can be found in the data distribution. Even in settings where are able to obtain data from interventional distributions, we need to understand how such data affects the query that we are actually interested in (which may involve a different intervention). This motivates the need for *causal inference*.

2.1.3 Causal Inference

This process of characterizing data as the output of a causal query from the underlying (unobserved) SCM, in order to reason about the output of other causal queries on that SCM, is at the core of *causal inference*. Broadly, causal inference consists of two problems: *identifiability*, and *estimation*. Identifiability asks whether the available data is sufficient to uniquely determine the answer to the query of interest, while estimation is the algorithmic process of answering the query from the given data.

In this section, we focus on inference for queries in each of the layers of the causal hierarchy in turn. First, we describe how each layer can be naturally characterized using *abstractions* of structural causal models; that is, coarser versions of SCMs that can answer queries at the respective layer of the hierarchy. Next, we discuss identifiability issues; namely, what assumptions we need to make to be able to perform cross-layer inferences. Finally, we describe approaches to estimation, focusing on the underlying principles.

2.1.3.1 $\mathcal{L}_1$ Inference

$\mathcal{L}_1$ (associational) queries correspond to properties of the observed distribution $p_{\mathcal{M}}(\boldsymbol{V})$ of a SCM. As such, the multivariate probability distribution $p_{\mathcal{M}}(\boldsymbol{V})$ itself is an abstraction of the SCM capable of answering any $\mathcal{L}_1$ query. In the following, we describe the process of $\mathcal{L}_1$ inference with a particular focus on Bayesian networks, a popular graphical representation of probability distributions.

Definition 2.8 (Bayesian Network)**.** *A Bayesian network (BN) $\mathcal{N} = (G, Pr)$ over variables $\boldsymbol{V}$ is a pair consisting of a directed acyclic graph (DAG) G over nodes $\boldsymbol{V}$, and a collection of conditional probability distributions (CPDs) $Pr_i(V_i|\mathrm{pa}_G(V_i))$ for each $V_i \in \boldsymbol{V}$, where $\mathrm{pa}_G(V_i)$ are the parents of V_i in G.*

A Bayesian network specifies a joint distribution over $\boldsymbol{V}$ defined by the factorization:

$$p_{\mathcal{N}}(\boldsymbol{V}) = \prod_{i=1}^{d} Pr_i(V_i|\mathrm{pa}_G(V_i)) \tag{2.7}$$

As defined, Bayesian networks provide an explicit construction of the joint probability distribution they represent. The graph of a Bayesian network places restrictions on the distributions $p(\boldsymbol{V})$ that can be expressed. For example, a fully connected Bayesian network graph (where V_i has the parent set $\{V_j|j < i\}$) can express any multivariate distribution $p(\boldsymbol{V})$, while a Bayesian network with no edges necessarily corresponds to a fully factorized distribution (i.e. every variable is independent). That is, the *absence* of edges corresponds to (conditional) independence assumption(s).

At first glance, Bayesian networks share many similarities with structural causal models; namely, they can both be interpreted graphically, where each node (variable) depends on the values of its parents, through a causal mechanism or conditional probability distribution. They also both define a joint distribution $p(\boldsymbol{V})$ over the endogenous variables $\boldsymbol{V}$. However, the key distinction is that Bayesian networks *only* have semantics relating to the observational distribution ($\mathcal{L}_1$), while structural causal models additionally define interventional and counterfactual distributions ($\mathcal{L}_2, \mathcal{L}_3$). That is, the Bayesian network graph, and each CPD in a Bayesian network, should not be interpreted as a *causal* relation, but merely a convenient means of specifying (constraints on) the joint distribution. This is also the reason why there is no need for latent/exogenous variables in the definition of a BN: there is no need to distinguish, for example, between an exogenous variable confounding two endogenous variables, and the variables themselves having a parent-child relationship.

Now, we turn to $\mathcal{L}_1$ inference from observational data. In this case, there are no identifiability issues, as we are using $\mathcal{L}_1$ data to answer $\mathcal{L}_1$ queries, and this simply reduces to *statistical* inference[1]. In statistical inference, the general procedure is to fit/learn a probabilistic model (e.g. Bayesian network) from data, and then reason on that model to answer the query of interest. Examples of specific queries of interest include conditional/predictive distributions $p(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x})$ and their mean $\mathbb{E}[\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x}]$ and mode $\max_{\boldsymbol{y}} p(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$.

[1]On the other hand, statistical models often make assumptions on the distribution, such as the conditional independences implied by a Bayesian network graph.

2.1.3.2 $\mathcal{L}_2$ Inference

$\mathcal{L}_2$ queries correspond to properties of the intervened distributions $p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}'})$. These typically correspond to some experimental setting of the data generating system which is of interest, but different to the current reality from which data is observed; for example, a world where we force all smokers to stop smoking, or collecting data from a randomized sample of the population instead of survey respondents. If we had access to data from this distribution, then, like in $\mathcal{L}_1$ inference, this would reduce to statistical inference; in fact, this is the principle of a randomized controlled trial (RCT), which aims to ensure that treatments are assigned randomly. However, this is often not possible, for example, due to economic or ethical reasons. Instead, it is common to consider the setting where we have access to either $\mathcal{L}_1$ (observational) data or a combination of $\mathcal{L}_1$ and $\mathcal{L}_2$ (experimental) data, where the $\mathcal{L}_2$ data corresponds to *different* interventions to the query. In this setting, there is a simple extension to Bayesian networks that can be used to represent not only the observational distribution, but also interventional distributions, known as a *causal Bayesian network*.

Definition 2.9 (Causal Bayesian Network)**.** *A causal Bayesian network $\mathcal{CBN} = (G, Pr)$ is a Bayesian network with the additional semantics that, given any collection of CPDs Pr', the interventional distribution is defined as:*

$$p_{\mathcal{CBN}}(\boldsymbol{V}_{Pr'}) = \prod_{i=1}^{d} Pr'_i(V_i|\text{pa}_G(V_i)) \tag{2.8}$$

As in SCMs, we can define hard interventions as a special case, where we set a subset of variables $\boldsymbol{X}$ to the value $\boldsymbol{x}$. The corresponding interventional distribution is given by $p_{\mathcal{CBN}}(\boldsymbol{V}_{\boldsymbol{x}}) = \prod_{i=1, V_i \notin \boldsymbol{X}}^{d} Pr(V_i|\text{pa}_G(V_i))|_{\boldsymbol{X}=\boldsymbol{x}}$, where $|_{\boldsymbol{X}=\boldsymbol{x}}$ denotes instantiating the variables $\boldsymbol{X}$ with $\boldsymbol{x}$.

In contrast to Bayesian networks, in causal Bayesian networks we do interpret parents of a variable as being direct causes, and the ancestor-descendant relation as being a cause-effect relation. Though seemingly innocuous and straightforward, causal Bayesian networks make the significant assumption that all interventional (causal) distributions should follow this truncated factorization. In particular, notice that in a Bayesian network, a variable is independent of all other variables which are not its effects (non-descendants) given its parents; in causal Bayesian networks, this means that in a hard intervened distribution, the variables $\boldsymbol{X}$ are independent of its non-effects. This is often known as the *causal Markov assumption.*

Under what circumstances is the causal Markov assumption justified? Leveraging the theory of SCMs, this amounts to understanding how interventional distributions of a causal Bayesian network relate to the $\mathcal{L}_2$ predictions of an SCM.

Theorem 2.2 (CBN-SCM [9])**.** *Given a Markovian SCM $\mathcal{M}$ with causal diagram G, there exists a causal Bayesian network $\mathcal{CBN} = (G, Pr)$ such that $\mathcal{M} \sim_2 \mathcal{CBN}$.*

The Markovian assumption is key; causal Bayesian networks are justified when there are no unobserved common causes (exogenous variables). The theorem says that given any *Markovian* SCM, there is a corresponding abstract representation in the form of a CBN that is sufficient to encode all $\mathcal{L}_2$ properties. Thus, if we are interested in $\mathcal{L}_2$ inference, rather than inferring a SCM, it is sufficient to infer a CBN, in order to answer any $\mathcal{L}_2$ query.

Inferring a causal Bayesian network from $\mathcal{L}_1$ (and possibly also $\mathcal{L}_2$) data, including both the graph and distribution, is a challenging problem that is typically treated separately in *causal discovery* or *causal structure learning.* For now, suppose that we know the CBN graph (obtained, for example, through domain knowledge or causal discovery). Remarkably, in this case, $\mathcal{L}_1$ data is sufficient to identify the CBN, and thus, all $\mathcal{L}_2$ queries:

Corollary 2.1 ($\mathcal{L}_2$ identifiability in Markovian SCMs [9])**.** *Given the diagram of a Markovian SCM $\mathcal{M}$, and the observational distribution $p(\boldsymbol{V})$, all $\mathcal{L}_2$ queries are identifiable.*

This result follows simply by noting that a CBN is defined by its graph and conditional probability distributions (CPD), and the CPDs follow from the joint distribution $p(\boldsymbol{V})$. This gives a simple procedure for performing $\mathcal{L}_2$ inference in Markovian SCMs given the diagram; learn the CBN distribution from data, apply the truncated factorization, and then compute the query from $Pr_{\boldsymbol{x}}(\boldsymbol{V})$.

We now turn to the semi-Markovian case, where there are unobserved common causes of endogenous variables. It is not possible to identify interventional distributions without some assumptions on the nature of the confounding. Thus, the question of identifiability is typically posed as follows: given the causal diagram of a semi-Markovian SCM (including bidirected edges) as an assumption, and a $\mathcal{L}_2$ query, is $\mathcal{L}_1$ data (or a combination of $\mathcal{L}_1, \mathcal{L}_2$ data) sufficient to identify the query?

To answer this question, Pearl derived a systematic set of rules called the *do-calculus* [130], for transforming probabilistic expressions involving interventions and observations into new but equivalent expressions. Each of the rules of the do-calculus consists of a *graphical criterion* on the causal diagram under which they can be applied, and the corresponding transformation. The goal is then to derive an expression not involving interventions by applying do-calculus rules. To automate this process, a polynomial-time algorithm for identifying causal effects was proposed by [177], and was later shown to be able to identify all identifiable distributions [168, 80].

2.1.3.3 $\mathcal{L}_3$ inference

$\mathcal{L}_3$, or counterfactual inference, corresponds to answering queries involving outcomes in more than one counterfactual world/intervention, i.e. $p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{V}_{\boldsymbol{F}^{(n)}})$. In this case, knowledge of the entire SCM, including functional relationships and the distribution of the exogenous variables, is typically required. Given a fully specified SCM, it is possible to express a counterfactual inference query as a marginal inference query on a *twin* network [8] (more generally, a n-world network).

Unfortunately, by the causal hierarchy theorem, if we do not have access to the underlying SCM, even $\mathcal{L}_2$ (i.e. experimental/interventional) data is not sufficient to make $\mathcal{L}_3$ inferences. There are two main approaches to this problem. First, we can make (strong) assumptions on the form of the SCM, which enable identifiability of the target query from data. A prominent example of this approach is the property of *monotonicity* [131] for binary variables, and *counterfactual stability* [124] for categorical variables, for identifiability of probabilities of causation. Alternatively, we can look to bound the query of interest under all possible SCMs consistent with the data [205].

2.1.4 Summary

To summarize, we have described the structural causal model framework for modelling causal systems. The core task of causal inference can then be broadly described as follows: given some information about the SCM (e.g. generated observational data, causal diagram) as input, answer some query about the SCM as output. Key to the formulation of SCMs is the *information hierarchy*, which specifies three different levels of queries that require increasingly more fine-grained knowledge of the SCM. This means that it is not possible, for example, to uniquely extract $\mathcal{L}_2$ information given only $\mathcal{L}_1$ information. More generally, we can speak of identifiability, which asks whether the input information is sufficient to uniquely determine the desired output information. However, even if a query is identifiable, that does not mean that we can tractably compute it.

The key question we are concerned with in this thesis is how to computationally reason about the input information, in order to answer a causal query of interest. This is not a single problem but rather a vast space of problems, induced by (i) what input information we are assumed to have; and (ii) the space of possible queries that we might want to answer (functions of the distributions in Definition 2.6). The perspective we take in this thesis is to organize by the input information we have, build/learn a *model* based on this information, and then use the model to reason about queries efficiently. In particular, in Chapter 4, we study the case where a fully specified causal Bayesian network or SCM is available as input; in Chapter 5, we will study the more challenging (but also more realistic) setting where we have access to observational data ($\mathcal{L}_1$), and the causal diagram; and in Chapter 6, we assume only access to observational data, and model uncertainty over the causal diagram. In order to construct models for the input information, we will employ and extend tools from *tractable probabilistic modelling*, which we introduce in the next section.

2.2 Tractable Probabilistic Models

In this section, we introduce the concept of tractable probabilistic modelling (TPMs), which is an approach to modelling probability distributions that focuses on designing model classes that enable efficient (i.e. polynomial time) and reliable (i.e. exact or approximate with guarantees) reasoning. In particular, we provide background on tractable probabilistic circuits, the key computational framework underlying the work in this thesis.

2.2.1 Probabilistic Models and Queries

A probabilistic model $\mathcal{P}$ is a particular representation of a (possibly unnormalized) distribution over a set of random variables $\boldsymbol{V}$. That is, it encodes some non-negative function $p_{\mathcal{P}} : \text{val}(\boldsymbol{V}) \to \mathbb{R}^{\geq 0}$. We have already seen an example of a probabilistic model in the previous section on causality: namely, Bayesian networks. Bayesian networks specify a distribution $p_{\mathcal{N}}$ by individually specifying mechanisms/CPDs for each variable, according to a directed acyclic graph. Given a model, the task of *reasoning* is to extract some useful information from the model. This can be formalized through the definition of queries, which are functions of the distribution that the model encodes. For example, the evidence query requires the evaluation of the distribution $p_{\mathcal{P}}$:

Definition 2.10 (Evidence Queries). *Given a probabilistic model $\mathcal{P}$, an evidence query is of the form $p_{\mathcal{P}}(\boldsymbol{v})$, where $\boldsymbol{v}$ is an instantiation of $\boldsymbol{V}$.*

A generalization of the evidence query is the marginal query, which instead asks for the probability of a partial assignment $\boldsymbol{w}$. This is often useful in practice, as $\boldsymbol{w}$ may represent some *event* that we are interested in, that does not involve the variables $\boldsymbol{V} \setminus \boldsymbol{W}$ (we say that these variables have been marginalized out).

Definition 2.11 (Marginal Queries). *Given a probabilistic model $\mathcal{P}$, a marginal query is of the form $p_{\mathcal{P}}(\boldsymbol{w})$, where $\boldsymbol{w}$ is an instantiation of a subset $\boldsymbol{W} \subseteq \boldsymbol{V}$ of variables.*

Another important query is the MAP (maximum a posteriori) query[2], which asks for the maximum value of $p_{\mathcal{P}(\boldsymbol{v})}$ over all instantiations of $\boldsymbol{V}$ (i.e. the mode of the distribution), possibly in conjunction with some evidence/event.

[2]In the graphical model literature, the MAP query is often referred to as MPE, while marginal MAP is referred to as MAP. In this thesis, we will use the terms MAP and marginal MAP (MMAP) to avoid ambiguity and in line with recent literature [108, 114, 32].

Definition 2.12 (MAP Queries)**.** *Given a probabilistic model $\mathcal{P}$, a MAP query is of the form* $\max_{\boldsymbol{Q}} p_{\mathcal{P}}(\boldsymbol{Q}, \boldsymbol{e})$*, where* $\boldsymbol{Q}, \boldsymbol{E}$ *partition* $\boldsymbol{V}$*, and* $\boldsymbol{e}$ *is an instantiation of* $\boldsymbol{E}$*.*

We can also extend the MAP query to cases where some variables have been marginalized out.

Definition 2.13 (Marginal MAP Queries)**.** *Given a probabilistic model $\mathcal{P}$, a marginal MAP (MMAP) query is of the form* $\max_{\boldsymbol{Q}} p_{\mathcal{P}}(\boldsymbol{Q}, \boldsymbol{e})$ *where* $\boldsymbol{Q}, \boldsymbol{E} \subseteq \boldsymbol{V}$*, and* $\boldsymbol{e}$ *is an instantiation of* $\boldsymbol{E}$*.*

All of these queries (as well as others) can be important to be able to reason about depending on the application domain and meaning of the variables. However, models may differ in terms of their tractability on each of these queries, even if they represent the same distribution p. The key thrust of tractable probabilistic modelling is to understand what makes a particular query tractable for particular models. To this end, it will often be useful to talk about a class of models $\boldsymbol{\mathcal{P}}$, and classes of queries $\boldsymbol{q}$ which is simply a semantic grouping of models and queries.

Definition 2.14 (Tractability)**.** *We say that a class of queries $\boldsymbol{q}$ is tractable for a class of probabilistic models $\boldsymbol{\mathcal{P}}$, if for any query $q \in \boldsymbol{q}$ and any model $\mathcal{P} \in \boldsymbol{\mathcal{P}}$, it is possible to compute q on $\mathcal{P}$ in $O(poly(|\mathcal{P}|))$ time, where $|\mathcal{P}|$ is the size of the model.*

For example, the class of evidence queries is tractable on Bayesian networks. However, (the decision variants of) MAP and marginal queries are `NP`-complete and `PP`-complete respectively.

2.2.2 Probabilistic Circuits

The recently introduced framework of probabilistic circuits (PC) [33] is a unifying language for tractable probabilistic models, which encompasses many previously known tractable models, such as bounded-treewidth Bayesian networks [34], arithmetic circuits [42], sum-product networks [146], cutset networks [149], and probabilistic sentential decision diagrams [89]. Probabilistic circuits are computational graphs which represent a non-negative function over a set of variables, often interpreted as a (possibly unnormalized) probability distribution.

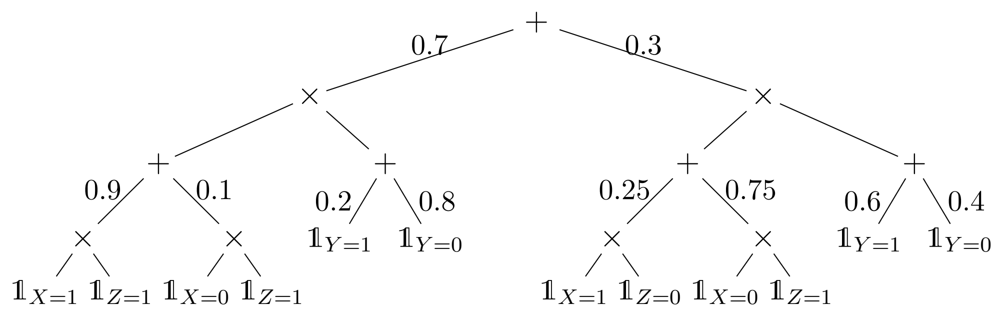


Figure 2.1: Decomposable, smooth and deterministic PC. Edges are labelled with sum node weights ω, while leaf nodes are labelled with their functions g_L.

Definition 2.15 (Probabilistic Circuit)**.** *A probabilistic circuit (PC) over variables $\boldsymbol{V}$ is a triple $\mathcal{C} = (G, \boldsymbol{\omega}, \boldsymbol{g})$, which encodes a non-negative function $p_{\mathcal{C}}(\boldsymbol{V})$, where G is the structure of the PC, $\boldsymbol{\omega}$ are the parameters of the PC, and $\boldsymbol{g}$ are the leaf node functions.*

Definition 2.16 (Syntax of Probabilistic Circuits)**.** *The structure G of a PC is a rooted directed acyclic graph consisting of three types of nodes N: leaf L, sum T and product P. Leaf nodes L are leaves of the graph, while each internal node (sum or product) N has a set of children, denoted* $\mathrm{ch}(N)$*. Each sum node T has a parameter/weight $\omega_{T,i} \in \mathbb{R}^{\geq 0}$ associated with each of their children N_i. Each leaf node L encodes a non-negative function $g_L : \phi(L) \to \mathbb{R}^{\geq 0}$ over a subset of variables $\phi(L) \subseteq \boldsymbol{V}$, known as its* scope*.*

In contrast to probabilistic graphical models such as Bayesian networks, the nodes in a PC do not represent the variables $\boldsymbol{V}$, but rather distributions over those variables, which are intermediate units used to construct the final distribution. In that sense, PCs are computational graphs that directly specify how the probability distribution they represent is evaluated.

Definition 2.17 (Semantics of Probabilistic Circuits)**.** *The function encoded by each node N is defined by:*

$$p_N(\boldsymbol{V}) := \begin{cases} g_L(\boldsymbol{V}) & \textit{if } N \textit{ is a leaf } L \\ \prod_{N_i \in \mathrm{ch}(P)} p_{N_i}(\boldsymbol{V}) & \textit{if } N \textit{ is a product } P \\ \sum_{N_i \in \mathrm{ch}(T)} \omega_i p_{T,N_i}(\boldsymbol{V}) & \textit{if } N \textit{ is a sum } T \end{cases} \tag{2.9}$$

The function encoded by the circuit, $p_{\mathcal{C}}(\boldsymbol{V})$, is defined to be the function encoded by its root node R. The size of a circuit, denoted $|\mathcal{C}|$, is defined to be the number of edges in the circuit.

As an example, in Figure 2.1 we show a probabilistic circuit over variables $\boldsymbol{V} = \{X, Y, Z\}$, where the edges linking sum nodes to their children have been labelled with the corresponding weight, and each leaf node has been labelled with its function, e.g. $g_L(X) = \mathbb{1}_{X=1}$.

We now turn to the tractability of probabilistic circuits as a model class. It is typically assumed that the leaf node functions are tractable; that is, we can compute the query on $p_L(\boldsymbol{V})$ in constant time. In practice, leaf node functions are often univariate density functions, e.g. a univariate Gaussian or discrete distribution. Under this assumption, the evidence query (that is, computing the density $p_{\mathcal{C}}(\boldsymbol{v})$ for some instantiation $\boldsymbol{v}$ of $\boldsymbol{V}$) is tractable in linear time for any probabilistic circuit. The corresponding algorithm involves a bottom-up (i.e. starting from the leaves) evaluation of $p_N(\boldsymbol{v})$ for every node in the PC, according to Equation (2.9). However, we are typically interested in classes of PCs that support tractable reasoning on more complex queries. To define these classes, we need to define the scope and support of a PC node.

Definition 2.18 (Scope and support of PC node)**.** *The scope of an internal node N is recursively defined by $\phi(N) := \bigcup_{N_i \in \mathrm{ch}(N)} \phi(N_i)$, and is the set of variables p_N specifies a function over. The support of any node N is defined as supp$(N) := \{\boldsymbol{w} \in$ val$(\phi(N)) : p_N(\boldsymbol{w}) > 0\}$, i.e. the set of all instantiations of its scope s.t. p_N is positive.*

The tractability of probabilistic circuits depends on constraints on the scope of the PC nodes (which we call scope properties) and support of the PC nodes (which we call support properties).

Definition 2.19 (Decomposability)**.** *A PC is decomposable if the children of a product node P have distinct scopes, i.e. $\forall N_1, N_2 \in \mathrm{ch}(P)$, $\phi(N_1) \cap \phi(N_2) = \emptyset$.*

Definition 2.20 (Smoothness)**.** *A PC is smooth if the children of a sum node T have the same scope, i.e. $\forall N_1, N_2 \in \mathrm{ch}(T)$, $\phi(N_1) = \phi(N_2)$.*

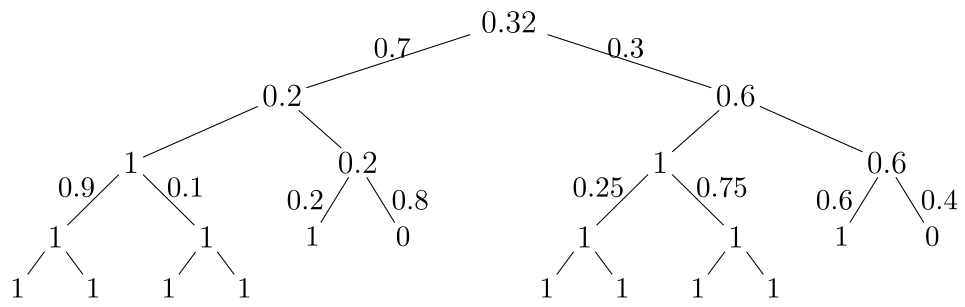


Figure 2.2: Computing the marginal $p_{\mathcal{C}}(Y = 1)$ for the PC in Figure 2.1.

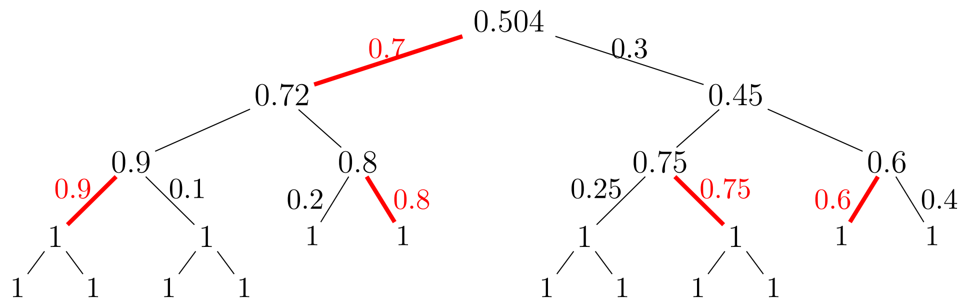


Figure 2.3: Computing the MAP $\max_{X,Y,Z} p_{\mathcal{C}}(X, Y, Z)$ for the PC in Figure 2.1.

Intuitively, decomposability requires that product nodes represent a factorized distribution over disjoint sets of variables, while smoothness requires that product nodes represent a mixture distribution over components with the same variables. Decomposability and smoothness together enable tractable marginal inference; that is, for any subset $\boldsymbol{W} \subseteq \phi(N)$ of the scope of a node N, we can compute $p_N(\boldsymbol{W})$ efficiently, where $p_N(\boldsymbol{W}) := \sum_{\phi(N)\setminus \boldsymbol{W}} p_N(\phi(N))$ is the *marginal* of the function. The corresponding algorithm is similar to the bottom-up evaluation of the evidence query. All of the circuits which we study in this thesis will satisfy these two properties. For example, the PC example in Figure 2.1 satisfies both properties. We show in Figure 2.2 the linear-time computation of the marginal $p_{\mathcal{C}}(Y = 1)$, where we label each node with its value under this computation; the output is the value at the root node 0.32.

As for support properties, by far the most common is determinism, which intuitively states that each child of a sum node is associated with different values of the variables:

Definition 2.21 (Determinism). *A PC is deterministic if the children of a sum node T have disjoint supports, i.e. $\forall N_1, N_2 \in \mathrm{ch}(T)$, supp$(N_1) \cap$ supp$(N_2) = \emptyset$.*

Determinism (together with decomposability) enables tractability of the MAP inference query, i.e. computing $\max_{\boldsymbol{V} \setminus \boldsymbol{E}} p_N(\boldsymbol{V} \setminus \boldsymbol{E}, \boldsymbol{e})$ for some instantiation $\boldsymbol{e}$ of a set of evidence variables $\boldsymbol{E} \subseteq \boldsymbol{V}$. Intuitively, this is because we can independently optimize for the most likely instantiation in each child of a sum node, and then select the most likely child. Determinism is also helpful for parameter learning in that maximum likelihood estimation of the PC parameters becomes tractable. The example PC in Figure 2.1 is deterministic, and we show in Figure 2.3 the corresponding MAP computation, where one child is chosen at every sum node. The maximum probability instantiation can be obtained by following the chosen child at every sum node (highlighted in red); in this case, it is $X = 1, Y = 0, Z = 1$.

Though other support properties exist, such as the decision property, partitioned determinism, and marginal determinism, their implications for tractable probabilistic reasoning are less well studied, and consequently they are less well known/commonly enforced in tractable probabilistic modelling. However, in Chapters 4 and 5, we will show that additional support properties besides determinism are vital for tractable causal reasoning.

2.2.3 Summary

To summarize, we have introduced the key concepts of tractable probabilistic modelling, namely, *models*, which are specific representations of probability distributions (more generally, non-negative functions), and the *queries* they can answer in polynomial time. In particular, probabilistic circuits are a framework for TPMs whose tractability for various inference queries can be characterized by their structure properties, such as decomposability, smoothness, and determinism.

As models of the (observational) probability distribution, from a causal perspective, probabilistic circuits can only answer $\mathcal{L}_1$ queries; however, they are distinguished from other types of probabilistic models (e.g. intractable generative models, discriminative models) in that they form a consistent, flexible and tractable knowledge base; i.e. they can compute many classes of queries exactly and efficiently, rather than just a single query. Such a knowledge base for causal modelling would have even greater appeal, as the space of $\mathcal{L}_2$ and $\mathcal{L}_3$ queries is much larger than for $\mathcal{L}_1$ queries (due to the interventional semantics). In the following chapters, for each of the settings described in Section 2.1.4, we will investigate to what extent it is possible to develop circuit-based models that can compute large classes of $\mathcal{L}_2$ and $\mathcal{L}_3$ queries efficiently.

Chapter 3

Literature Review

Contents

In this Chapter, we review the literature on tractable probabilistic modelling and computational problems in causality, in order to provide further context for the research questions that we tackle in the upcoming Chapters.

3.1 Tractable Probabilistic Models

The term *tractable probabilistic modelling* refers broadly to an approach to probabilistic learning and reasoning, in which the representation of the probability distribution is designed to enable tractable exact computation of a set of inference queries of interest. Many such model classes now exist, from those admitting tractable likelihoods (e.g. Bayesian networks [129], autoregressive models, normalizing flows [152]), to tractable marginals (e.g. sum-product networks [146]), to tractable computation of various advanced inference queries (e.g. probabilistic sentential decision diagrams [89]). In this section, we survey the literature on tractable probabilistic modelling in order to contextualize the contributions of this thesis.

3.1.1 The Spectrum of Tractable Probabilistic Models

In the following, we review the most important classes of tractable probabilistic models, starting from classical probabilistic graphical models, and then moving into more recent proposals inspired by machine learning.

Bayesian networks [129] are directed probabilistic graphical models that specifies a distribution over a set of variables as the product of local conditional probability distributions for each variable. Evaluating the probability (density) of any instantiation of the variables can thus be performed in linear time. In contrast, computing densities in undirected probabilistic graphical models (Markov networks) requires computing the partition function (normalizing constant), which is `NP`-hard [93]. However, it is often of more interest to compute the probability of some event, or marginal, of the joint distribution. This task is also known to be `NP`-hard through a reduction from the Boolean satisfiability problem [36]. For this reason, Bayesian networks have not historically been considered tractable models, and much effort has been expended into developing effective exact and approximate inference algorithms.

The complexity of exact (marginal) inference algorithms for Bayesian networks can be bounded in practice by the *treewidth* of its graph [99]. For this reason, one way to ensure tractable inference is to restrict to low or bounded-treewidth Bayesian networks [34, 7, 57, 94] or mixtures thereof [117], where marginal inference is tractable by design. Such methods can now scale to learning networks containing thousands of variables [157]. Unfortunately, bounded treewidth is not always a tenable assumption, as it inevitably imposes conditional independences on the distribution that restricts expressivity. To efficiently represent Bayesian networks with high treewidth, it is possible to exploit *determinism*[1] [101] and *context-specific independences* (CSI) [15] to more compactly express the conditional probability distributions defining a Bayesian network. These properties of the distribution, sometimes referred to as *local structure*, can then be exploited for efficient inference [42, 26].

The seminal work of [42] showed that the process of inference in Bayesian networks can be traced and represented as an *arithmetic circuit* (AC), which is a representation

[1]Determinism in this context refers to the determinism of conditional probability distributions of BNs; not to be confused with determinism as a circuit property.

of the Bayesian network distribution that admits linear-time marginal and MAP inference. Similar proposals, based on extensions of the ordered binary decision diagram (OBDD) in automated verification, were also introduced in the form of probabilistic decision graphs [16, 83] and AND/OR multi-valued decision diagrams [115]. All of these can be interpreted as decomposable, smooth, and deterministic probabilistic circuits. Later, it was shown that these circuits could be learned directly from data, exploiting context-specific independence for compactness [84, 111].

The insights from these earlier works have spurred further research into the precise properties of circuits required to enable tractable inference for different queries. In particular, we highlight two major advances in this area. The first was the introduction of sum-product networks (SPN) [146], which relaxed the requirement for circuits to satisfy determinism, while retaining decomposability and smoothness to ensure tractable marginal inference[2]. This has been shown to result in potentially exponentially more succinct circuits, but at the cost of tractable MAP inference [31]. SPNs can be interpreted as deep, hierarchical latent variable models [136] that satisfy the necessary structure for tractable inference. Secondly, probabilistic sentential decision diagrams (PSDD) [89] provide an implementation of the stronger properties and structured decomposability and partitioned (strong) determinism [142, 144] for probabilistic models, based upon logical circuits known as sentential decision diagrams (SDDs) [44]. These properties were later shown to additionally enable tractable probabilistic operations, such as products of circuits [162], KL-divergence of two circuits[105], and expected predictions [87].

Much effort has also been made in developing more scalable architectures and learning algorithms for probabilistic circuits, as well as adapting them for new tasks. The task of learning PCs can be divided into *parameter learning*, which relates to learning the weights of sum nodes in the PC, and *structure learning*, which relates to learning the topology of the PC graph. Prominent algorithms for learning the structure of probabilistic circuits include top-down learning algorithms, such as LearnSPN [66] and ID-SPN [154], that hierarchically split the root sum node using (subsets of)

[2]In the sum-product network literature, smoothness is often known as *completeness*, and determinism as *selectivity*. In addition, a condition weaker than decomposability called *consistency* is used, but this has no practical implication on the size of circuits [138].

the data, in a divide-and-conquer approach; and bottom-up learning algorithms [135, 103, 41], which iteratively grow the circuit from the leaf node distributions. Parameter (weight) learning for probabilistic circuits can be performed using maximum likelihood estimation for deterministic circuits, while the EM algorithm [51] can be used for non-deterministic circuits [146, 211]. A recent trend in PC learning has been to utilize *random* but scalable structures upon which parameter learning can be performed; works in this direction include RAT-SPN [139] and EiNets [137], which are SPN architectures, and XPCs [53], which can additionally impose structured decomposability and determinism.

While probabilistic circuits are typically thought of as generative models describing a joint distribution, they can also be adapted for learning conditional distributions of some set of output variables, given a set of input variables. In the case where the output set is a singleton (e.g. classification), it is possible to learn probabilistic circuits over the input variables in a discriminative manner [65, 1, 139, 104], with the output being interpreted as the classification probability. For output sets with more variables, there have been a number of proposals. Conditional cutset networks [148] and conditional SPNs [161] employ external function approximators (e.g. neural network) to express PC weights as a function of the input variables. On the other hand, conditional PSDDs [163] explicitly include the input variables as part of the tractable circuit; this extra tractability is important, for example, for being able to multiply two conditional distributions [164].

A class of tractable models that is not efficiently captured by graphical models or probabilistic circuits exploiting local structure is the determinantal point process (DPP) [97]. (Discrete) DPPs efficiently express probability distributions exhibiting *global negative dependence*; that is, distributions with negative correlations between any subset of variables. Marginal inference in DPPs corresponds to matrix operations which can be performed in polynomial time. Unfortunately, such distributions are not possible to represent compactly as probabilistic circuits [203]; very recently, a new tractable model known as a probabilistic generating circuit [204] was introduced that generalizes both circuit-based models and DPPs.

Finally, we touch briefly on probabilistic models based on neural networks, which have shown increasingly impressive performance in terms of their modelling capabilities in recent years. Most generative models that have been proposed, such as variational autoencoders [88], generative adversarial networks [70], and diffusion models [170, 171] do not admit tractable density evaluation and can only be sampled from. Autoregressive models [182, 18], which explicitly predict the density of each variable (token) conditional on the past, and normalizing flows [151], which utilize invertible transformations, are notable exceptions that have tractable densities.

3.1.2 Tractable Probabilistic Models for Causality

We now review the literature on tractable probabilistic models for causal reasoning. In this context, causal reasoning inherits many of the challenges of probabilistic reasoning, but also poses new questions, such as relating to identifiability and tractability of interventional and counterfactual queries. Existing works can broadly be divided into two strands of research, which differ in terms of their motivation and methods. The first strand extends the knowledge compilation approach to probabilistic inference, to compiling causal models such as causal Bayesian networks and structural causal models. The second strand focuses instead on general (learned) probabilistic circuits, and under what circumstances they can be interpreted causally and/or used for tractable causal reasoning.

In the first strand, the underlying causal model is generally assumed to be fully known, such that there are no identifiability concerns. The main focus is thus on identifying and computing representations of the model that are *tractable for causal queries* and *succinct*. Such a representation can be obtained by applying the symbolic Bayesian network compilation methods of [42] to causal Bayesian networks and structural causal models. By exploiting the semantic relationship between circuit parameters and Bayesian network parameters, it has been noted that the compiled circuits can be used for tractable interventional marginal reasoning [147, 191, 46] [3]. This approach has been extended to counterfactual reasoning by instead compiling a twin network [8] representation of SCMs [73]. For succinctness, a new compilation/inference

[3]I am a co-author on the second paper, on which Chapter 4 is partially based.

technique has recently been introduced that exploits the functional mechanisms present in SCMs [46, 45, 27]; the resulting compilations are exponential in the so-called causal treewidth, which is always less than the treewidth.

The second strand of research aims to exploit general probabilistic circuits, which, as in probabilistic inference, are not limited by treewidth and can be learned from data. The challenge, however, is that such circuits no longer have a direct reference point to a causal model (i.e. CBN or SCM), and consequently they do not have direct causal semantics [210]. One approach is to use PCs in a purely probabilistic manner, for example to model conditional probability distributions in CBNs and SCMs. These models can then be used to compute causal quantities in conjunction with causal assumptions: for example, to adjust for observed confounders via the backdoor formula [202]. However, the resulting computation is intractable unless there is massive determinism (that is, most instantiations have zero probability) [72], which could result in violations of the positivity assumption in causal inference. The other approach is to interpret a circuit as representing a causal model. Unfortunately, [126] showed that known interpretations of common circuits (SPNs and PSDDs) as graphical models lead to degenerate causal models. As a workaround, [201] proposed to use different weights on the same circuit to model different interventional distributions, with the weights predicted by neural networks [161]; however, there is no guarantee that the resulting interventional distributions are consistent with any causal model.

Despite the disparity in these approaches, the common thread is that one is attempting to extract an interventional distribution from a circuit modelling the observational distribution. In the former case, we employ external machinery (e.g. backdoor adjustment) to define the interventional distribution, while in the latter case, we assume that the interventional distribution(s) can be somehow encoded into the circuit such that reasoning remains tractable over the interventional distribution. This raises the fundamental question: for what circuits is the interventional distribution tractable to compute? In Chapter 5, we will explore this question in depth.

3.2 Computational Causality

We now turn to the broader literature on causality, with a particular focus on computational problems. First, we review work on causal identification and reasoning over the space of SCMs and compare and contrast to the questions in this thesis. Then, we review work on causal structure learning, which is of importance in Chapter 6.

3.2.1 Structural Causal Models and Causal Identification

We begin by considering the problem of identification of interventional distributions, given access to a causal diagram and observational data. An interventional distribution is said to be *identifiable* if it is the same for all SCMs consistent with the diagram and data. The identification problem is then to develop an algorithm for (i) deciding whether a given interventional distribution is identifiable given the diagram; and (ii) if it is, returning a formula giving the interventional distribution in terms of the observational distribution. An identification algorithm is said to be *sound* if it only returns True (and an identification formula) when the interventional distribution is identifiable, and *complete* if it always does so.

Pearl's celebrated do-calculus [130] provides a set of rules (together with basic probabilistic operations) for soundly transforming interventional distributions into functions of the observational distribution based on the structure of the diagram. Later, a sound and complete poly-time algorithm known as the ID algorithm [177] was introduced for automatically applying the do-calculus to arbitrary causal diagrams and interventional distributions, effectively solving the identification problem from a computational perspective. More recent works have considered extensions to the identification problem. In [38], a set of rules known as the σ-calculus was developed for the problem of identification of interventional distributions with stochastic rather than hard interventions; follow-up work resulted in an efficient sound and complete algorithm [37] in this setting. In the setting where one has additional information on context-specific independences involving the latent and observed variables in the SCM, [179] showed that more interventional distributions become identifiable. Unfortunately, the resulting identification problem becomes NP-hard. Finally, in the so-called general identifiability setting, where additional information in the form of a set of interventional distributions (with different interventions to the target interventional distribution) is available, a sound and complete polytime algorithm exists [102, 90].

One can also define other reasoning problems over the space of structural causal models based on logic. Recently, in [81, 121], languages for expressing causal statements on SCMs were introduced, as an extension to probabilistic languages for probability functions[59]. These languages allow for expressing interventions, sums, conditionals, and products, in addition to the standard propositional logic operators. The resulting satisfiability problems (i.e. existence of a SCM satisfying the formula) were shown to be NP-complete or $\exists\mathbb{R}$-complete, depending on whether products are allowed in the language. This has very recently been extended to additionally allow for variable summation (marginalization) operations, with a corresponding jump in complexity [183]. Such a language can then be used to, for example, check the validity of a do-calculus formula for a given causal diagram.

These works are mostly orthogonal to this thesis as we are mostly concerned with reasoning on statements about a specific causal model, rather than statements about the space of all causal models. Identification algorithms output an expression in terms of the observational distribution, but do not specify how to compute it given a particular SCM/observational distribution. Similarly, even though checking satisfiability of causal (or probabilistic) expressions not involving multiplication is "only" NP-complete, this does not imply that probabilistic (or causal) inference over a given SCM is tractable [56]. Further, rather than the complexity of reasoning on arbitrary SCMs/representations of distributions, we are concerned with tractable representations where reasoning is possible in polynomial time.

3.2.2 Causal Structure Learning

While in causal inference (part of) the causal graph is assumed to be known, in causal structure learning (also known as causal discovery) one seeks to uncover the causal relations between variables. To perform causal structure learning from data, it is necessary to make assumptions on how the structure of the causal model manifests in the observational distribution. In this respect, the three most common assumptions are the *causal Markov condition* (CMC), the *causal faithfulness assumption*, and the *causal sufficiency assumption* [173]. Informally, the causal Markov condition states that every variable is independent of its non-descendants given its parents, the faithfulness assumption states that any conditional independences in the observed distribution is implied by the CMC on the underlying graph, and the sufficiency assumption states that there are no unobserved confounders.

Even under these assumptions, it is usually not possible to uniquely identify the underlying graph even with infinite data, as multiple graphs can induce the same observational distribution; the resulting equivalence classes of indistinguishable DAGs are known as Markov equivalence classes (MEC) [187]. In order to distinguish between DAGs in an equivalence class, one can make assumptions on the functional form of the structural causal model; in particular, for linear non-Gaussian models [167], non-linear additive noise models [76], and post-nonlinear models [206, 207], it becomes possible to identify the causal direction between two variables. This can be used to orient the edges within a MEC.

The primary computational challenge in causal structure learning relates to searching over the combinatorial space of directed acyclic graphs, which grows super-exponentially with the number of variables. Causal structure learning algorithms can broadly be categorized into two types: *constraint-based* and *score-based*. Constraint-based algorithms employ conditional independence tests on the observed data in order to narrow down the space of possible graphs. Prominent examples include the PC algorithm [173], which returns a Markov equivalence class (MEC) of DAGs that imply the same conditional independences, and the FCI [173] and RFCI [35] algorithms, which can additionally deal with violations to causal sufficiency. Score-based algorithms aim to globally optimize a score function defined over the space of graphs, that measures the degree of fit to the data (along with penalties for model complexity). Unfortunately, this optimization is in general `NP`-hard [30], even for bounded-treewidth graphs [94]. In practice, approximate optimization methods need to be used: in particular, greedy equivalence search (GES) [29] is a greedy algorithm that nonetheless retains consistency guarantees in the large-sample limit. More recently, continuous optimization methods based on a relaxation to the acyclicity requirement have been proposed that do not come with guarantees, but demonstrate impressive empirical performance [212, 199, 213, 17, 12].

Bayesian methods for causal structure learning [74] are closely related to score-based approaches, but explicitly define a probability distribution over DAGs via a prior over graphs, and a likelihood for graphs given data. The posterior then represents uncertainty about the underlying causal graph. Unfortunately, exact Bayesian inference methods for structure learning can only scale to around 20 variables [92, 91]. As a result, there has been much interest in approximate methods, which can be divided into sampling-based and variational approaches. Sampling-based approaches utilize MCMC sampling over the space of DAGs; an early proposal known as structure MCMC used edge additions and deletions as Metropolis-Hastings proposals [113, 67]. Unfortunately, the posterior space of structures is highly multimodal and irregular, leading to slow mixing times. This has motivated the development of more sophisticated MCMC schemes for structure learning. Notable works in this direction include OrderMCMC [62], which operates over the much smaller space and smoother posterior landscape of topological orders, [71] which introduced a new edge-reversal proposal, and PartitionMCMC [96], which operates over the space of node partitions.

Alternatively, some recent works have applied variational inference to the Bayesian structure learning problem. In variational inference, one defines a tractable *variational family* of distributions, such as multivariate Gaussians, and then optimizes within this family to find the closest tractable approximation to the true posterior, via a quantity known as the evidence-lower bound (ELBO). Unfortunately, employing variational inference in high-dimensional, discrete spaces such as the space of directed acyclic graphs is highly challenging, for two reasons. Firstly, it is very difficult to specify a variational family of distributions over such spaces (due to the acyclicity constraint) that is sufficiently tractable to use for sampling, density estimation, or other downstream tasks. Secondly, when optimizing the ELBO, pathwise gradient estimators (e.g. reparameterization trick) [88, 120] are not applicable in discrete settings, while the score-function estimator can suffer from prohibitively high variance. In [110], the authors propose a latent variable model and particle-based variational inference, obtaining a sample of graphs. In [3], the authors propose using a neural autoregressive model for the entries of the graph adjacency matrix. Finally, in [40] it is proposed to use a continuous relaxation to the (intractable) Boltzmann distribution over variable orderings, which enables the application of pathwise gradient estimators.

Aside from discovering causal relations in a system, the other key motivation for causal structure learning is to use the learned causal graph to answer downstream causal inference queries. However, it has long been recognized that committing to a single graph can lead to unreliable results, particularly when the graph is not identifiable[4] from observational data [174]. In such cases, there are a couple of approaches to account for the uncertainty [172], usually restricted to the case of linear Gaussian SCMs. One approach is to consider sets of DAGs that can be identified from data, i.e. Markov equivalence classes, and to obtain bounds on the quantity of interest. Notably, the IDA algorithm [112] uses linear regression to estimate causal effects for each DAG in a given equivalence class, returning a (multi)set of possible values for the causal effect. However, this does not account for the statistical uncertainty *between* equivalence classes when there is limited data. Another approach is to use Bayesian structure learning approaches to represent the (statistical and causal) uncertainty over graphs, and then employ Bayesian model averaging [75, 61] to estimate the causal query in expectation over the posterior. This has been implemented in BIDA [140] and OB-MA [22], which average the IDA estimate for each graph in the posterior, while Beeps [188] explicitly computes the exact causal effect for each graph in the approximate posterior; these Bayesian methods have been shown to outperform IDA in accuracy in low sample-size settings. Finally, in non-linear settings, two recent works [178, 180] have proposed end-to-end procedures for causal structure learning and inference utilizing the Bayesian structure learner DiBS [110], though the resulting procedure is highly computationally expensive.

[4]Note that, in causal structure learning, identifiability is typically used to mean identifiability of the causal graph, rather than identifiability of a specific causal inference query given the graph as is the case in causal inference.

Chapter 4

Advanced Causal Reasoning via Compilation

Contents

In this Chapter, we consider reasoning about interventional ($\mathcal{L}_2$) and counterfactual ($\mathcal{L}_3$) queries in the case where the causal graph (DAG) of the underlying system is known. Such situations arise, for example, when expert domain knowledge permits the specification of a structural causal model or causal Bayesian network, or through discovery of causal relations when observational and/or experimental data is available. Given knowledge of the causal graph, we can represent the causal model computationally as a Bayesian network; that is, a collection of conditional probability distributions (parameters) for each variable, conditional on its parents in the graph.

Crucially, we can perform causal interventions in constant time on this representation by modifying the parameters. This means that any interventional query that can be expressed as an associational ($\mathcal{L}_1$) query on an intervened model is no harder than the associational query itself. For example, using classical inference algorithms, both associational and interventional marginal probabilities are exponential in the treewidth of the graph.

Given this, the use of *compiled* representations of causal graphs is particularly appealing. Compilation approaches to inference in Bayesian networks [42] convert a Bayesian network into a more tractable circuit representation, in which marginal and MAP inference queries can be computed efficiently in linear time in the size of the circuit. One of the key features of compilation-based inference is that the compilation process is symbolic with respect to the variables, such that the circuit efficiently "stores" the result of all possible inference queries. In other words, the cost of multiple inference queries (e.g. marginal queries over different sets of variables) can be amortized by compiling once, and then querying the arithmetic circuit for each query. More pertinently for causal reasoning, compiled circuits are also symbolic with respect to the *parameters* of the Bayesian network. This allows us also to amortize the cost of inference across different interventions. As such, we can interpret a compiled circuit not just as a tractable representation of a specific distribution, but a tractable representation of the intervened distributions induced by all possible interventions.

However, such circuits are not fully tractable representations for causal queries, as the space of causal queries extends far beyond those which can be formulated as an associational query on an intervened model. We focus in particular on two such classes of queries of both theoretical and practical importance. The first involves maximization of marginal probabilities over a set of different interventions, which is closely related to the problem of credal inference in the imprecise probability literature. This can be applied, for example to model robustness to distribution shift induced by interventions, or to make decisions about which intervention to apply; recent work has also shown the applicability of credal inference to bounding non-identifiable queries [200]. The second is that of computing counterfactuals ($\mathcal{L}_3$), which involves reasoning over multiple worlds corresponding to different interventions and is arguably a hallmark of human reasoning. Unfortunately, previous work has found that both of these classes of queries have previously been found to result in significantly increased complexity compared to marginal inference on Bayesian networks [39, 49, 73], both in theory and practice.

To tackle these problems, we re-examine the structure of compiled circuits in order to identify conditions under which the queries can be efficiently computed or soundly approximated. Our key finding is that by applying partial ordering constraints corresponding to the topology of the Bayesian network graph, it becomes possible to compute/soundly approximate the queries in linear time in the size of the circuit. Importantly, and analogously to associational inference queries, the compiled circuit "stores" the results for all possible queries in the two classes, including any configuration of interventions. The drawback is that compiling the circuit is now exponential in the topology-constrained treewidth of the Bayesian network. Our results show that using elimination orders corresponding to the topology of a causal graph has practical implications for tractability of (causal) inference, and that circuits compiled with such orderings can answer queries at all levels of the causal hierarchy. Empirically, we evaluate our algorithms on a range of benchmark Bayesian networks and find significantly improved scalability compared to existing approaches.

4.1 Compiled Representations of Models

In this section, we introduce compiled circuit representations of probabilistic and causal graphical models. In this context, we use the term "compiled" in the sense of *knowledge compilation* [48], which is an approach to reasoning over (propositional) logical theories that seeks to transform (compile) a theory from a less tractable language into a more tractable target language, for which reasoning about queries of interest is easy. The motivation behind such an approach is to invest computational resources once into the offline compilation, and then to use the compiled theory to answer many online queries efficiently. The seminal work of [42] applied the framework of knowledge compilation to probabilistic inference in Bayesian networks. Here, instead of logical theories, the object of interest is the probability distribution represented by the Bayesian network, and the target representation is a tractable class of arithmetic circuits, where answering marginal inference queries can be done in linear time. We begin this section by describing compiled representations of Bayesian networks, and then explain how the approach can be extended to causal Bayesian networks and structural causal models.

For the rest of this chapter, we will make the assumption that the cardinality of all variables (endogenous and exogenous) is finite and fixed. This is necessary in order to apply compilation methods, which rely on a direct parameterization of the conditional probability distributions. It has been shown that any SCM over discrete endogenous variables and continuous exogenous variables can be translated to an equivalent SCM where the exogenous variables are also discrete [8, 205] (though the cardinality of these variables may be large). Thus, the main assumption being made is that the observed (endogenous) variables are discrete. We will discuss at the end of this chapter how our methods might be extended also to continuous or mixed variables.

4.1.1 Compiling Bayesian Networks

The core tradeoff in knowledge compilation is between the succinctness of a target representation [69], and the queries that can be computed on them in polynomial time in the size of the representation. Recall that the distribution of a Bayesian network $\mathcal{N} = (G, Pr)$ is given by:

$$p_{\mathcal{N}}(\boldsymbol{V}) = \prod_{i=1}^{d} Pr_i(V_i|\mathrm{pa}_G(V_i)) \tag{4.1}$$

Unfortunately, this product representation is not tractable for marginal inference queries. For example, if we interpret (4.1) as a PC product node with d children, the product is not decomposable as variables are shared between the children. However, there is a representation of the distribution, known as the network polynomial, which *is* tractable for marginal inference. For every value v_i of a variable $V_i \in \boldsymbol{V}$, we define an *indicator variable* $\mathbb{1}_{V_i=v_i}$, which takes the value 1 when $V_1 = v_i$, and 0 otherwise. Further, for every value $\boldsymbol{pa}_i$ of its parents $\mathrm{pa}_G(V_i)$, we define a *parameter variable* $\theta_{v_i|\boldsymbol{pa}_i} := Pr_i(V_i|\mathrm{pa}_G(V_i))$. Then the probability distribution of the Bayesian network can be rewritten in terms of these variables:

$$p_{\mathcal{N}}(\boldsymbol{V}) = \prod_{i=1}^{d} Pr_i(V_i|\mathrm{pa}_G(V_i)) \tag{4.2}$$

$$= \sum_{\boldsymbol{v}\in\mathrm{val}(\boldsymbol{V})} \left(\prod_{i=1}^{d} \mathbb{1}_{V_i=v_i}\right)\left(\prod_{i=1}^{d} Pr_i(v_i|\boldsymbol{pa}_i)\right) \tag{4.3}$$

$$= \sum_{\boldsymbol{v}\in\mathrm{val}(\boldsymbol{V})} \prod_{i=1}^{d} \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{pa}_i} \tag{4.4}$$

Definition 4.1 (Network Polynomial [42, 43])**.** *The network polynomial of a Bayesian network $\mathcal{N} = (G, Pr)$ is defined as:*

$$f_{\mathcal{N}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) := \sum_{\boldsymbol{v}\in val(\boldsymbol{V})} \prod_{i=1}^{d} \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{pa}_i} \tag{4.5}$$

where $\boldsymbol{\lambda}, \boldsymbol{\Theta}$ are the sets of all indicator and parameter variables respectively. Each component $\prod_{i=1}^{d} \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{pa}_i}$ of the summation is called a term, and is associated with a complete instantiation $\boldsymbol{v}$ of the variables $\boldsymbol{V}$.

We can interpret the network polynomial as a *shallow* and *wide* probabilistic circuit over variables $\boldsymbol{V}$, consisting of a root sum node with a product node child for every instantiation of $\boldsymbol{V}$, and leaf nodes given by indicators or parameters. Here, an indicator leaf L with $g_L(V_i) = \mathbb{1}_{V_i=v_i}$ has scope $\phi(L) = V_i$ (for some $V_i \in \boldsymbol{V}$), while parameter leaves L with $g_L() = \theta_{v_i|\boldsymbol{pa}_i}$ have empty scope. This circuit can easily be seen to be decomposable (children of a product each mention one of the variables), smooth and deterministic (children of the sum correspond to different instantiations $\boldsymbol{v}$). This implies that we can compute marginals and MAP in linear time in the size of the network polynomial. The problem, however, is that the network polynomial representation of any Bayesian network is itself exponential in the number of variables $|\boldsymbol{V}|$. In other words, the network polynomial is tractable, but not succinct. The goal of compilation, then, is to find a probabilistic circuit $\mathcal{C}$ over variables $\boldsymbol{V}$ (usually known in this context as an *arithmetic circuit* [42]) that more succinctly represents this polynomial (more specifically, multilinear) function of the indicators and parameters, while also satisfying the required properties for tractability. We say such a circuit *symbolically computes the Bayesian network.*

To formalize this, we define the concept of a complete sub-circuit [23, 31, 33]:

Definition 4.2 (Complete Subcircuit)**.** *A complete subcircuit $\mathcal{C}_{sub} = (G_{sub}, \boldsymbol{\omega}_{sub}, \boldsymbol{g}_{sub})$ of a PC $\mathcal{C}$ is a circuit constructed by recursively traversing $\mathcal{C}$ starting from the root R, choosing one child of every visited sum node T, and choosing all children of every visited product node P. G_{sub} is the subgraph of G consisting of all visited nodes and edges, $\boldsymbol{\omega}_{sub}$ the set of all weights of visited edges, and $\boldsymbol{g}_{sub}$ the set of functions for each visited leaf node. The term $M_{\mathcal{C}_{sub}}$ of a complete subcircuit is defined to be:*

$$M_{\mathcal{C}_{sub}} = \prod_{\omega \in \boldsymbol{\omega}_{sub}} \omega \prod_{g_L \in \boldsymbol{g}_{sub}} g_L^{n_L} \tag{4.6}$$

where n_L is the number of times L is reachable from the root in $\mathcal{C}_{sub}$ [1]*.*

The distribution $p_{\mathcal{C}}(\boldsymbol{V})$ represented by the probabilistic circuit is given by the sum over the terms of all complete subcircuits, when the leaf node functions are evaluated at $\boldsymbol{V}$. This gives rise to the polynomial interpretation of PCs, where the sum over terms is a polynomial where the indeterminates (variables) are the leaf node functions $\boldsymbol{g}$ (and possibly the weights $\boldsymbol{\omega}$). In our case, we will instead define a polynomial over variables $\boldsymbol{\lambda}, \boldsymbol{\Theta}$, by assuming that the weights ω and leaf node distributions g are themselves given by indicators or parameters.

Definition 4.3 (PC polynomial)**.** *The polynomial of a probabilistic circuit $\mathcal{C}$ is defined to be:*

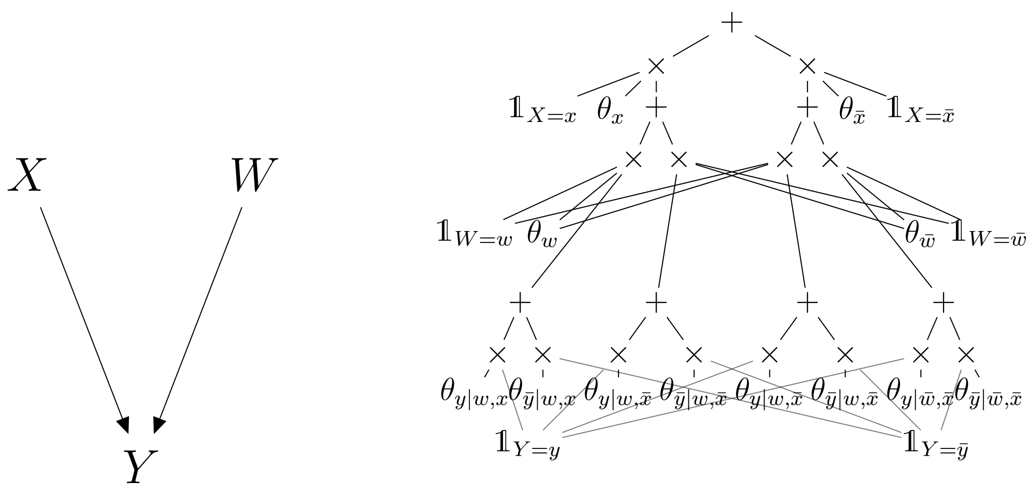
$$f_{\mathcal{C}}(\boldsymbol{\omega}, \boldsymbol{g}) = \sum_{\mathcal{C}_{sub} \in \mathcal{C}} M_{\mathcal{C}_{sub}}(\boldsymbol{\omega}, \boldsymbol{g}) \tag{4.7}$$

If every weight $\omega \in \boldsymbol{\omega}$ is either equal to 1 or a parameter θ, and each leaf node function $g \in \boldsymbol{g}$ is either an indicator λ or a parameter θ, then we can equivalently write this as a polynomial over $\boldsymbol{\lambda}, \boldsymbol{\Theta}$:

$$f_{\mathcal{C}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) = \sum_{\mathcal{C}_{sub} \in \mathcal{C}} M_{\mathcal{C}_{sub}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) \tag{4.8}$$

For compiled PCs (arithmetic circuits), the parameter variables are typically all represented in the leaf node functions, and all sum node weights are equal to 1. However, the ability to incorporate parameters into $\boldsymbol{\omega}$ will be important later on. We will write $\boldsymbol{\omega}(\boldsymbol{\Theta})$ and $\boldsymbol{g}(\boldsymbol{\lambda}, \boldsymbol{\Theta})$ to highlight the dependence of the sum node weights and leaf functions on the indicators/parameters.

[1]In this Chapter, and in general for decomposable circuits, n_L will be equal to 1.


X
W
Y
$+$
$\times$
$\mathbb{1}_{X=x}$ θ_x
$\theta_{\bar{x}}$ $\mathbb{1}_{X=\bar{x}}$
$\mathbb{1}_{W=w}$ θ_w
$\theta_{\bar{w}}$ $\mathbb{1}_{W=\bar{w}}$
$\theta_{y|w,x}$ $\theta_{\bar{y}|w,x}$ $\theta_{y|w,\bar{x}}$ $\theta_{\bar{y}|w,\bar{x}}$ $\theta_{y|w,\bar{x}}$ $\theta_{\bar{y}|w,\bar{x}}$ $\theta_{y|\bar{w},\bar{x}}$ $\theta_{\bar{y}|\bar{w},\bar{x}}$
$\mathbb{1}_{Y=y}$
$\mathbb{1}_{Y=\bar{y}}$

(a) Bayesian Network (b) Compiled Circuit

Figure 4.1: Example of Bayesian network and corresponding compiled circuit. All sum node weights are equal to 1 in the circuit.

Definition 4.4 (PC symbolically computing Bayesian Network)**.** *A probabilistic circuit* $\mathcal{C} = (G, \boldsymbol{\omega}, \boldsymbol{g})$ *symbolically computes a Bayesian network* $\mathcal{N}$ *if*

$$f_{\mathcal{C}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) = f_{\mathcal{N}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) \tag{4.9}$$

This definition says that the circuit must explicitly encode the same polynomial over indicators and parameters; that is, the term of each subcircuit $M_{\mathcal{C}_{sub}}$ corresponds to a term $\prod_{i=1}^{d} \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{pa}_i}$ of the BN polynomial, and thus each subcircuit corresponds to a specific instantiation $\boldsymbol{v}$ of the domain variables $\boldsymbol{V}$. The importance of symbolic computation is that the distribution over $\boldsymbol{V}$ encoded by the circuit is the same as that of the Bayesian network for *any* values of the CPDs Pr.

The ability to represent the network polynomial more compactly rests on two principles. Firstly, by taking advantage of the topology of the Bayesian network graph, it is possible to exchange the order of sums and products. This is the basis for exact inference methods for probabilistic inference in graphical models such as variable elimination (VE) and jointree inference, whose efficiency depends on the *treewidth* of the Bayesian network graph. Secondly, aside from the graph, one can exploit knowledge of the *values* of the parameters of Pr, e.g. when $Pr_i(v_i|\boldsymbol{pa}_i) = 0$ (determinism) or $Pr_i(v_i|\boldsymbol{pa}_i) = Pr_i(v_i|\boldsymbol{pa}'_i)$ (context-specific independence) for some parameters [24, 25, 26]. However, the disadvantage is that the resulting circuits are no longer valid for all values of the parameters, i.e. the circuit does not symbolically compute the BN. This is particularly problematic for causal inference, as we will shortly see. For this reason, we restrict to compilation methods that do not (or are configured not to) make such assumptions.

We will now define a new property of probabilistic circuits, known as the *decision property.* This property intuitively states that every sum node has product node children corresponding to each of the values of a particular variable V.

Definition 4.5 (Decision Property)**.** *A sum node T in a probabilistic circuit decides on a variable V if it has $|val(V)|$ product node children $\{P_v\}_{v \in val(V)}$, such that P_v has a child $\mathbb{1}_{V=v}$. We say that a circuit satisfies the decision property if all of its sum nodes decide on a variable, and write $\psi(T)$ for the variable T decides on.*

For example, if V was binary, we could have $(\mathbb{1}_{V=0} \times N_0) + (\mathbb{1}_{V=1} \times N_1)$, where N_0, N_1 are PC nodes. This property is also known as *representing a variable* [156], and can be viewed as the functional/probabilistic analogue of the concept of decision-DNNF circuits in propositional logic [48, 78], but replacing conjunctions with $\times$, and disjunctions with $+$ [79]. It is clear from the definition that the decision property implies determinism, as it explicitly specifies conflicting indicators for each child; however, not all deterministic circuits satisfy the decision property. We might wonder, then, whether enforcing the stronger decision property allows for tractability of more queries. It has been shown that enforcing certain constraints on the relative *ordering* of decision nodes in a circuit allows for linear-time computation of marginal MAP, or, more generally, functional E-MAJSAT with respect to a set of variables $\boldsymbol{W} \subseteq \boldsymbol{V}$ [128, 77, 143]. In particular, a Decision-PC is said to be $\boldsymbol{W}$*-constrained* if all sum nodes which decide on variables in $\boldsymbol{W}$ appear above variables deciding on $\boldsymbol{V} \setminus \boldsymbol{W}$.

More generally, we can speak of a circuit satisfying a *strict partial order.* A strict partial order $<$ on a set $\boldsymbol{V}$ is a relation between elements of $\boldsymbol{V}$, which satisfies irreflexivity and transitivity. Importantly, not all elements of $\boldsymbol{V}$ need to be comparable. We say that an PC satisfies a given partial order $<$ if the topology of the PC is consistent with the partial order:

Definition 4.6 (PC Satisfying Partial Order)**.** *A probabilistic circuit $\mathcal{C}$ satisfies a partial order $<$ over $\boldsymbol{V}$, if it satisfies the decision property, and for any pair of sum nodes T, T' in $\mathcal{C}$,*

$$\psi(T) < \psi(T') \implies T' \textit{ is not a descendant of } T \tag{4.10}$$

Unfortunately, enforcing a partial order can increase the complexity of compilation of circuits. Compilation methods based on variable elimination take as input an *elimination order* σ, which is a total ordering over the variables. The resulting compiled circuit satisfies the decision property and the total order σ', where σ' is the *reversal* of σ. Thus, to enforce a partial order $<$, one needs to choose an elimination ordering whose reversal is consistent with $<$. The treewidth of a Bayesian network is given by the minimum width among all total elimination orders. Clearly, the minimum width among all elimination orders (whose reversal is) consistent with a given partial order (called the *constrained treewidth* by [127, 128] in the case of a $\boldsymbol{W}$-constrained partial order) is at least as large, and can be considerably larger depending on the partial order. In practice, we cannot compute either the treewidth or $<$-constrained treewidth efficiently (the former is NP-hard). As a result, we must rely on heuristics to choose an elimination ordering. These heuristics typically choose the variables in the ordering one-by-one, at each point computing a heuristic value for each remaining variable and choosing the variable with the largest value. Given any partial order $<$, we can find a $<$-constrained total order by choosing at each point the variable with the largest heuristic value which would not violate $<$. In our experiments, we use the empirically successful *min-fill* heuristic.

As an example of these definitions, in Figure 4.1 we show a simple Bayesian network over three binary variables, and a decomposable, smooth, and deterministic circuit over $\{X, W, Y\}$ which computes the Bayesian network distribution. Note that we have omitted weights from the edges of the sum nodes, by which we mean that all such weights are equal to 1. It can be checked (e.g. by explicitly enumerating all subcircuits) that the circuit symbolically computes the Bayesian network. Further, the circuit satisfies the decision property, as the root sum node decides on X, the two subsequent sum nodes on W, and the bottom four sum nodes on Y.

4.1.2 Compiled Representations for Causality

In a similar vein to probabilistic inference queries on Bayesian networks, our aim is to identify tractable target representations for various causal ($\mathcal{L}_2$ and $\mathcal{L}_3$) queries on structural causal models and causal Bayesian networks. We begin by noting that both causal Bayesian networks and structural causal models can be compiled in a similar manner to Bayesian networks. For causal Bayesian networks $\mathcal{CBN}$, which share syntax with Bayesian networks, the compilation is entirely similar. For structural causal models $\mathcal{M} = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, \mathrm{Pr})$, we can instead compile to a circuit over both the endogenous and exogenous variables $\boldsymbol{U} \cup \boldsymbol{V}$, as follows. For every value u of a exogenous variable U, we define an indicator variable $\mathbb{1}_{U=u}$, and a parameter variable $\theta_u := \mathrm{Pr}_U(u)$. For every value v_i of an endogenous variable V_i, we define an indicator variable $\mathbb{1}_{V_i=v_i}$. Finally, for every value $\boldsymbol{u}_i$ of its exogenous parents and $\boldsymbol{pa}_i$ of its endogenous parents, we define a parameter variable $\theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i} := \mathbb{1}_{v_i=F_i(\boldsymbol{u}_i,\boldsymbol{pa}_i)}$. The resulting "SCM polynomial" can be compiled exactly as if it were a Bayesian network over $\boldsymbol{U} \cup \boldsymbol{V}$.

Definition 4.7 (SCM polynomial)**.** *Given an SCM $\mathcal{M}$, the SCM polynomial is defined as:*

$$f_{\mathcal{M}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) := \sum_{\boldsymbol{u} \in val(\boldsymbol{U}), \boldsymbol{v} \in val(\boldsymbol{V})} \prod_{U \in \boldsymbol{U}} \mathbb{1}_{U=u} Pr_U(u) \prod_{i=1}^{d} \mathbb{1}_{V_i=v_i} \theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i} \tag{4.11}$$

where $\boldsymbol{\lambda}, \boldsymbol{\Theta}$ are the sets of all indicator and parameter variables respectively. Each component $\prod_{U \in \boldsymbol{U}} \mathbb{1}_{U=u} Pr_U(u) \prod_{i=1}^{d} \mathbb{1}_{V_i=v_i} \theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i}$ of the summation is called a term, and is associated with a complete instantiation $\boldsymbol{u}, \boldsymbol{v}$ of the variables $\boldsymbol{U}, \boldsymbol{V}$.

We now consider computing interventional $\mathcal{L}_2$ and counterfactual $\mathcal{L}_3$ queries. We first identify a case in which interventional reasoning is no harder than associational reasoning: namely, when the interventional query can be expressed as an associational query on an intervened model. For example, for causal Bayesian networks, the interventional distribution for intervention Pr' is given by:

$$p_{\mathcal{CBN}}(\boldsymbol{V}_{Pr'}) = \prod_{i=1}^{d} Pr'_i(V_i|\mathrm{pa}_G(V_i)) \tag{4.12}$$

$$= \sum_{\boldsymbol{v} \in \mathrm{val}(\boldsymbol{V})} \prod_{i=1}^{d} \mathbb{1}_{V_i = v_i} \theta'_{v_i|\boldsymbol{pa}_i} \tag{4.13}$$

where $\theta'_{v_i|\boldsymbol{pa}_i} = Pr'_i(V_i|\mathrm{pa}_G(V_i))$. Given a PC $\mathcal{C} = (G, \boldsymbol{\omega}(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}, \boldsymbol{\Theta}))$ which symbolically computes the Bayesian network, we can obtain a new PC $\mathcal{C} = (G, \boldsymbol{\omega}(\boldsymbol{\Theta}'), \boldsymbol{g}(\boldsymbol{\lambda}, \boldsymbol{\Theta}'))$ where $\boldsymbol{\Theta}'$ has been modified by changing $\theta_{v_i|\boldsymbol{pa}_i}$ to $\theta'_{v_i|\boldsymbol{pa}_i}$ for any leaf corresponding to a parameter variable. As a result, using a compiled circuit it is possible to compute any standard probabilistic inference query (e.g. marginals, MPE) which is tractable on a decomposable, smooth, and deterministic circuit on any interventional distribution.

However, not all causal queries can be expressed as an associational query on an intervened model; in particular, queries that involve reasoning over multiple different interventions simultaneously do not fall under this category. In the rest of this Chapter, we focus on two such classes of queries. Firstly, for interventional $\mathcal{L}_2$ reasoning, we may be interested in the probability of some event under a set of different interventions, that could represent different environments, or different actions. A natural query in such a setting is to compute the maximum or minimum probability of the event over the set of interventions. As we will see, such queries have important applications for analyzing robustness and for decision-making in causal systems. Secondly, we consider reasoning over counterfactual $\mathcal{L}_3$ distributions for SCMs. Whereas compiled circuits allow us to tractably reason over interventional distributions, it is unclear how we can even evaluate a counterfactual distribution $p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{V}_{\boldsymbol{F}^{(n)}})$, which involves transferring information across counterfactual worlds induced by different interventions. In both cases, we will derive tractability conditions on compiled circuits and algorithms that allow us to compute or soundly approximate the query.

4.2 Interventional Robustness

In this section, we consider the query of maximizing the probability of an event $\boldsymbol{E} = \boldsymbol{e}$, where $\boldsymbol{E} \subseteq \boldsymbol{V}$, over a specified set of interventions, which we call the *interventional* *robustness* query. We begin by discussing how to specify sets of interventions using *credal sets*, and then analyze the complexity of the problem. This motivates the need for a scalable approximate procedure that can produce guaranteed bounds on the query. We then describe a condition on circuits compiled from CBNs, which allows us to efficiently upper bound *any* interventional robustness query. Finally, we show how to tighten the upper bounds and additionally obtain lower bounds.

4.2.1 Interventional Robustness and Credal Sets

We first consider how to specify a set of interventions on a causal Bayesian network. For this, we can use the concept of credal sets in the field of imprecise probability, where one assigns a (closed and convex) set of probability measures representing imprecise knowledge. In the case of Bayesian networks, this is done by specifying credal sets $K_i(V_i|\boldsymbol{PA}_i)$ on each individual CPD in the network. For example, for a binary variable V_i with one parent PA_i, one way to construct a credal set is to specify intervals on each individual probability as follows:

$$K_i(V_i|PA_i = 0) = \{Pr_i : Pr_i(V_i = 1|PA_i = 0) \in [0.2, 0.7]\} \tag{4.14}$$

$$K_i(V_i|PA_i = 1) = \{Pr_i : Pr_i(V_i = 1|PA_i = 1) \in [0.6, 0.8]\} \tag{4.15}$$

If a CPD cannot be intervened on, then we can represent this with a credal set that contains only one element, namely the original CPD. Now, given sets of interventions on each CPD, and their corresponding credal sets $K_i(V_i|\boldsymbol{PA}_i)$, we now consider extending to a credal set for the parameters of the entire causal Bayesian network. Here, we follow the principle of *independent causal mechanisms* (ICM) [141], which informally states that in causal systems, the mechanisms/CPDs for each variable are independent in the sense that they do not inform and influence each other. In other words, intervening on one mechanism should not affect another mechanism, or indeed in our case, affect the set of interventions that are available on the other mechanism. For this reason, we consider credal sets that allow for any combination of interventions on each CPD. In the credal inference literature, this is often known as the *strong extension* of the credal sets [39].

$$K = \{Pr : Pr_i \in K_i(V_i|\boldsymbol{PA}_i)\} \tag{4.16}$$

The pair of a Bayesian network (graph), and a credal set for the CPDs, is referred to as a *credal network*. The problem of maximizing the probability of an event on a credal network is known as the *credal marginals* problem. In the context of causal interventions, we can define the query as follows:

Definition 4.8 (Interventional Robustness Query)**.** *Given a causal Bayesian network $\mathcal{CBN}$, the interventional robustness query is parameterized by a credal set of interventions $K = \{Pr : Pr_i \in K_i(V_i|\boldsymbol{PA}_i)\}$ and an event $\boldsymbol{e}$, and is defined by:*

$$\texttt{INTROB}(\mathcal{CBN}; K, \boldsymbol{e}) = \max_{Pr' \in K} p_{\mathcal{CBN}}(\boldsymbol{e}_{Pr'}) \tag{4.17}$$

4.2.2 Bounding Interventional Robustness via Compilation

Our goal for the rest of this section will be to find a representation of causal Bayesian networks that allows us to efficiently bound any interventional robustness query on that CBN. As a starting point, let us consider a PC $\mathcal{C}$ compiled from the CBN. Maximizing over a credal set K then corresponds to maximizing over the leaf node functions corresponding to the parameters, that is:

$$\max_{\boldsymbol{\Theta}' \in K} p_{\mathcal{C}(G, \boldsymbol{\omega}(\boldsymbol{\Theta}'), (\boldsymbol{\lambda}, \boldsymbol{\Theta}'))}(\boldsymbol{e}) \tag{4.18}$$

Unfortunately, maximizing over the values of parameters $\theta_{v_i|\boldsymbol{pa}_i}$ is not straightforward, due to the constraint that $(\theta_{v_i|\boldsymbol{pa}_i})_{v_i \in \text{val}(V_i)} \in K_i(V_i|\boldsymbol{pa}_i)$. In other words, different parameters, which may be spread out in different parts of the circuit, are jointly constrained. This prevents us from performing local optimizations in the circuit in a similar manner to, for example, the computation of the MAP query in deterministic circuits. A trivial way to resolve this would be to replace the simplex constraints with the (much) weaker constraints that $\theta_{v_i|\boldsymbol{pa}_i} \in [0, 1]$ for all parameters. The corresponding optimization problem produces an upper bound on 4.18, and can easily be solved due to the independent constraints on each parameter. Unfortunately, such a bound is clearly uninformative as it violates even the simplex constraint that $\sum_{v_i \in \text{val}(V_i)} \theta_{v_i|\boldsymbol{pa}_i} = 1$.

In order to make progress, we will impose additional constraints on the structure of the PC. In particular, we consider compiled PCs satisfying $<_{\text{top}}$, where $<_{\text{top}}$ is a partial order corresponding to the topology of the causal Bayesian network graph G. That is, $V_i <_{\text{top}} V_j$ iff V_i is an ancestor of V_j in G. Under this condition, we can show the following result, which intuitively states that when a circuit splits on variable V_i, the values of its parents are already "known":

Lemma 4.1 (Known Parents)**.** *Let $\mathcal{C}$ be a compiled PC that symbolically computes the CBN $\mathcal{N}$, and satisfies topological ordering constraints $<_{top}$. Then, for any sum node T deciding on a variable V_i, every complete subcircuit $\mathcal{C}_{sub}$ of $\mathcal{C}$ which contains T must contain the same indicators for the parents $\boldsymbol{PA}_i$.*

For example, in Figure 4.1, we see that the compiled circuit satisfies $<_{\text{top}}$. By the Lemma, we can deduce that, for example, the bottom four sum nodes, which decide on Y, must also correspond to a specific value of (W, X). This can be checked to hold by noting that e.g. the leftmost sum node only appears in the subcircuit with $\mathbb{1}_{X=x}$ and $\mathbb{1}_{W=w}$.

Given this result, and assuming topological ordering constraints, we propose below an algorithm that converts the compiled PC into a new PC where the parameters variables $\boldsymbol{\Theta}$ are all assigned to weights $\boldsymbol{\omega}$ of sum nodes in the PC, instead of leaves. Intuitively, because the parent values $\boldsymbol{pa}_i$ are known at a sum node deciding on a variable V_i, we can directly assign each $\theta_{v_i|\boldsymbol{pa}_i}$ for $v_i \in \text{val}(V_i)$ to the respective children of the sum node. Algorithmically, we can determine the value of the parents (for each sum node) using a single pass through the PC, as follows. We use $\boldsymbol{P}(N, V_i)$ to denote the set of "possible values" of the variable V_i at a PC node N:

$$\boldsymbol{P}(N, V_i) := \{v_i : \exists L \in \text{desc}(N) : g_N = \theta_{v_j|\boldsymbol{pa}_j} \text{ and } v_i \in \boldsymbol{pa}_j\} \tag{4.19}$$

That is, this contains the set of all values of V_i, such that there is a descendant leaf node of N corresponding to a parameter that specifies that value of V_i. The point of this definition is that 1) at a sum node T deciding on V_i, this uniquely tells us the value of the parents of V_i; and 2) this can be easily computed using a bottom up pass.

Proposition 4.1. *Suppose T decides on variable V_i. Then, for any parent $PA_i \in \boldsymbol{PA}_i$ of V_i, $\boldsymbol{P}(T, PA_i)$ is a singleton set.*

Algorithm 4.1: Transpilation of Compiled PC

```
Input: Compiled PC $\mathcal{C} = (G, \boldsymbol{\omega} = 1, \boldsymbol{g}(\boldsymbol{\lambda}, \boldsymbol{\Theta}))$ symbolically computing $\mathcal{CBN}$
       and satisfying topological ordering constraints.
Result: PC $\mathcal{C}' = (G, \boldsymbol{\omega}(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}))$
1 begin
2     For leaf nodes $L$ corresponding to indicators, compute $\boldsymbol{P}(L, V_i) = \{\}$ for all
        $V_i \in \boldsymbol{V}$;
3     For leaf nodes $L$ corresponding to a parameter $\theta_{v_i|\boldsymbol{pa}_i}$, compute
        $\boldsymbol{P}(L, PA_i) = \{pa_i\}$ for $PA_i \in \boldsymbol{PA}_i$ and $\boldsymbol{P}(L, V_j) = \{\}$ for $V_j \in \boldsymbol{V} \setminus \boldsymbol{PA}_i$;
4     For internal nodes $N$, compute $\boldsymbol{P}(N, V_i) = \bigcup_{N_j \in \mathrm{ch}(N)} \boldsymbol{P}(N_j, V_i)$ for all
        $V_i \in \boldsymbol{V}$;
5     For sum nodes $T$ deciding on $V_i$, assign $\omega_{T,j} \leftarrow \theta_{v_i(j)|\boldsymbol{pa}_i}$, where $v_i(j)$ is the
        value of $V_i$ corresponding to the $j^{th}$ child of $T$, and for each $PA_i \in \boldsymbol{PA}_i$,
        $pa_i$ is the unique value in $PA_i$ in $\boldsymbol{P}(T, PA_i)$;
6     Remove all leaf nodes corresponding to parameters.
```

In Algorithm 4.1, we show the overall procedure for transferring the parameters $\boldsymbol{\Theta}$ to the weights of the sum nodes in the PC, which we call *transpilation.* The algorithm executes a bottom-up computation through the PC, where for each node we compute $\boldsymbol{P}$, and at sum nodes, we assign the parameter values to the appropriate weights. Finally, having assigned all weights, we remove all leaf nodes corresponding to parameters from the circuit. The final PC has all of the parameter variables $\boldsymbol{\Theta}$ in the sum node weights, and all of the Crucially, we can show that the transpiled PC represents the same function of the indicators and parameters, meaning that it also symbolically computes $\mathcal{CBN}$.

Proposition 4.2 (Correctness of Transpiled PC)**.** *The output of Algorithm 4.1 symbolically computes $\mathcal{CBN}$.*

4.2.2.1 Upper Bounding via Constraint Relaxation

The motivation behind transpilation is that this creates a direct correspondence between the weights of a sum node $\boldsymbol{\omega}_T$, and the set of causal parameters $(\theta_{v_i|\boldsymbol{pa}_i})_{v_i \in \mathrm{val}(V_i)}$. As such, we can now locally maximize the probability $p_T(\boldsymbol{e})$ of the event $\boldsymbol{e}$, by maximizing over $\boldsymbol{\omega}_T \in K_i(V_i|\boldsymbol{pa}_i)$ given the maximal probability $p_{N_j}(\boldsymbol{e})$ of the event under each of the children N_j of T.

Algorithm 4.2: Upper Bounding

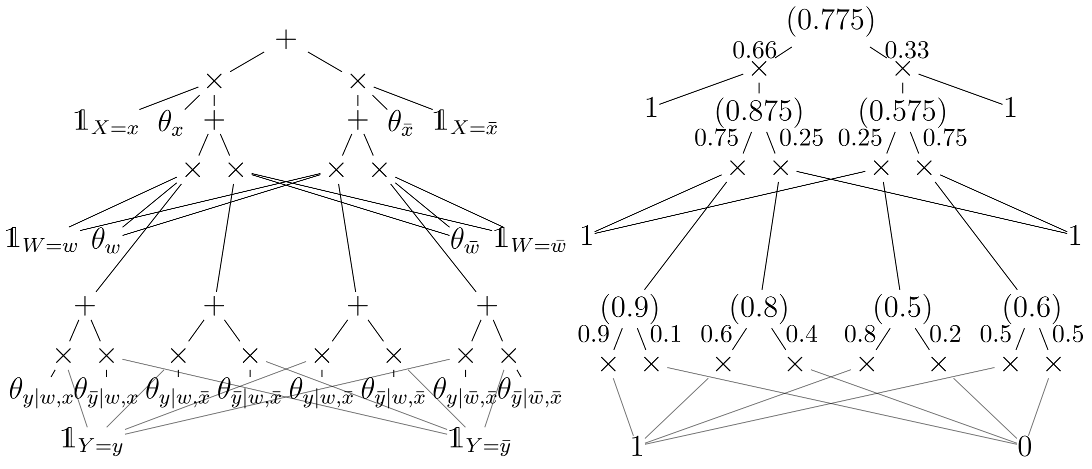
```
Input: Transpiled PC $\mathcal{C} = (G, \boldsymbol{\omega}(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}))$ symbolically computing $\mathcal{CBN}$,
       credal set $K$, event $\boldsymbol{e}$
Result: Upper bound on interventional robustness query
1 begin
2     For leaf nodes $L$ corresponding to an indicator $\lambda = \mathbb{1}_{V_i=v_i}$, compute
        $p_{upper}(L) = \mathbb{1}_{v_i \models \boldsymbol{e}}$ if $V_i \in \boldsymbol{E}$, and $p_{upper}(L) = 1$ otherwise;
3     For product nodes $P$, compute $p_{upper}(P) = \prod_{N_j \in \mathrm{ch}(P)} p_{upper}(N_j)$;
4     For sum nodes $T$ deciding on $V_i$ with parent values $\boldsymbol{pa}_i$, compute
        $\max_{\boldsymbol{\omega}_T \in K_i(V_i|\boldsymbol{pa}_i)} \sum_{N_j \in \mathrm{ch}(T)} \omega_{T,j} p_{upper}(N_j)$;
5 Return $p_{upper}(R)$ where $R$ is the root of $\mathcal{C}$.
```

The complication, however, is that the set of parameters $(\theta_{v_i|\boldsymbol{pa}_i})_{v_i \in \mathrm{val}(V_i)}$ may appear for multiple different sum nodes T in the transpiled circuit, which all split on V_i and have parents value $\boldsymbol{pa}_i$. That is, aside from the constraint $\boldsymbol{\omega}_T \in K_i(V_i|\boldsymbol{pa}_i)$, we also have a second constraint that $\boldsymbol{\omega}_T = \boldsymbol{\omega}_{T'}$ for any two such sum nodes. This prevents us from optimizing the weights in the transpiled PC for these sum nodes independently.

We thus propose to upper bound the interventional robustness query by removing this second equality constraint, so that the constrained optimization over $\boldsymbol{\omega} \in K$ becomes a local constrained optimization $\boldsymbol{\omega}_T \in K_i(V_i|\boldsymbol{pa}_i)$ for each sum node. For this relaxed problem, we can compute the maximum probability of any event using the algorithm from [116], which we adapt in our context in Algorithm 4.2. In Figure 4.2, we show an example execution of the algorithm for the event $Y = 1$, and for a credal set K the details of which we omit for brevity. Notice in particular that the two sum nodes deciding on W have different optimized weights, even though they correspond to the same parameters $\theta_w, \theta_{\bar{w}}$. For this reason, the output 0.775 is only an upper bound on the interventional robustness query.

4.2.2.2 Lower Bounding via Projection

We now consider deriving lower bounds on the interventional robustness query. By definition, any assignment $\boldsymbol{\Theta}'$ to the parameters $\boldsymbol{\Theta} \in K$ provides a lower bound, given by $p_{\mathcal{C}(G,\boldsymbol{\omega}(\boldsymbol{\Theta}'),\boldsymbol{g}(\boldsymbol{\lambda}))}(\boldsymbol{e})$. We can thus consider searching over the space of parameters in order to find a tight lower bound. To aid in the search process, we propose to initialize



(a) Compiled PC (b) Upper bounding on transpiled PC

Figure 4.2: Example of upper bounding routine, for the event $Y = 1$

the search using *projection* from the obtained upper bound. In particular, we assign each set of parameters $(\theta_{v_i|\boldsymbol{pa}_i})_{v_i \in \mathrm{val}(v_i)} \leftarrow \boldsymbol{\omega}_T^*$, where $\boldsymbol{\omega}_T^*$ are the optimized parameters for some arbitrary T that corresponds to $V_i, \boldsymbol{pa}_i$. We then perform local changes to the parameters, greedily keeping any that increase the lower bound.

4.2.3 Case Study: Robustness Analysis of Classifiers

In this case study, we look at a causal Bayesian network model which models car insurance risks, shown in Figure 4.3 [13]. Suppose an insurance company wishes to use a classifier to predict `MedCost` (the medical cost of an insurance claim), and has access to an insurant's `Age`, `DrivHist`, and `MakeModel` (categorical variables with 3-5 values). `MedCost` is either *BelowThousand* (0) or *AboveThousand* (1). They fit a Naïve Bayes classifier to historical data, obtaining a decision function F. This is then used as part of their decision-making policy determining what premiums to offer to customers. We can represent this by adding an additional node F to the Bayesian network, which has parents $\{\texttt{Age}, \texttt{DrivHist}, \texttt{MakeModel}\}$.

The company is particularly concerned about false negatives, as this could result in the company losing a lot of money in payouts. Based on the original Bayesian network model (Figure 4.3) and their new classifier, this should occur 2.5% of the time. However, in reality, insurants may attempt to game the classifier to predict *BelowThousand* (so that they get lower premiums), while actually being likely to have

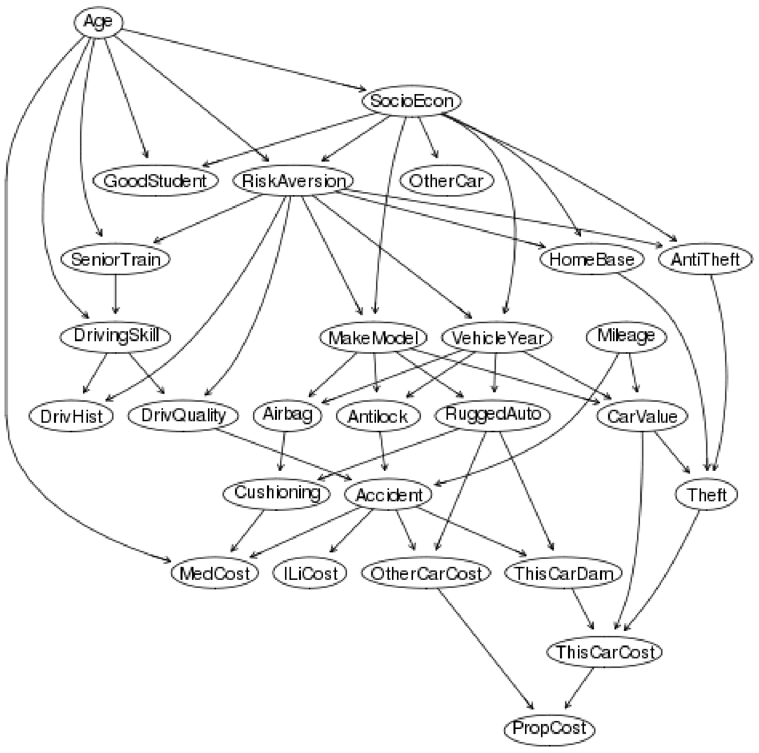


Figure 4.3: INSURANCE Bayesian network [13]

a high medical cost. That is, we assume that insurants can causally intervene on some of `DrivHist` (perhaps hide some accident history), `MakeModel` (drive a different type of car than they would normally choose), and `Cushioning` (upgrade/downgrade the degree of protection inside the car). The company would thus like to understand how *robust* their classifier is to these adaptations. In order to model this scenario, we can assign credal sets $K_{\texttt{DrivHist}}, K_{\texttt{MakeModel}}, K_{\texttt{Cushioning}}$ to each of these mechanisms, while leaving the parameters of the other mechanisms fixed (i.e. trivial credal set). For simplicity (and modelling a situation where we want to be extremely conservative), we define these credal sets to allow any value of the parameters in $[0, 1]$, subject to the simplex constraint that the appropriate parameters sum to 1.

We seek to obtain guaranteed upper bounds on these two quantities:

- **FN**: The probability of a false negative $p(F = 0, \texttt{MedCost} = 1)$, i.e. predicted low medical cost, but high actual medical cost.
- **P**: The probability of a positive $p(\texttt{MedCost} = 1)$, i.e. high actual medical cost.

Intervenable Variables	**FN**	**P**
Empty Set	2.5%	7.2%
{DrivHist}	7.2%	7.2%
{MakeModel}	5.7%	10.0%
{Cushioning}	6.1%	12.9%
{DrivHist, MakeModel}	10.0%	10.0%
{DrivHist, Cushioning}	12.9%	12.9%
{MakeModel, Cushioning}	13.0%	13.9%
{DrivHist, MakeModel, Cushioning}	13.9%	13.9%

Table 4.1: Guaranteed upper bounds on FN and P, under different credal sets

The results are shown in Table 4.1, where in each column, we allow a different subset of {`DrivHist, MakeModel, Cushioning`} to be intervened on. Note that all of these are computed on the *same* transpiled circuit. The insurance company can use these bounds to assess risk, and improve their classifier's robustness if they deem the false negative rate under intervention unacceptable.

The bounds can also provide further insight. We notice that whenever `DrivHist` is intervenable, the percentage of false negatives is the same as positives, i.e. the classifier always predicts wrong when `MedCost` is 1. This turns out to be because the Naïve Bayes classifier always predicts 0 whenever `DrivHist` is *None*, regardless of the other input variables. Thus, an insurant who can change their `DrivHist` can always fool the classifier to predict 0. In addition, the percentage of positives doesn't increase from the original BN: this can be seen from the causal graph, where `DrivHist` has no causal influence on `MedCost`. On the other hand, `Cushioning` significantly increases the positive rate. Notice that, in the graph, intervening on `Cushioning` will not have any influence on the inputs to the classifier; thus, the increase in FN to 6.1% is not due to fooling the classifier, but rather making high medical expenses generally more likely, by downgrading the quality of cushioning. In this way, the intervention is "taking advantage" of the classifier not having full information about cushioning.

4.3 Counterfactual Reasoning using Circuits

We now consider reasoning about $\mathcal{L}_3$ queries on structural causal models. As we have seen in Section 4.1.2, any SCM can be represented as a polynomial function of indicators and parameters, where the indicators correspond to values of the exogenous variables $\boldsymbol{U}$ and endogenous variables $\boldsymbol{V}$, and the parameters correspond to the distributions on the exogenous variables and the causal mechanisms $\boldsymbol{F}$. Recall also that a general counterfactual distribution can be written as:

$$p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{V}_{\boldsymbol{F}^{(n)}}) = \sum_{\boldsymbol{U}} \Pr(\boldsymbol{U}) \prod_{j=1}^{n} \left(\prod_{i=1}^{d} \mathbb{1}_{V_{\boldsymbol{F}^{(j)},i} = F_i^{(j)}(\boldsymbol{U}_i, \boldsymbol{PA}_{\boldsymbol{F}^{(j)},i})} \right) \tag{4.20}$$

where $\boldsymbol{F}^{(1)}, ..., \boldsymbol{F}^{(n)}$ are interventions. The most common type of counterfactual query involves reasoning over two *worlds*: one corresponding to the observed reality, and one corresponding to an imaginary world. These worlds are represented by two interventions; a "null" intervention $\boldsymbol{F}^{(1)} = \boldsymbol{F}$ which represents the real world, and an intervention $\boldsymbol{F}^{(2)}$ which represents the changes to functional mechanisms in the imaginary world. For example, suppose that the variables $\boldsymbol{V}$ represents the observed course of a patient's illness (for example, their pre-existing conditions, treatments administered, etc.), and $Y \in \boldsymbol{V}$ represents the patient's survival. Then, if $\boldsymbol{v}_{\boldsymbol{F}}$ represents events in the observed world, where the patient died ($Y_{\boldsymbol{F}} = 0$), and the intervention $\boldsymbol{F}^{(2)}$ represents a potential alternative treatment regimen, the query $p_{\mathcal{M}}(Y_{\boldsymbol{F}^{(2)}} = 1|\boldsymbol{v}_{\boldsymbol{F}})$ represents the probability that the patient *would have survived* had the alternative regimen been administered.

More generally, for n worlds, counterfactual marginal reasoning simply corresponds to computing a marginal of the counterfactual distribution in Equation 4.20, that is:

$$p_{\mathcal{M}}(\boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}}) = \sum_{\boldsymbol{V}_{\boldsymbol{F}^{(1)}} \backslash \boldsymbol{E}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{V}_{\boldsymbol{F}^{(n)}} \backslash \boldsymbol{E}^{(n)}_{\boldsymbol{F}^{(n)}}} p_{\mathcal{M}}(\boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, \boldsymbol{V}_{\boldsymbol{F}^{(1)}} \backslash \boldsymbol{E}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}}, \boldsymbol{V}_{\boldsymbol{F}^{(n)}} \backslash \boldsymbol{E}^{(n)}_{\boldsymbol{F}^{(n)}}) \tag{4.21}$$

where $\boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}}$ are instantiations of variable subsets $\boldsymbol{E}^{(1)}_{\boldsymbol{F}^{(1)}} \subseteq \boldsymbol{V}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{E}^{(n)}_{\boldsymbol{F}^{(n)}} \subseteq \boldsymbol{V}_{\boldsymbol{F}^{(n)}}$ respectively, and correspond to events in each of the counterfactual worlds.

In order to perform counterfactual reasoning, the *twin network technique* [8] augments the SCM $\mathcal{M}$ by explictly adding the set of new variables $\boldsymbol{V}_{\boldsymbol{F}^{(j)}}$ corresponding to each intervention to the SCM (the word "twin" refers to the case where there are $n = 2$ interventions; more generally, we can refer to *n-world* networks[2]). Counterfactual inference then directly corresponds to an associational inference query of the form

[2]The corresponding graph has been called the *parallel worlds graph* by [6, 169].

(4.21) on this augmented SCM $\mathcal{M}'$. Unfortunately, n-world networks are much larger than the original Bayesian network, resulting in a corresponding jump to inferential complexity. It was recently shown that the treewidth w' of n-world networks is at most $n(w+1)-1$, where w is the treewidth of the original SCM graph over $\boldsymbol{U}, \boldsymbol{V}$ [73]; in other words, if the number of worlds is bounded, then counterfactual reasoning is only polynomially harder than associational reasoning. However, the significant increase (at least doubling) in treewidth can still limit scalability in practice.

In this section, we consider a restricted case of counterfactual reasoning, which, as we will show, can always be executed efficiently on a compilation of the *original* SCM $\mathcal{M}$, and, in particular, does not depend on the number of worlds. The restriction we impose is all of the events in Equation 4.21 must be *complete instantiations* of the variables in the corresponding world, except one. That is, we have that $\boldsymbol{E}^{(1)}_{\boldsymbol{F}^{(1)}} = \boldsymbol{V}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{E}^{(n-1)}_{\boldsymbol{F}^{(n-1)}} = \boldsymbol{V}_{\boldsymbol{F}^{(n-1)}}$ and $\boldsymbol{E}^{(n)}_{\boldsymbol{F}^{(n)}} \subseteq \boldsymbol{V}_{\boldsymbol{F}^{(n)}}$. While this is restrictive compared to the general counterfactual marginal, it is still a fairly broad class of queries: for example, it trivially includes evaluating any counterfactual probability $p_{\mathcal{M}}(\boldsymbol{v}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{v}_{\boldsymbol{F}^{(n)}})$ (evidence query) for any number of worlds. Our method applies the three-step process known as *abduction-action-prediction* [8]. In this approach, we seek to compute a conditional of the form $p_{\mathcal{M}}(\boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}} | \boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n-1)}_{\boldsymbol{F}^{(n-1)}})$ using the following steps:

1. **Abduction**: Compute a posterior $p_{\mathcal{M}}(\boldsymbol{U} | \boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n-1)}_{\boldsymbol{F}^{(n-1)}})$;

2. **Action**: Apply the intervention $\boldsymbol{F}^{(n)}$;

3. **Prediction**: Compute the distribution:

$$p_{\mathcal{M}}(\boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}} | \boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n-1)}_{\boldsymbol{F}^{(n-1)}}) = \sum_{\boldsymbol{U}} p_{\mathcal{M}}(\boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}} | \boldsymbol{U}) p_{\mathcal{M}}(\boldsymbol{U} | \boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n-1)}_{\boldsymbol{F}^{(n-1)}})$$

Unfortunately, this procedure is typically intractable as it involves summing over all values of the exogenous variables $\boldsymbol{U}$. Even though the distribution over $\boldsymbol{U}$ in the original SCM is fully factorized, the posterior over $\boldsymbol{U}$, in general, is highly correlated, due to conditioning on the events. In order to make progress, as previously mentioned,

```
Algorithm 4.3: Posterior Circuit Computation
  Input: Transpiled PC $\mathcal{C} = (G, \boldsymbol{\omega}(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}))$ symbolically computing $\mathcal{M}$,
         satisfying topological ordering constraints, event $\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}$
  Result: Modified PC $\mathcal{C}' = (G, \boldsymbol{\omega}'(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}))$
1 begin
2     For leaf nodes $L$ corresponding to an indicator $\mathbb{1}_{V_i=v_i}$, compute
        $p_{cond}(L) = \mathbb{1}_{v_i \models \boldsymbol{v}^{(1)}}$;
3     For leaf nodes $L$ corresponding to an indicator $\mathbb{1}_{U=u}$, compute $p_{cond}(L) = 1$;
4     For product nodes $P$, compute $p_{cond}(P) = \prod_{N_j \in \mathrm{ch}(P)} p_{cond}(N_j)$;
5     For sum nodes $T$ deciding on $V_i$ with exogenous parent values $\boldsymbol{u}_i$ and
        endogenous parent values $\boldsymbol{pa}_i$, compute $p_{cond}(T) = p_{cond}(N_j)$, where $N_j$
        corresponds to the value $F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i)$ of $V_i$; further, replace $T$ with a
        product node $P$, which has two children: $T$, and a leaf $L$ with constant
        leaf function $g_L() = \mathbb{1}_{v_i^{(1)} = F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}$;
6     For sum nodes $T$ deciding on $U$, compute
        $p_{cond}(T) = \sum_{N_j \in \mathrm{ch}(T)} \omega_{T,j} p_{cond}(N)$;
7     For root node $R$, assign $\omega'_{R,i} \leftarrow \frac{\omega_{R,i}}{p_{cond}(R)}$.
8 Return $\mathcal{C}' = (G, \boldsymbol{\omega}'(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}))$
```

we restrict to events $\boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n-1)}_{\boldsymbol{F}^{(n-1)}}$ that are complete instantiations of the variables, i.e. of the form $\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{v}^{(n-1)}_{\boldsymbol{F}^{(n-1)}}$.

We now consider implementing the abduction step. First, let us consider computing the posterior $p_{\mathcal{M}}(\boldsymbol{U}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$ for a single event $\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}$. This can be written as:

$$p_{\mathcal{M}}(\boldsymbol{U}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}) = \frac{p_{\mathcal{M}}(\boldsymbol{U}, \boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} \tag{4.22}$$

Thus, if we have any compiled circuit that symbolically computes the SCM, we can obtain a circuit that represents $p_{\mathcal{M}}(\boldsymbol{U}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$ over $\boldsymbol{U}$, simply by evaluating at $\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}$ and dividing by the scalar $p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$. Unfortunately, the resulting posterior may not permit efficient prediction, i.e. computation of the distribution $p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}) = \sum_{\boldsymbol{U}} p_{\mathcal{M}}(\boldsymbol{V}_{\boldsymbol{F}}|\boldsymbol{U}) p_{\mathcal{M}}(\boldsymbol{U}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$ for some new intervention $\boldsymbol{F}$. To overcome this, we need another condition; namely that the circuit satisfies topological constraints. This ensures that we can assign the parameters $\theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i} \in \{0, 1\}$ (corresponding to functional mechanisms $F_i(\boldsymbol{u}_i, \boldsymbol{pa}_i)$) to a sum node deciding on V_i (and with parent values $\boldsymbol{u}_i, \boldsymbol{pa}_i$) using our transpilation procedure.

In Algorithm 4.3, we provide a procedure for deriving a circuit that symbolically computes the joint posterior over exogenous variables $\boldsymbol{U}$ and (intervened) endogenous variables $\boldsymbol{V_F}$, $p_{\mathcal{M}}(\boldsymbol{U}, \boldsymbol{V_F}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}) = p_{\mathcal{M}}(\boldsymbol{V_F}|\boldsymbol{U})p_{\mathcal{M}}(\boldsymbol{U}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$. To understand what this means, we first derive the polynomial for $p_{\mathcal{M}}(\boldsymbol{U}, \boldsymbol{V_F}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$.

The SCM polynomial $f_{\mathcal{M}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) = p_{\mathcal{M}}(\boldsymbol{U}, \boldsymbol{V_F})$, where the values of $\boldsymbol{U}, \boldsymbol{V}$ are encoded into the indicators and the mechanisms $\boldsymbol{F}$ into the parameters as in Section 4.1.2. Every term $\prod_{U\in\boldsymbol{U}} \mathrm{Pr}_U(u) \prod_{i=1}^d \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i}$ of the SCM polynomial corresponds to a specific instantiation $\boldsymbol{u}, \boldsymbol{v}$ of $\boldsymbol{U}, \boldsymbol{V}$. The corresponding term $p_{\mathcal{M}}(\boldsymbol{u}, \boldsymbol{v_F}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$ is given by:

$$
\begin{aligned}
p_{\mathcal{M}}(\boldsymbol{u}, \boldsymbol{v_F}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}) &= p_{\mathcal{M}}(\boldsymbol{v_F}|\boldsymbol{u})p_{\mathcal{M}}(\boldsymbol{u}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}) && (4.23)\\
&= \frac{p_{\mathcal{M}}(\boldsymbol{v_F}, \boldsymbol{u})}{p_{\mathcal{M}}(\boldsymbol{u})} \frac{p_{\mathcal{M}}(\boldsymbol{u})p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}|\boldsymbol{u})}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} && (4.24)\\
&= p_{\mathcal{M}}(\boldsymbol{v_F}, \boldsymbol{u}) \frac{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}|\boldsymbol{u})}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} && (4.25)\\
&= p_{\mathcal{M}}(\boldsymbol{v_F}, \boldsymbol{u}) \frac{1}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} \prod_{i=1}^d \mathbb{1}_{v_i^{(1)}=F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})} && (4.26)\\
&= \frac{\prod_{i=1}^d \mathbb{1}_{v_i^{(1)}=F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} \prod_{U\in\boldsymbol{U}} \mathbb{1}_{U=u}\mathrm{Pr}_U(u) \prod_{i=1}^d \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i} && (4.27)
\end{aligned}
$$

The polynomial of the model $p_{\mathcal{M}}(\boldsymbol{U}, \boldsymbol{V_F}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$, which we write $f_{\mathcal{M}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}}$, is then given by:

$$
f_{\mathcal{M}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}} = \sum_{\boldsymbol{u}\in\mathrm{val}(\boldsymbol{U}), \boldsymbol{v}\in\mathrm{val}(\boldsymbol{V})} \frac{\prod_{i=1}^d \mathbb{1}_{v_i^{(1)}=F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} \prod_{U\in\boldsymbol{U}} \mathbb{1}_{U=u}\mathrm{Pr}_U(u) \prod_{i=1}^d \mathbb{1}_{V_i=v_i}\theta_{v_i|\boldsymbol{u}_i,\boldsymbol{pa}_i} \tag{4.28}
$$

This is in the form of an SCM polynomial, except that we have an extra factor $\frac{\prod_{i=1}^d \mathbb{1}_{v_i^{(1)}=F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})}$ which affects the exogenous variables. Now, if we have a transpiled PC $\mathcal{C}$ symbolically computing the SCM, then Algorithm 4.3 implements this extra factor by attaching the indicators to sum nodes deciding on the appropriate variable, and reweighting the root sum node weights by the scalar $\frac{1}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})}$, as computed by p_{cond}.

Proposition 4.3 (Correctness of Algorithm 4.3). *The output $\mathcal{C}'$ of Algorithm 4.3 has polynomial $f_{\mathcal{M}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}}$.*

As such, we can iterate this algorithm to condition on $\boldsymbol{v}^{(2)}_{\boldsymbol{F}^{(2)}}, \boldsymbol{v}^{(3)}_{\boldsymbol{F}^{(3)}}, ...\boldsymbol{v}^{(n-1)}_{\boldsymbol{F}^{(n-1)}}$, obtaining a circuit computing the polynomial $f_{\mathcal{M}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}},...,\boldsymbol{v}^{(n-1)}_{\boldsymbol{F}^{(n-1)}}} = p_{\mathcal{M}}(\boldsymbol{U}, \boldsymbol{V}_{\boldsymbol{F}}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}, ...\boldsymbol{v}^{(n-1)}_{\boldsymbol{F}^{(n-1)}})$. Perfomring the prediction step to compute $p_{\mathcal{M}}(\boldsymbol{e}^{(n)}_{\boldsymbol{F}^{(n)}}|\boldsymbol{e}^{(1)}_{\boldsymbol{F}^{(1)}}, ..., \boldsymbol{e}^{(n-1)}_{\boldsymbol{F}^{(n-1)}})$ then simply amounts to a marginal query on the final circuit with $\boldsymbol{F} = \boldsymbol{F}^{(n)}$.

4.4 Experiments

In this section, we empirically evaluate our algorithms for the interventional robustness problem. In particular, we are interested in evaluating the scalability of the approach to large causal Bayesian networks, as well as the tightness of the upper and lower bounds obtained.

4.4.1 Robustness of Classifiers to Causal Interventions

We extend the setting of the case study in Section 4.2.3 to different credal intervention sets on five Bayesian networks, namely `child`, `insurance`, `win95pts`, `hepar2` and `andes`. In Table 4.2, we perform a quantitative analysis of the tightness of the bounds produced on the false negative and false positive probabilities of the classifiers. For each network, we compute lower and upper bounds on the false negative/positive probability under different credal intervention sets. Overall, we find small or nonexistent gaps between the lower and upper bounds across all networks and intervention sets evaluated, suggesting that in many settings of interest it is possible to obtain tight guarantees using our algorithms. Further, both bounding algorithms are very fast to execute, taking no more than a few seconds for each run. This is remarkable given the sizes of the intervention sets. For instance, for the insurance network, the parametric intervention set P2 (second row) covers involves credal sets over the mechanisms for 6 variables, 248 parameters, and $\sim 10^{36}$ different possible interventions (extremal points of the credal set).

4.4.2 Credal Inference Benchmarks

We additionally evaluate our method on the new CREPO [20] credal inference benchmark, which contains many small-to-moderately sized networks (< 10 nodes) and

Network	False Negatives				False Positives			
	BeforeIntv	LBound	UBound	Δ	BeforeIntv	LBound	UBound	Δ
child	0.06922	0.07098	0.07098	**0**	0.1629	0.1947	0.1947	**0**
	0.06922	0.07325	0.07329	**0.00004**	0.1629	0.2762	0.3069	**0.0307**
	0.06922	0.06978	0.07127	**0.00149**	0.1629	0.1717	0.2009	**0.0292**
insurance	0.02453	0.1181	0.1276	**0.0095**	0.1981	0.4157	0.4161	**0.0004**
	0.02453	0.3275	0.3433	**0.0158**	0.1981	0.9123	0.9130	**0.0007**
	0.02453	0.02453	0.02453	**0**	0.1981	0.1981	0.1981	**0**
	0.02453	0.1181	0.1297	**0.0116**	0.1981	0.4157	0.4168	**0.0011**
win95pts	0.2106	0.2111	0.2111	**0**	0.005170	0.005416	0.005445	**0.000029**
	0.2106	0.2163	0.2191	**0.0028**	0.005170	0.007200	0.008665	**0.001465**
	0.2106	0.2972	0.2985	**0.0013**	0.005170	0.01430	0.01445	**0.00015**
	0.2106	0.2109	0.2117	**0.0008**	0.005170	0.05494	0.05674	**0.00180**
hepar2	0.03673	0.09445	0.09445	**0**	0.2360	0.2408	0.2408	**0**
	0.03673	0.09585	0.09585	**0**	0.2360	0.9041	0.9041	**0**
	0.03673	0.1029	0.1029	**0**	0.2360	0.43758	0.43773	**0.00015**
	0.03673	0.1029	0.1029	**0**	0.2360	0.43758	0.43793	**0.00035**
andes	0.001400	0.001400	0.002540	**0.001140**	0	0	0	**0**
	0.001400	0.001400	0.002656	**0.001256**	0	0	0	**0**

Table 4.2: Analysis of the tightness of bounds (on probability of false negatives/false positives) produced by the upper bounding and lower bounding algorithms. For each network, each row corresponds to a different credal intervention set. We show the probability of false negatives/false positives in the original Bayesian network (BeforeIntv), along with lower and upper bounds under each intervention set.

corresponding credal sets. We include three of our methods: i) **CUB**, which computes an upper bound; ii) **CUB**$_{max}$, which searches over elimination orders satisfying the required topological constraints, in order to obtain a better bound; and iii) **CLB**, which computes a lower bound. We compare the performance of our method to exact credal variable elimination [39] (where feasible) and ApproxLP [4], an approximate method returning a lower bound which has been shown to be state of the art both in terms of scalability and accuracy of inferences. We split CREPO into two subsets, CREPO-exact where an exact solution could be computed, and CREPO-hard where the exact method ran out of memory. For further comparison, we include the (much larger) 70-node `hepar2` network in our tests.

In Table 4.3, for all benchmarks we report the time taken by each method. For CREPO-exact, we compute the average difference in computed probability to the exact result (positive/negative for upper/lower bounds respectively), while for the other sets, we report the average difference to the best upper bound, as the exact

Network		**Exact**	**ApproxLP**	**CLB**	**CUB**	**CUB$_{max}$**
CREPO-exact	Diff	0	-0.0523	-0.0432	0.0018	0.0015
	Time(ms)	626	384	46(6)	2(6)	209(618)
CREPO-hard	Diff	-	-0.0529	-0.0742	0.0220	0
	Time(ms)	-	1154	65(6)	2(6)	231(618)
Hepar2	Diff	-	-	-0.0917	0	-
	Time (s)	-	-	429(287)	4(287)	-

Table 4.3: Average computation time (compilation time in parenthesis) and difference in computed probability to exact/upper bound. $-$ indicates the method failed to complete due to time or memory limits.

result is not known. Remarkably, we see that our upper-bounding and lower-bounding algorithms dominate ApproxLP on CREPO-exact, with better lower bounds being produced in an order of magnitude less time. On CREPO-hard, our upper bounding is the only method capable of providing guarantees. Meanwhile, our lower bound performs worse on average than ApproxLP but only by a small amount while using significantly less time. Finally, we see that our method is the only one to scale to the `hepar2` network, completing in a reasonable amount of time.

4.5 Discussion

In this Chapter, we consider causal reasoning in the setting where the causal graph is known. In particular, we began by demonstrating that compiled representations of causal graphs (either causal Bayesian network or structural causal model) are sufficient to answer any interventional ($\mathcal{L}_2$) marginal query. Motivated by this, we investigated two more difficult classes of causal queries; namely, interventional robustness on CBNs, and computing counterfactuals on SCMs. In both cases, we found that the same condition on compiled circuits (respecting topological orderings) was sufficient to soundly approximate/compute the query. Crucially, this condition is purely a function of the topology of the causal graph, meaning that once compiled, the circuit can be used to answer any query in linear time in the size of circuit.

Empirically, we demonstrated the application of the interventional robustness query to analysing the robustness of classifiers. We also compared our (guaranteed) lower and upper bounding methods to existing methods for the equivalent problem of credal marginal inference, and found that our methods are significantly more scalable. Further, whereas credal inference methods typically can only infer a single credal marginal query at a time starting from the Bayesian network, the cost of multiple queries is amortized by our compilation procedure.

The tractability of compiled circuits for difficult causal queries makes them a good candidate as a tractable causal model. However, there are two important factors that limit the applicability of compilation. The first relates to the assumption that all variables are discrete. To remedy this, it may be possible to adapt recent works that extended weighted model counting (upon which arithmetic circuit compilation is based) to weighted model integration [10, 54], which works on hybrid domains. The second limitation is that we are limited to compiled circuits based on causal graphs, the size of which is unavoidably controlled by the treewidth. As a result, scalability is limited by the treewidth of the causal graph, which may be prohibitive even if the observational distribution could be represented compactly (e.g. via context-specific independences).

Chapter 5

Tractability of Causal Inference

Contents

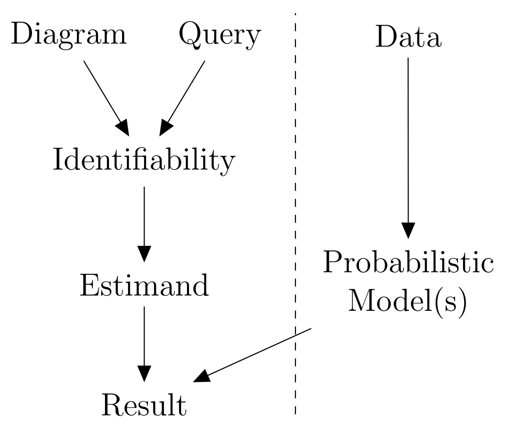


Figure 5.1: Separating causal and probabilistic models

In the previous Chapter, we studied tractable representations of causal graphs as compiled probabilistic circuits. In particular, we focused on symbolic compilation methods, where the compiled PC contains symbolic parameters corresponding directly to the parameters of the causal graph, allowing us to clearly define interventional semantics on the circuit by reference to the corresponding interventions on the causal graph. In this setting, applying interventions is as easy as changing the parameters of the circuit. Further, we have seen how compiled circuits satisfying the decision property, and particular orderings of the variables, admit causal interpretations of the mixtures represented by sum nodes.

However, the tractability of causal learning and reasoning of such models comes at a significant cost that we have ignored thus far: namely, that assumptions about the causal graph are “baked in” to the compiled circuit. On the one hand, this introduces an information bottleneck where we must commit to a specific, fully-specified graph before performing any learning. On the other hand, there is a computational bottleneck in which learning from different datasets *for the same graph* is easy, but learning over different graphs, even with the same dataset, requires a separate expensive compilation step for each graph.

Even if we are willing to commit to a particular, fully-specified causal graph, another issue with compiled causal models is that we are limited to learning probability distributions specified as a Bayesian network over discrete variables. While this is sufficiently expressive to specify (or approximate) any causal model consistent with the given causal graph, many high-dimensional distributions may lead to highly connected graphs where inference is intractable. In the tractable probabilistic modelling literature, dramatic advances in expressivity and scalability have been achieved moving beyond traditional tractable graphical models such as hidden Markov models and low-treewidth Bayesian networks, to tractable deep generative models such as sum-product networks and probabilistic sentential decision diagrams, while retaining tractability of inference. Learning algorithms for the structure and parameters of these circuit-based models can exploit detailed information such as context-specific independences for compactness, with recent learners showing the scalability of probabilistic circuits to high-dimensional datasets outside of the reach of classical tractable probabilistic graphical models [137, 107].

Taking a step back, there is no reason a-priori that compiled (arithmetic) circuits should be the *only* representation in which causal inference queries are tractable. In Figure 5.1, we depict an idealized conceptual setup of causal inference, in which the causal graph and query are separated from the data/probability distribution until the final reasoning step. For every identifiable query, the answer to the query is a function of the observational probability distribution. The fundamental question we ask in this chapter is, for which TPMs is this function tractable to compute? The answer to this question will shed light on key computational challenges in causal inference, as well as chart a way forward through a new subclass of probabilistic circuits.

5.1 On The Tractability of Exact Causal Inference

The problem of causal estimation is to estimate a given identifiable causal query on a data-generating system, given some assumed causal diagram over observed variables, and available data generated from that system. For example, consider a data-generating system over $\boldsymbol{V} = \boldsymbol{W} \cup \boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z}$ computing the observational distribution, where $\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ are disjoint sets of variables. In Figure 5.2 we depict two causal diagrams in which the interventional distribution $p_{\boldsymbol{X}}(\boldsymbol{Y})$[1] is identifiable, with the directed and bidirected edges indicating direct causation or confounding

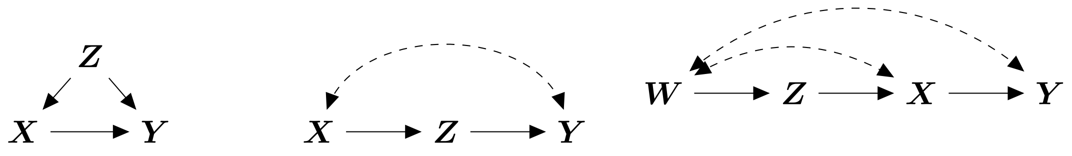
[1]In this chapter, we will use $p_{\boldsymbol{X}}(\boldsymbol{Y})$ to denote the interventional distribution $p_{\mathcal{M}}(\boldsymbol{Y}_{\boldsymbol{X}})$ in any SCM inducing the causal diagram.

between these sets of variables[2]. Given these assumptions, the goal is then to estimate the *interventional distribution* $p_{\boldsymbol{X}}(\boldsymbol{Y})$. For the graph in Figure 5.2a, this is identified exactly by the backdoor adjustment formula $p_{\boldsymbol{X}}(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z})p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$. For example, in a medical treatment setting, $\boldsymbol{Z}$ could constitute features of an individual patient influencing whether they take a particular treatment $\boldsymbol{X}$ as well as their clinical outcome $\boldsymbol{Y}$. For the frontdoor graph, we instead have the frontdoor formula $p_{\boldsymbol{X}}(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}) \sum_{\boldsymbol{X}'} p(\boldsymbol{X}')p(\boldsymbol{Y}|\boldsymbol{X}', \boldsymbol{Z})$, and in the case of the napkin graph in Figure 5.2c we can obtain $p_{\boldsymbol{X}}(\boldsymbol{Y}) = \frac{\sum_{\boldsymbol{W}} p(\boldsymbol{X},\boldsymbol{Y}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p(\boldsymbol{X}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W})}$, by applying the do-calculus.

Even though the interventional distribution is identified in these cases, causal estimation remains a significant practical challenge for complex, high-dimensional distributions p for two main reasons. The first challenge is to obtain accurate probabilistic models for the observational distribution, perhaps learned from data. However, even given an *exact* model for the observational distribution, a second *computational* challenge remains: to compute the function given by the causal formula. The identification formulae involve potentially intractable summations, meaning that we must typically resort to an approximate or heuristic algorithm such as a Monte Carlo estimate. For other causal formulae, the situation is even more complex as the approximation scheme may have to be hand-designed and/or come with little or no guarantees; for example, for the frontdoor and napkin formulae, effective estimators have only very recently been developed [63, 85].

To bypass these difficulties, we investigate in this Chapter whether it is feasible to *exactly* compute the interventional distribution, by identifying and restricting to classes of probabilistic models where the identifying formulae are tractable to compute. As a natural starting point, we consider the class of decomposable and smooth probabilistic circuits, as these are tractable for the corresponding associational ($\mathcal{L}_1$) queries of computing marginal and conditional probabilities. The tractability of computing interventional $\mathcal{L}_2$ probabilities for PCs, however, has remained an important open question. For example, for the backdoor formula, though we can compute $p(\boldsymbol{z})$ and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ for any specific values of $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ in linear time, we cannot directly use this

[2]This could also represent a cluster DAG, where there are additional causal relations between the variables in a given set (cluster); the do-calculus has been shown to be valid for causal-effect identification in this case as well [2].



(a) Backdoor (b) Frontdoor (c) Napkin

Figure 5.2: Examples of causal diagrams

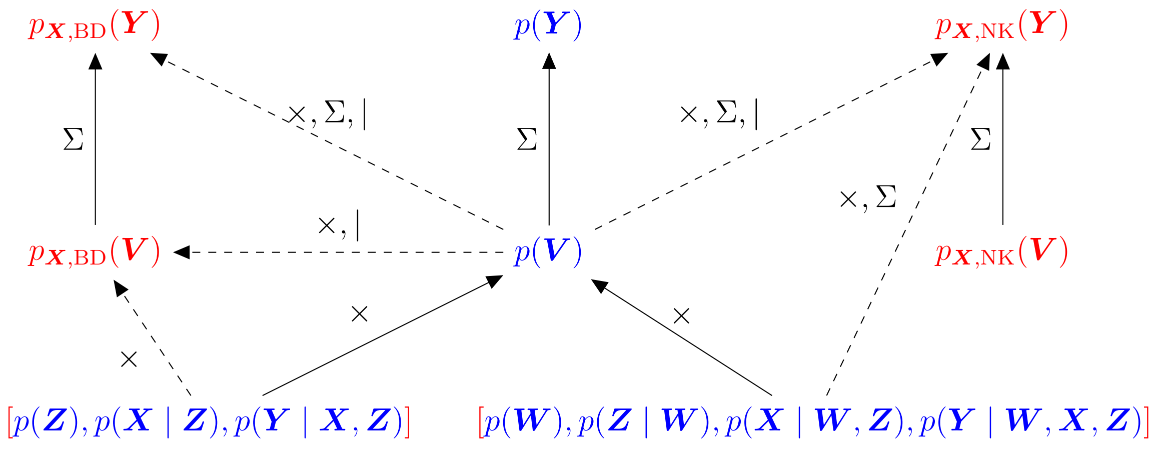

$p_{\boldsymbol{X},\mathrm{BD}}(\boldsymbol{Y})$
$p(\boldsymbol{Y})$
$p_{\boldsymbol{X},\mathrm{NK}}(\boldsymbol{Y})$
Σ
$\times, \Sigma, |$
Σ
$\times, \Sigma, |$
Σ
$\times, \Sigma$
$\times, |$
$p_{\boldsymbol{X},\mathrm{BD}}(\boldsymbol{V})$
$p(\boldsymbol{V})$
$p_{\boldsymbol{X},\mathrm{NK}}(\boldsymbol{V})$
$\times$
$\times$
$\times$
$[p(\boldsymbol{Z}), p(\boldsymbol{X} \mid \boldsymbol{Z}), p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{Z})]$
$[p(\boldsymbol{W}), p(\boldsymbol{Z} \mid \boldsymbol{W}), p(\boldsymbol{X} \mid \boldsymbol{W}, \boldsymbol{Z}), p(\boldsymbol{Y} \mid \boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Z})]$

Figure 5.3: Probabilistic (blue) and causal (red) quantities, with probabilistic (—) and causal inference (– –) for backdoor/napkin graphs. Note that $p_{\boldsymbol{X},\mathrm{NK}}(\boldsymbol{V})$ has no incoming arrows as it is non-identifiable.

to compute the interventional distribution, due to summation over $\boldsymbol{Z}$ which takes time exponential in $|\boldsymbol{Z}|$. In the rest of the section, we will investigate the fundamental difficulties associated with computing interventional distributions.

5.1.1 Causal and Probabilistic Inference

To illustrate the computational challenge of causal inference as compared to probabilistic inference, we illustrate the computational procedures underlying probabilistic and causal inference tasks for the backdoor and napkin graphs in Figure 5.3. The Figure consists of three layers of probabilistic/causal quantities. In the bottom layer, we have two *causal* (i.e. following a topological order on the observed variables) factorizations of the joint distribution, in the backdoor and napkin causal graphs respectively. In the middle layer, we have three joint distributions over the variables $\boldsymbol{V}$: the observational joint $p(\boldsymbol{V})$, and the interventional joints $p_{\boldsymbol{X},\mathrm{BD}}(\boldsymbol{V})$, $p_{\boldsymbol{X},\mathrm{NK}}(\boldsymbol{V})$ in the backdoor and napkin cases respectively. Finally, in the top layer we have the observational and interventional marginal distributions on $\boldsymbol{Y} \subseteq \boldsymbol{V}$.

In probabilistic inference, we are interested in computing marginal probabilities $p(\boldsymbol{Y})$, given the corresponding joint distribution $p(\boldsymbol{V})$ expressed in some form. Given either of the factorizations at the bottom of the Figure, probabilistic inference amounts to computing *products* ($\times$) of the factors, and then a single *marginalization* (Σ) operation, i.e $p(\boldsymbol{Y}) = \sum_{\boldsymbol{V} \setminus \boldsymbol{Y}} p(\boldsymbol{Z}) \times p(\boldsymbol{X}|\boldsymbol{Z}) \times p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$ or $p(\boldsymbol{Y}) = \sum_{\boldsymbol{V} \setminus \boldsymbol{Y}} p(\boldsymbol{W}) \times p(\boldsymbol{Z}|\boldsymbol{W}) \times p(\boldsymbol{X}|\boldsymbol{W}, \boldsymbol{Z}) \times p(\boldsymbol{Y}|\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Z})$. Similarly, if we are given a model of the interventional distribution $p_{\boldsymbol{X}}(\boldsymbol{V})$ (under either the backdoor or napkin cases), then computing interventional probabilities simply requires a single marginalization on the intervened model. In other words, probabilistic inference is given by a marginalization over products.

In causal inference, we instead wish to compute the interventional (marginal) distribution $p_{\boldsymbol{X}}(\boldsymbol{Y})$ using models of the observational distribution $p(\boldsymbol{V})$. In the backdoor case, if we are given the factorization $p(\boldsymbol{Z}), p(\boldsymbol{X}|\boldsymbol{Z}), p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$, then since there are no latent confounders, we can derive the interventional joint distribution as a *product* $p_{\boldsymbol{X}}(\boldsymbol{V}) = p(\boldsymbol{Z}) \times p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$, and then the interventional marginal $p_{\boldsymbol{X}}(\boldsymbol{Y}) = \sum_{\boldsymbol{X}} p(\boldsymbol{Z}) \times p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$ using a *marginalization* operation, as in probabilistic inference. This is possible because the interventional joint distribution $p_{\boldsymbol{X}}(\boldsymbol{V})$ is identified, and can be expressed using a modified version of the *causal factorization* of the observational joint distribution. As a result, given a causal factorization of the observational joint in which all variables are modelled, causal (marginal) inference is no harder than probabilistic inference. This is precisely the case we examined in the previous Chapter.

However, this turns out to be a special case; if we do not have access to this causal factorization, or if there exist latent confounders that are not present in the probabilistic model, then causal inference becomes more complex. In the backdoor case, if we are given the observational joint $p(\boldsymbol{V})$, or a non-causal factorization (e.g. $p(\boldsymbol{Y}), p(\boldsymbol{Z}|\boldsymbol{Y}), p(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{Z})$), then we can no longer express the interventional marginal as a marginalization over products. Instead, to use the backdoor adjustment formula, we need to implement a conditioning ($|$) operation, which outputs a model $p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$ given $p(\boldsymbol{V})$, followed by a product, and then a marginalization. In the napkin case, where there are unobserved confounders, the interventional joint $p_{\boldsymbol{X}}(\boldsymbol{V})$ is not even identifiable (and so there is no incoming arrow in the diagram), even though the interventional marginal $p_{\boldsymbol{X}}(\boldsymbol{Y})$ is identifiable. The napkin formula $p_{\boldsymbol{X}}(\boldsymbol{Y}) = \frac{\sum_{\boldsymbol{W}} p(\boldsymbol{X},\boldsymbol{Y}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p(\boldsymbol{X}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W})}$ can also be expressed as a composition of products, marginals and conditionals. To do so, we first formally define these operations:

Definition 5.1 (Marginalization). *$\texttt{MARG}(\cdot;\boldsymbol{W})$ is a unary operation that takes as input a probabilistic model $\mathcal{P}$ over variables $\boldsymbol{V}$ (and a subset $\boldsymbol{W} \subseteq \boldsymbol{V}$), and outputs a probabilistic model over $\boldsymbol{V} \setminus \boldsymbol{W}$ representing $\sum_{\boldsymbol{W}} p_{\mathcal{P}}(\boldsymbol{W}) = p_{\mathcal{P}}(\boldsymbol{V})$.*

Definition 5.2 (Product). *$\texttt{PROD}(\cdot,\cdot)$ is a binary operation that takes as inputs two probabilistic models $\mathcal{P}, \mathcal{P}'$ over variables $\boldsymbol{V}, \boldsymbol{V}'$ and outputs a probabilistic model representing $p_{\mathcal{P}}(\boldsymbol{V}) \times p_{\mathcal{P}'}(\boldsymbol{V}')$ over variables $\boldsymbol{V} \cup \boldsymbol{V}'$.*

Definition 5.3 (Conditioning). *$\texttt{COND}(\cdot;\boldsymbol{W})$ is a unary operation that takes as input a probabilistic model $\mathcal{P}$ over variables $\boldsymbol{V}$ (and a subset $\boldsymbol{W} \subseteq \boldsymbol{V}$), and outputs a probabilistic model over $\boldsymbol{V} \setminus \boldsymbol{W}$ representing:*

$$p_{\mathcal{P}}(\boldsymbol{V} \setminus \boldsymbol{W} \mid \boldsymbol{W}) := \begin{cases} \frac{p_{\mathcal{P}}(\boldsymbol{V})}{p_{\mathcal{P}}(\boldsymbol{W})} & \text{if } p_{\mathcal{P}}(\boldsymbol{W}) > 0 \\ 0 & \text{if } p_{\mathcal{P}}(\boldsymbol{W}) = 0 \end{cases} \tag{5.1}$$

Now given a model $\mathcal{P}_1$ representing $p(\boldsymbol{V})$, we have

$$\mathcal{P}_2 = \texttt{COND}(\mathcal{P}_1; \boldsymbol{W} \cup \boldsymbol{Z}); p_{\mathcal{P}_2}(\boldsymbol{W},\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z}) = p(\boldsymbol{X},\boldsymbol{Y}|\boldsymbol{W},\boldsymbol{Z}) \tag{5.2}$$

$$\mathcal{P}_3 = \texttt{MARG}(\mathcal{P}_1; \boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z}); p_{\mathcal{P}_3}(\boldsymbol{W}) = p(\boldsymbol{W}) \tag{5.3}$$

$$\mathcal{P}_4 = \texttt{PROD}(\mathcal{P}_2; \mathcal{P}_3); p_{\mathcal{P}_4}(\boldsymbol{W},\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z}) = p(\boldsymbol{X},\boldsymbol{Y}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W}) \tag{5.4}$$

$$\mathcal{P}_5 = \texttt{MARG}(\mathcal{P}_4; \boldsymbol{W}); p_{\mathcal{P}_5}(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z}) = \sum_{\boldsymbol{W}} p(\boldsymbol{X},\boldsymbol{Y}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W}) \tag{5.5}$$

$$\mathcal{P}_6 = \texttt{COND}(\mathcal{P}_5; \boldsymbol{X} \cup \boldsymbol{Z}); p_{\mathcal{P}_6}(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z}) = \frac{\sum_{\boldsymbol{W}} p(\boldsymbol{X},\boldsymbol{Y}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p(\boldsymbol{X}|\boldsymbol{W},\boldsymbol{Z})p(\boldsymbol{W})} \tag{5.6}$$

Then, we have $p_{\mathcal{P}_6}(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z}) = p_{\boldsymbol{X}}(\boldsymbol{Y})$ in the napkin case[3]. Note that in this case, even if we were given the causal factorization, the interventional marginal is still not expressible as a marginalization over products.

[3]Assuming that the distribution is generated by an SCM respecting the napkin diagram, the napkin formula is valid and equivalent for any value of $\boldsymbol{Z}$.

It turns out that we can, in theory, generalize this analysis to *any* identifiable interventional distribution, as it is always possible to express the interventional distribution as a composition of marginalization, product, and conditional operations. This is implemented by the ID algorithm, which employs only these operations and is complete for causal identification [168, 169, 153]. Thus, to analyze the tractability of causal inference on a class of probabilistic models, we can analyze the tractability of these operations on those models.

5.1.2 Conditioning in Probabilistic Circuits

Turning to probabilistic circuits, we know that the marginalization operation is tractable for decomposable and smooth circuits. For computing products of two circuits, the most direct approach is to simply introduce a product node, which has the roots of the circuits as children. However, such a circuit would not be decomposable if the circuits share any variables. Instead, if the circuits satisfy a property known as structured decomposability (which we will introduce shortly), then it is possible to compute the product as a structured decomposable (implying decomposability) circuit in time linear in the product of the sizes of the input circuits [162, 185]. This allows us to perform *probabilistic inference* simply by applying marginalization over products, which is tractable so long as the number of products is fixed; such an approach is sometimes called *bottom-up* compilation (in contrast to the *top-down* compilation methods discussed in the previous Chapter) [162, 47].

However, the tractability of the conditioning operation for PCs is not yet well understood. It is important to make a distinction between computing conditional probabilities $p(\boldsymbol{v} \setminus \boldsymbol{w} | \boldsymbol{w})$, which is possible in linear time by computing the ratio of two marginal probabilities, and the conditioning operation, which returns another PC representing the function $p(\boldsymbol{V} \setminus \boldsymbol{W} | \boldsymbol{W})$. For this reason, we will refer to the conditioning operation as *symbolic conditioning.* To analyze the tractability of this operation, we need to define a new property, known as *marginal determinism*:

Definition 5.4 (Marginalized Support)**.** *Given a PC node N and set of variables $\boldsymbol{Q}$, we define the marginalized support of N with respect to $\boldsymbol{Q}$ as $supp_{\boldsymbol{Q}}(N) := \{\boldsymbol{q} \in val(\boldsymbol{Q}) : p_N(\boldsymbol{q}) > 0\}$.*

Recall that any node N defines a function $p_N(\phi(N))$ over a set of variables $\phi(N)$, known as its *scope*, and the *support* of N consists of all values of $\phi(N)$ such that the function is positive. The *marginalized support* represents the set of partial instantiations $\boldsymbol{q}$ of the scope, such that there exists a full instantiation with positive value. Note that $\boldsymbol{Q}$ can also contain variables outside of $\phi(N)$; in a slight abuse of notation, we write $p_N(\boldsymbol{q})$ for $p_N(\boldsymbol{q} \cap \phi(N))$.

Definition 5.5 (Marginal Determinism)**.** *A sum node T is marginal deterministic with respect to a set of variables $\boldsymbol{Q}$ (written $\boldsymbol{Q}$-deterministic) if the children of the sum node have disjoint marginalized support, i.e. $supp_{\boldsymbol{Q}}(N_i) \cap supp_{\boldsymbol{Q}}(N_j) = \emptyset$ for N_i, N_j distinct children of T.*

Definition 5.6 (Marginal Determinism of PC)**.** *A PC is marginal deterministic with respect to a subset $\boldsymbol{Q} \subseteq \boldsymbol{V}$ (written $\boldsymbol{Q}$-deterministic) if for every sum node T, either:*

- *$\phi(T)$ does not overlap with $\boldsymbol{Q}$, i.e. $\phi(T) \cap \boldsymbol{Q} = \emptyset$; or*
- *The sum node T is $\boldsymbol{Q}$-deterministic.*

Intuitively, a PC over variables $\boldsymbol{V}$ is $\boldsymbol{Q}$-deterministic, if the circuit obtained after marginalizing out $\boldsymbol{V} \setminus \boldsymbol{Q}$ is deterministic. In particular, $\boldsymbol{V}$-determinism is equivalent to the usual definition of determinism in PCs. Marginal determinism was very recently introduced [33] as a sufficient condition for tractability of marginal MAP; that is, computing $\max_{\boldsymbol{Q}} p_{\mathcal{C}}(\boldsymbol{Q}, \boldsymbol{e})$ for disjoint sets of variables $\boldsymbol{Q}, \boldsymbol{E} \subseteq \boldsymbol{V}$ and $\boldsymbol{e}$ an instantiation of $\boldsymbol{E}$. In particular, it was shown $\max_{\boldsymbol{Q}} p_{\mathcal{C}}(\boldsymbol{Q}, \boldsymbol{e})$ is tractable in linear time if the circuit is $\boldsymbol{Q}$-deterministic, for the same set $\boldsymbol{Q}$. We show that the same condition is sufficient for tractable conditioning in linear time:

Proposition 5.1 (Tractable Symbolic Conditioning)**.** *Let $\mathcal{C}$ be a decomposable, smooth and $\boldsymbol{Q}$-deterministic circuit over variables $\boldsymbol{V}$. Then computing $\texttt{COND}(\mathcal{C}; \boldsymbol{Q})$ as a decomposable, smooth and $\boldsymbol{Q}$-deterministic circuit is tractable in $O(|\mathcal{C}|)$ time and space.*

The corresponding algorithm is shown in Figure 5.1, where each node N in the output circuit represents $p_N(\phi(N) \setminus \boldsymbol{Q} | \boldsymbol{Q})$. We assume that symbolic conditioning

Algorithm 5.1: `COND`$(\mathcal{C}; \boldsymbol{Q})$

```
Input: Decomposable, smooth and $\boldsymbol{Q}$-deterministic circuit $\mathcal{C} = (G, \boldsymbol{\omega}, g)$ over
       $\boldsymbol{V}$, conditioning set $\boldsymbol{Q} \subseteq \boldsymbol{V}$
Result: Decomposable, smooth and $\boldsymbol{Q}$-deterministic circuit $\mathcal{C}' = (G, \boldsymbol{\omega}', g')$
        over $\boldsymbol{V}$
1  $c \leftarrow []$;
2  for nodes $N \in G$ in bottom-up order do
3      if $N$ is leaf $L$ then
4          $c[L] \leftarrow \sum_{\phi(L)} g_L(\phi(L))$;        // normalizing constant of leaf fn
5          $g'_L \leftarrow$ COND$(g_L; \boldsymbol{Q})$;
6      else if $N$ is product $P$ then
7          $c[P] \leftarrow \prod_{N_i \in \mathrm{ch}(P)} c[N_i]$;
8      else if $N$ is sum $T$ then
9          $c[T] \leftarrow \sum_{N_i \in \mathrm{ch}(T)} \omega_{T,i} c[N_i]$;
10         if $\phi(T) \cap \boldsymbol{Q} \neq \emptyset$ then
11             $\omega'_{T,i} \leftarrow \mathbb{1}_{\omega_{T,i} > 0}$;        // Represents conditioning on $\boldsymbol{Q}$
12         else
13             $\omega'_{T,i} \leftarrow \frac{\omega_{T,i} c[N_i]}{\sum_{N_j \in \mathrm{ch}(T)} \omega_{T,j} c[N_j]}$;        // Renormalize weights
14 Return $\mathcal{C}' = (G, \boldsymbol{\omega}', g')$
```

is tractable on the leaf node functions. For example, if each leaf node had a single variable V in its scope, then if (a) $V \in \boldsymbol{Q}$ we would set $g'_L(V) = \mathbb{1}_{V \in \mathrm{supp}(L)}$, while if (b) $V \notin \boldsymbol{Q}$ then we would set $g'_L(V) = \frac{g_L(V)}{\sum_V g_L(V)}$. Intuitively, the algorithm takes advantage of the fact that by $\boldsymbol{Q}$-determinism, the support of $\boldsymbol{Q}$ is partitioned at (and only at, as any other sum nodes do not contain variables from $\boldsymbol{Q}$) sum nodes T where $\phi(T) \cap \boldsymbol{Q} \neq \emptyset$. We can thus implement the conditioning operation by equalizing the weights at such nodes. In the following Sections, we will derive a systematic theory and methodology for marginal determinism, that provides insight into why marginal MAP and symbolic conditioning share this tractability condition.

5.1.3 Hardness of the Backdoor Query

Returning to causal inference, we have now seen that symbolic conditioning is tractable with marginal determinism. For the backdoor formula, this means that if we have a $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic circuit, we can compute $p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z})$ efficiently. In the following Sections, we will see how to compose this with products and marginalization to tractably compute the interventional distribution. For now, we ask the opposite question: is the backdoor formula tractable for existing classes of circuits not satisfying marginal determinism? We consider the problem of computing a single interventional probability $p_{\boldsymbol{x}}(\boldsymbol{y})$ for specific values $\boldsymbol{x}, \boldsymbol{y}$ of $\boldsymbol{X}, \boldsymbol{Y}$, which we call the *backdoor query*. Hardness of this query would imply hardness of representing the interventional distribution $p_{\boldsymbol{X}}(\boldsymbol{Y})$ as any type of PC, since evaluating probabilities in PCs is always tractable. Despite the tractability of marginal probabilities for decomposable and smooth circuits, we now show that even the more restrictive conditions of structured decomposability and determinism are not sufficient to avoid hardness of interventional marginals:

Theorem 5.1 (Hardness of Backdoor Query)**.** *The backdoor query for decomposable and smooth PCs is* #P*-hard, even if the PC is structured decomposable and deterministic.*

It is worth noting that this hardness result pertains entirely to computing the backdoor query $\sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(y|x, \boldsymbol{Z})$, and not to a specific causal graph. The backdoor query is equal to the interventional marginal $p_{\boldsymbol{x}}(\boldsymbol{y})$ whenever the backdoor adjustment criterion holds for $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$, i.e. $\boldsymbol{Z}$ blocks all backdoor paths between $\boldsymbol{X}, \boldsymbol{Y}$. For example, for the frontdoor graph in Figure 5.2b, the interventional probability $p_{\boldsymbol{z}}(\boldsymbol{y})$ is identifiable through a backdoor adjustment on $\boldsymbol{X}$, and is thus hard to compute for structured decomposable and deterministic circuits.

Corollary 5.1. *For any interventional query $p_{\boldsymbol{x}}(\boldsymbol{y})$ and causal diagram G such that the query is identifiable through a backdoor adjustment, and the observational distribution $p(\boldsymbol{V})$ given as a decomposable and smooth circuit $\mathcal{C}$ encoding p, computing $p_{\boldsymbol{x}}(\boldsymbol{y})$ is* #P*-hard, even if the circuit is structured decomposable and deterministic.*

Proof. By the identifiability condition, we have that $p_{\boldsymbol{x}}(\boldsymbol{y}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{Z}) p(\boldsymbol{Z}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{Z})$, which is the backdoor query. Hardness of computing the causal effect then follows from hardness of backdoor queries for the probabilistic circuit. □

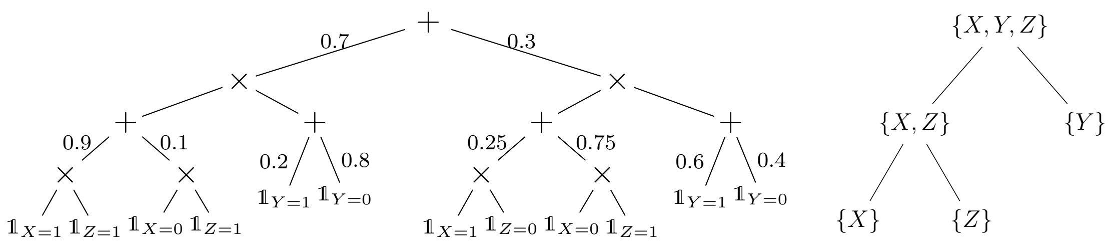

+
0.7
0.3
×
×
+
+
+
+
0.9
0.1
0.2
0.8
0.25
0.75
0.6
0.4
$\mathbb{1}_{X=1}$ $\mathbb{1}_{Z=1}$ $\mathbb{1}_{X=0}$ $\mathbb{1}_{Z=1}$
$\mathbb{1}_{Y=1}$ $\mathbb{1}_{Y=0}$
$\mathbb{1}_{X=1}$ $\mathbb{1}_{Z=0}$ $\mathbb{1}_{X=0}$ $\mathbb{1}_{Z=1}$
$\mathbb{1}_{Y=1}$ $\mathbb{1}_{Y=0}$
$\{X,Y,Z\}$
$\{X,Z\}$
$\{Y\}$
$\{X\}$
$\{Z\}$

Figure 5.4: Example of structured decomposable circuit, and vtree it respects.

5.2 A Theory of Marginal Determinism in Structured Decomposable Circuits

In the previous section, we have seen that marginal determinism enables tractable computation of the symbolic conditioning operation, which, alongside marginalization and products, is one of the key operations for causal inference. However, marginal determinism is not a single property, but rather a family of properties that is defined with respect to every subset of the variables of a PC. Previous work has shown that it is not possible for a circuit to satisfy marginal determinism with respect to all variable subsets unless the distribution represented by the circuit is fully factorized [33] (i.e. losing expressivity); however, it is not yet well understood which marginal determinism properties a circuit *can* simultaneously satisfy. Related to this, it is unclear how one can practically check or enforce marginal determinism(s) on a circuit [32]. These questions pose serious challenges to the ability to use circuits as a probabilistic model for tractable causal inference.

5.2.1 Structured Marginal Determinism

5.2.1.1 Structured Decomposability

In this section, we will develop a systematic theory of marginal determinism in *structured decomposable* circuits. Structured decomposability [142, 89, 165] is a stronger version of decomposability that requires the scope of nodes in the circuit to decompose according to a *vtree*.

Definition 5.7 (Vtree)**.** *A vtree $v = (M, E)$ for a set of variables $\boldsymbol{V}$ is a rooted binary tree with nodes M and edges E, whose leaves m each correspond to a distinct variable $\phi(m) \in \boldsymbol{V}$.*

We define the *scope* of a leaf m to be $\phi(m)$, and the scope of any other node to be $\phi(m) = \cup_{m_i \in ch(m)} \phi(m_i)$ (treating leaf scopes as singleton sets). Further, we write $v_m = (M_m, E_m)$ to denote the vtree rooted at m. A vtree provides a "reference" for the decomposition of scope in a circuit: in particular, it requires each product node to have exactly two children[4], and for the partitioning of scope for the product node to match the partitioning of scope for a vtree node.

Definition 5.8 (PC respecting vtree)**.** *Let $\mathcal{C}$ be a PC and $v = (M, E)$ be a vtree, both over variables $\boldsymbol{V}$. We say that $\mathcal{C}$ respects v if for every product node P corresponds to (is normalized for) a vtree node m; that is, P has exactly two children* $\mathrm{ch}(P) = \{N_1, N_2\}$, *and $\phi(P_i) = \phi(m_i)$ for $i = 1, 2$ where* $\mathrm{ch}(m) = (m_1, m_2)$.

A PC $\mathcal{C}$ is structured decomposable if it respects some vtree v.

This immediately implies that the scope of P matches the scope of m, i.e. $\phi(P) = \phi(N_1) \cup \phi(N_2) = \phi(m_1) \cup \phi(m_2) = \phi(m)$. If we additionally assume smoothness, then *every* node in the circuit has a corresponding vtree node with the same scope:

Proposition 5.2 (Scope of Structured Decomposable and Smooth Circuits)**.** *Given a structured decomposable and smooth circuit $\mathcal{C}$ respecting vtree v, with at least one product node, every node N in the circuit has a corresponding vtree node m such that $\phi(N) = \phi(m)$.*

An example of a structured decomposable and smooth circuit, and the vtree it respects, is shown in Figure 5.4. We restrict to structured decomposable (and smooth) circuits for two main reasons. Firstly, this property is required for tractable computation of the product of two circuits respecting *compatible* vtrees [162, 185], which is important for causal inference. Secondly, as we will see, fixing the variable decomposition using a vtree allows us to compactly describe the marginal determinisms that a circuit satisfies, a formulation which we call *structured marginal determinism.* We will show in particular how structured marginal determinism enables us to easily and efficiently check and enforce arbitrary marginal determinisms on a circuit. Along the way, we will critically examine how marginal determinism relates to other existing support properties, such as determinism, the decision property, and strong determinism.

[4]This is not a significant restriction, as every product can be converted into a series of binary products.

5.2.1.2 Properties of Marginal Determinism

We begin our study of marginal determinism, by noting that there is no straightforward relation between $\boldsymbol{Q}$-determinism and $\boldsymbol{Q}'$-determinism for PCs, for different sets $\boldsymbol{Q}, \boldsymbol{Q}'$. Recall that a PC $\mathcal{C}$ is defined to be $\boldsymbol{Q}$-deterministic if every sum node T with overlapping scope (i.e. $\boldsymbol{Q} \cap \phi(T)$) is $\boldsymbol{Q}$-deterministic. Note, however, that there is no such requirement on sum nodes with disjoint scope from $\boldsymbol{Q}$. Thus, for example, neither determinism ($\boldsymbol{V}$-determinism) nor $\boldsymbol{Q}$-determinism imply each other in general; for example, a circuit can be $\boldsymbol{Q}$-deterministic but not deterministic if there exist some sum nodes with $\phi(T) \cap \boldsymbol{Q} = \emptyset$. Thus, we use the following definition to capture the $\boldsymbol{Q}$-determinisms a circuit satisfies:

Definition 5.9 (md-signature)**.** *For a PC $\mathcal{C}$, the md-signature $\mathcal{Q}(\mathcal{C})$ is defined to be the family of all subsets $\boldsymbol{Q} \subseteq \boldsymbol{V}$ such that $\mathcal{C}$ is $\boldsymbol{Q}$-deterministic.*

The md-signature $\mathcal{Q}(\mathcal{C})$ can be viewed as a *signature* of the tractability of the PC; for example, two PCs with the same $\mathcal{Q}$ would enable efficient marginal MAP/symbolic conditioning computation for the same sets $\boldsymbol{Q}$. Unfortunately, the md-signature is given by a powerset of the powerset over variables $\boldsymbol{V}$, which is clearly not feasible to characterize explicitly (there are $2^{2^{|\boldsymbol{V}|}}$ possible md-signatures). We thus propose a more systematic approach to specifying support properties. The key idea is to *locally* characterize the support properties that each *sum node* satisfies, as opposed to specifying the *global* properties of the circuit as in the md-signature. For a given PC $\mathcal{C}$, let ψ be a function, that maps any sum node T in that PC, to the set of all sets $\boldsymbol{Q}$ such that T is $\boldsymbol{Q}$-deterministic; we call this a *labelling function.* Note that the labelling function ψ is a *specification* of marginal determinism for the circuit; that is, it is sufficient to determine whether the circuit is $\boldsymbol{Q}$-deterministic for any $\boldsymbol{Q}$ (by definition), and thus to deduce the md-signature $\mathcal{Q}(\mathcal{C})$.

Unfortunately, the labelling function is also too large to characterize explicitly. We make two observations that allow us to simplify the labelling function, one straightforward, and one more subtle. Firstly, we note that $\boldsymbol{Q}$-determinism for a circuit imposes the same requirement on all sum nodes with the same scope; thus we restrict ψ to have the same value for all sum nodes with the same scope. For structured decomposable circuits, where each sum node T corresponds to a *vtree* node m (i.e. $\phi(T) = \phi(m)$), we can thus write $\psi(m)$ as a function of the vtree node m.

The second observation is that, under some assumptions, we can actually specify $\psi(m)$ using a *single* set $\boldsymbol{Q} \subseteq \boldsymbol{V}$. In order to show this, we need to first show a couple of results regarding marginal determinism.

Proposition 5.3 (Conflicting $\boldsymbol{Q}$-Determinisms for Sum Nodes)**.** *Let T be a non-trivial*[5] *sum node T, and let $\boldsymbol{Q}, \boldsymbol{Q'}$ be sets of variables such that neither is a subset of the other. Then, if T is both $\boldsymbol{Q}$-deterministic and $\boldsymbol{Q'}$-deterministic, but not $(\boldsymbol{Q} \cap \boldsymbol{Q'})$-deterministic, it cannot have full support, i.e. $supp(T) \subset val(\phi(T))$.*

Proposition 5.3 says that, if we want $\psi(m)$ to contain two sets $\boldsymbol{Q}, \boldsymbol{Q'}$ which are not subsets of each other, then this necessarily restricts the support of any sum node T corresponding to m. As a simple example that shows the idea behind this result, suppose that $\boldsymbol{Q} = \{A\}$ and $\boldsymbol{Q'} = \{B\}$, where A, B are binary variables, and that T has two children. If T is both $\boldsymbol{Q}$-deterministic and $\boldsymbol{Q'}$-deterministic, then the children must correspond to different values of A (say $0, 1$ respectively), and to different values of B (say $1, 0$ respectively). But then neither child has support over $(A, B) = (0, 0)$.

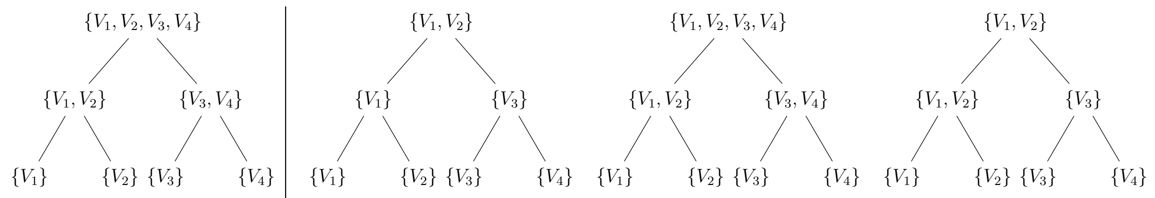
While it can be beneficial to enforce a restricted support on a PC if we have prior knowledge [89], it is undesirable in our case where restricting support comes as a *side effect* of enforcing tractability, as this can result in bias when learning. As such, we only consider labellings $\psi(m)$ where, for every $\boldsymbol{Q}, \boldsymbol{Q'} \in \psi(m)$, we have $\boldsymbol{Q} \subseteq \boldsymbol{Q'}$ or $\boldsymbol{Q'} \subseteq \boldsymbol{Q}$. We can further restrict possible labellings using the following result:

Proposition 5.4 (Superset $\boldsymbol{Q}$-Determinisms for Sum Nodes)**.** *Suppose that a sum node T is $\boldsymbol{Q}$-deterministic. Then it is also $\boldsymbol{Q'}$-deterministic for any $\boldsymbol{Q} \subseteq \boldsymbol{Q'} \subseteq \boldsymbol{V}$.*

Using Proposition 5.4, it now follows that $\psi(m)$ must take the form $\{\boldsymbol{Q'} | \boldsymbol{Q} \subseteq \boldsymbol{Q'} \subseteq \boldsymbol{V}\}$ for some $\boldsymbol{Q}$. As a result, we can just label our vtree node m with a single set $\boldsymbol{Q}$, i.e. $\psi(m) = \boldsymbol{Q}$.

To summarize, we have shown that for structured decomposable circuits, the marginal determinism properties of the circuit can be effectively described using a

[5]A sum node is non-trivial if it has more than one child.



(a) vtree with scope function ϕ (b) vtree with label $\psi^{(\text{str})}$ (c) vtree with label $\psi^{(\text{det})}$ (d) Regular label $\psi^{(\text{opt})}$ for $\psi^{(\text{str})}$

Figure 5.5: Example of md-vtree for variables $\boldsymbol{V} = \{V_1, V_2, V_3, V_4\}$. In the leftmost subfigure, we show each node m with its scope $\phi(m)$, whilst in each of the three other subfigures, we show each node with its label $\psi(m)$ (for each of three different labelling functions).

labelling function ψ, which assigns a set of variables to each vtree node. Since a vtree can have at most $|\boldsymbol{V}|$ nodes, and each labelling is a subset of $\boldsymbol{V}$, the number of possible labelling functions is bounded by $2^{|\boldsymbol{V}|^2}$. This is a significant simplification over the number of possible md-signature, which was doubly exponential in $|\boldsymbol{V}|$.

5.2.1.3 Md-vtrees

We now formally characterize structured marginal determinism using the concept of a *md-vtree*, which provides a means of specifying the support properties that a structured decomposable circuit satisfies.

Definition 5.10 (md-vtree)**.** *A md-vtree $w = (v, \psi)$ for a set of variables $\boldsymbol{V}$ consists of a vtree $v = (M, E)$ over $\boldsymbol{V}$, together with a labelling function ψ.*

The labelling function maps a vtree node $m \in M$ to some element in $\mathcal{P}(\phi(m)) \cup \{U\}$, where U is the universal set.[6]

Definition 5.11 (PC respecting md-vtree)**.** *Let $\mathcal{C}$ be a PC and $w = (v, \psi)$ be a md-vtree, both over variables $\boldsymbol{V}$. Then we say that $\mathcal{C}$ respects w if 1) $\mathcal{C}$ respects v; and 2) for any sum unit $T \in \mathcal{C}$, T is marginally deterministic with respect to $\psi(m)$, where m is the vtree node such that $\phi(T) = \phi(m)$.*

We denote the class of circuits respecting w by $\boldsymbol{\mathcal{C}}_w$.

Intuitively, md-vtrees capture both structured decomposability, as well as a marginal determinism "pattern" that the circuit must follow.

[6]The universal set satisfies, for any set S, $U \supseteq S$, $U \not\subseteq S$ (unless S is U), $U \cap S = S$, and $U \cup S = U$.

Definition 5.12 (Implied $\boldsymbol{Q}$-Determinisms)**.** *For any set $\boldsymbol{Q} \subseteq \boldsymbol{V}$, we say that $\boldsymbol{Q}$-determinism is implied by a md-vtree w if, for every vtree node $m \in M$ such that $\phi(m) \cap \boldsymbol{Q} \neq \emptyset$, it is the case that $\boldsymbol{Q} \supseteq \psi(m)$. We write $\mathcal{Q}(w)$ to denote the set of all sets $\boldsymbol{Q}$ s.t. $\boldsymbol{Q}$-determinism is implied by w.*

Proposition 5.5 (Validity of Implied $\boldsymbol{Q}$-Determinisms)**.** *For any PC $\mathcal{C}$ respecting md-vtree w, we have that $\mathcal{Q}(w) \subseteq \mathcal{Q}(\mathcal{C})$.*

Notice that there is a trade-off between the generality of the PC class, and the support properties it supports. Increasing the size of the labelling sets will improve the former, but hurt the latter.

Theorem 5.2 (Generality-Tractability Tradeoff)**.** *Let $w = (v, \psi)$ and $w' = (v, \psi')$ be two md-vtrees, such that $\psi'(m) \supseteq \psi(m)$ for all $m \in M$. Then we have that $\mathcal{Q}(w) \supseteq \mathcal{Q}(w')$, and $\boldsymbol{\mathcal{C}}_w \subseteq \boldsymbol{\mathcal{C}}_{w'}$.*

It is worth commenting on the two extremes of possible labels; namely, the universal set, and the empty set. The role of the universal set label U is to indicate that no $\boldsymbol{Q}$-determinism properties hold (including normal determinism). On the other hand, the empty set indicates that any sum node T corresponding to m can only have one child N_i which is not zero, i.e. $p_{N_i} \equiv 0$; thus the sum node must be trivial. In practice, this means that it must represent a factorized distribution with factors corresponding to the scopes of the children of m.

5.2.1.4 Support Properties and Md-vtrees

Thus far in this thesis, we have seen three different types of support properties; namely, determinism, the decision property, and marginal determinism. Another important support property is *strong determinism* [44, 89, 47], which is enforced in the probabilistic sentential decision diagram (PSDD) [89], arguably the most well-known implementation of structured decomposable PCs. We now show how all of these properties can be captured in the md-vtree framework. For structured decomposable circuits, these properties can be formulated as follows:

Definition 5.13 (Right-linear vtree)**.** *A vtree v is right-linear if, for every non-leaf vtree node m, the first (left) child m_1 of m is a leaf node.*

Definition 5.14 (Determinism, Strong Determinism, and Decision Properties)**.** *Let $\mathcal{C}$ be a structured decomposable and smooth PC respecting vtree v. Let T be a sum node, with children $\mathrm{ch}(T) = \{N_1, ..., N_K\}$, corresponding to a vtree node m with children $\mathrm{ch}(m) = (m_1, m_2)$. Then, we define the following properties:*

- ***Determinism:*** *For all sum nodes T, $supp(N_i) \cap supp(N_j) = \emptyset$ for all $i \neq j$;*
- ***Strong Determinism:*** *For all sum nodes T, $supp_{\phi(m_1)}(N_i) \cap supp_{\phi(m_1)}(N_j) = \emptyset$ for all $i \neq j$;*
- ***Decision:*** *The vtree v is right-linear and strong determinism holds.*

These properties impose increasingly stringent requirements on the circuit. Determinism specifies that the children of a sum node T must correspond to different values of $\phi(T) = \phi(m)$. Strong determinism specifies that the children must correspond to different values of $\phi(m_1)$. The decision property is a special case of strong determinism, where, due to the assumed right-linear vtree structure, $\phi(m_1)$ is always a single variable. We can express these conditions as special cases of marginal determinism of a sum node (Definition 5.5); determinism corresponds to $\phi(m)$-determinism, while strong determinism/decision correspond to $\phi(m_1)$-determinism. The corresponding md-vtree (Definition 5.10) for determinism has the labelling function $\psi^{(\mathrm{det})}(m) = \phi(m)$, while for strong determinism we have $\psi^{(\mathrm{det})}(m) = \phi(m_1)$. We call these labelling functions deterministic/strongly deterministic labelling functions respectively, and any md-vtree with the labelling function as deterministic/strongly deterministic md-vtres respectively.

For example, for the vtree over $\boldsymbol{V} = \{V_1, V_2, V_3, V_4\}$, shown in Figure 5.5a together with the scopes for each vtree node, the label function corresponding to determinism $\psi^{(\mathrm{det})}$ is shown in Figure 5.5c, while the label function $\psi^{(\mathrm{str})}$ corresponding to strong determinism is given in Figure 5.5b.

One might ask, what are the advantages of enforcing strong determinism or the decision property over determinism? To analyse this in the md-vtree framework, we compare the labelling function $\psi^{(\mathrm{str})}, \psi^{(\mathrm{det})}$ in the example in Figure 5.5. With these representations, we can deduce the $\boldsymbol{Q}$-determinism properties that any structured decomposable and strongly deterministic, or structured decomposable and deterministic circuit, must satisfy, by finding the set $\mathcal{Q}(w)$ of sets $\boldsymbol{Q}$ which its md-vtree implies. In this example, by enumerating all sets $\boldsymbol{Q} \subseteq \boldsymbol{V}$ and checking the condition, we can see that $\mathcal{Q}(w^{\mathrm{str}}) = \{\{V_1, V_2\}, \{V_1, V_2, V_3\}, \{V_1, V_2, V_3, V_4\}\}$, while $\mathcal{Q}(v^{\mathrm{det}}) = \{\{V_1, V_2, V_3, V_4\}\}$. This shows that enforcing strong determinism leads to additional implied $\boldsymbol{Q}$-determinisms, which means, for example, that strongly deterministic circuits are more tractable with regards to MMAP queries. In fact, in the particular case of strong determinism, the implied $\boldsymbol{Q}$-determinisms $\mathcal{Q}(w^{\mathrm{str}})$ coincide with the definition of $\boldsymbol{Q}$-constrained vtrees [125].

We have now seen that determinism, the decision property and strong determinism can be captured using particular labelling functions; however, the md-vtree framework allows for the specification of many other possible labelling functions, which have not been explored to date. In the next section, we will analyze these possible choices of labelling functions in full generality.

5.2.2 Regular Md-vtrees and Enforcing Marginal Determinism

Given the trade-off between generality and tractability for md-vtrees, we might ask how to choose the labelling function to achieve a good balance. At the very least, any labelling function should not be dominated by another labelling function that is more general, yet achieves the same tractability in terms of marginal determinisms. We capture this in the notion of *admissibility*:

Definition 5.15 (Admissibility of Labelling Function)**.** *Given a vtree $v = (M, E)$, a labelling function ψ is inadmissible if there exists a labelling function ψ' such that (i) $\psi'(m) \supseteq \psi(m)$ for all $m \in M$, (ii) $\exists m^* \in M$ such that $\psi'(m) \supset \psi(m)$, and (iii) $\mathcal{Q}(w') \supseteq \mathcal{Q}(w)$, where $w = (v, \psi), w' = (v, \psi')$.*

The number of possible labelling functions for a vtree over variables $\boldsymbol{V}$ is strongly exponential in $|\boldsymbol{V}|$, as we can independently choose a subset of $\boldsymbol{V}$ (specifically, of

$\phi(m)$) at every vtree node m. It turns out, however, that the number of admissible labelling functions is much smaller.

Definition 5.16 (Regular Labelling Function)**.** *Given a vtree v, a labelling function ψ is regular if for every non-leaf node m, and its children m_1, m_2, it holds that either $\psi(m) = \emptyset$, $\psi(m) = \psi(m_1)$, $\psi(m) = \psi(m_2)$, or $\psi(m) = \psi(m_1) \cup \psi(m_2)$. In this case, we will also say that the md-vtree $w = (v, \psi)$ is regular.*

Theorem 5.3 (Admissible Labelling Functions are Regular)**.** *Given a vtree v, let ψ be any non-regular labelling function. Then there exists a regular labelling function ψ' such that (i) $\psi'(m) \supseteq \psi(m)$ for all $m \in M$, (ii) $\exists m^* \in M$ such that $\psi'(m) \supset \psi(m)$, and $\mathcal{Q}(w') = \mathcal{Q}(w)$, where $w := (v, \psi), w' := (v, \psi')$.*

This Theorem, which we prove constructively in Appendix B.1, implies that any admissible labelling function must also be regular. This means that we can restrict our attention to the much smaller space of regular md-vtrees. In particular, the labelling function of a regular md-vtree is entirely determined by the labelling of each leaf node $\psi(m_{\text{leaf}})$, and a quaternary variable over values $\{f, s, b, n\}$ for each non-leaf node, indicating whether the label depends on the label of the first child, second child, both, or neither. Thus, the number of labelling functions we need to consider is now only exponential in the number of variables $|\boldsymbol{V}|$, which is again a massive simplification from general labelling functions. In Section 5.3, we will further analyze the properties of regular md-vtrees, and show how to design PC architectures for them.

Now, we have shown that only regular labelling functions can be admissible, but choosing a regular labelling function still involves a tradeoff between generality and tractability. In practice, a common scenario is that we will want to choose the largest class of circuits that satisfies the constraints that we need in order to efficiently compute an inference query of interest (e.g. MMAP). To formalize this, suppose that we are given a set of required marginal determinisms $\boldsymbol{S}$, such that we require that the md-vtree imply $\boldsymbol{Q}$-determinism for each $\boldsymbol{Q} \in \boldsymbol{S}$. Suppose also that we are given a vtree v, and need to choose the labelling function ψ. This means for each vtree node m, and every $\boldsymbol{Q} \in \boldsymbol{S}$ such that $\phi(m) \cap \boldsymbol{Q} \neq \emptyset$, we must have $\boldsymbol{Q} \supseteq \psi(m)$. Given the result of Theorem 5.2, we should set the label $\psi(m)$ to be the largest set that fulfils this condition. This can be done directly as in the following definition:

Definition 5.17 ($\boldsymbol{S}$-constrained Label)**.** *Given a set $\boldsymbol{S}$ of subsets $\boldsymbol{Q} \subseteq \boldsymbol{V}$, and a vtree v, the $\boldsymbol{S}$-constrained labelling function $\psi_{\boldsymbol{S}}$ is defined by:*

$$\psi_{\boldsymbol{S}}(m) = \begin{cases} U & \text{if } \boldsymbol{Q} \cap \phi(m) = \emptyset \ \ \forall \boldsymbol{Q} \in \boldsymbol{S} \\ \phi(m) \cap \left(\bigcap_{\boldsymbol{Q} \in \boldsymbol{S}: \boldsymbol{Q} \cap \phi(m) \neq \emptyset} \boldsymbol{Q}\right) & \text{otherwise} \end{cases} \tag{5.7}$$

Proposition 5.6 (Correctness and Admissibility of $\boldsymbol{S}$-constrained Label)**.** *For any vtree v and set of marginal determinisms $\boldsymbol{S}$, the md-vtree $w := (v, \psi_{\boldsymbol{S}})$ satisfies $\mathcal{Q}(w) \supseteq \boldsymbol{S}$. Further, $\psi_{\boldsymbol{S}}$ is admissible.*

This proposition says that $\boldsymbol{S}$-constrained labelling function is *correct* in the sense that it does imply the required marginal determinisms, and is also admissible, meaning that it is optimal in terms of generality among all labelling functions which imply the required determinisms. It follows by Theorem 5.3 that the labelling is also regular; this can be checked by directly inspecting the definition.

While $\boldsymbol{S}$-constrained labels are always admissible for a given vtree, both the expressivity and succinctness of the resulting circuit class may differ depending on the vtree. For example, for some vtrees, the $\boldsymbol{S}$-constrained labelling function may induce an empty label $\psi(m)$ for some vtree nodes m, i.e. a factorized distribution, limiting expressivity. That is, though md-vtrees give us the power to separate scope ϕ from support ψ, there still exists a certain scope configurations (vtrees) that are more amenable to certain marginal determinisms.

5.2.2.1 Non-Admissibility of Strong Determinism

With the theory of admissibility and regularity in md-vtrees in hand, we now return to the example from Figure 5.5. It can be easily checked that the corresponding md-vtree $w^{(\text{str})}$ in Figure 5.5b is not regular, and so there must be a regular md-vtree (with the same vtree) that is more general. We therefore construct in Figure 5.5d an regular md-vtree $w^{(\text{opt})}$ that retains the same $\boldsymbol{Q}$-determinisms $\boldsymbol{S} := \mathcal{Q}(w^{(\text{str})}) = \{\{V_1, V_2\}, \{V_1, V_2, V_3\}, \{V_1, V_2, V_3, V_4\}\}$, but with the admissible $\boldsymbol{S}$-constrained labelling. The labels for each vtree node are the same as for strong determinism, except that the label for the vtree node m with scope $\{V_1, V_2\}$ is $\psi^{(\mathrm{opt})}(m) = \{V_1, V_2\}$, instead of $\psi^{(\mathrm{str})}(m) = \{V_1\}$ for strong determinism. By Theorem 5.2, we can conclude that the resulting md-vtree $w^{(\mathrm{opt})}$ is more general than $w^{(\mathrm{str})}$, i.e. admits a larger class of circuits. In other words, strong determinism imposes more constraints than it "needs to" in order to obtain its marginal determinism properties.

5.2.3 Succinctness: Exponential Separation

To summarize the results so far, we have seen that md-vtrees provide a general, structural specification of support properties, with determinism and strong determinism corresponding to specific choices of the labelling function. One key insight from this more general formulation is that we can enforce arbitrary sets of $\boldsymbol{Q}$-determinisms onto a given vtree, which correspond to new labelling functions which were not possible with determinism or strong determinism. We now ask a slightly different question, inspired by the observation that labelling functions for strong determinism are non-regular: are classes of circuits corresponding to regular md-vtrees also more succinct?

For example, we saw at the start of this chapter that the symbolic conditioning, marginal MAP and backdoor queries are tractable given a particular (single) $\boldsymbol{Q}$-determinism. This can already be implemented using existing techniques by choosing the vtree appropriately and enforcing strong determinism [125]. However, as this is a very restrictive subclass of circuits, it may not be possible to represent some functions/probability distributions as succinctly as if we were allowed to use a larger class of circuits. This is important as the computational cost of inference algorithms on PCs depends on the size of the circuit. We have seen that strong determinism is not, in general, regular, and therefore we might hope to uncover a larger class of circuits that also respect the required $\boldsymbol{Q}$-determinism. The question is, how much more succinctly can this larger class of circuits represent functions/distributions? We begin by formally defining important subclasses of circuits:

Definition 5.18 (Circuit Subclasses). *Given a condition c, we define $\boldsymbol{\mathcal{C}}_c$ to be the class of all circuits respecting md-vtrees that satisfy c, i.e. $\boldsymbol{\mathcal{C}}_c = \bigcup_{w \models c} \boldsymbol{\mathcal{C}}_w$. The conditions are defined as follows:*

- ***Strong Determinism**: We use the suffix $_{str}$ to denote md-vtrees $w = (v, \psi)$ where for each vtree node m and its children m_1, m_2, we have $\psi(m) = \phi(m_1)$.*

- ***Vtree**: We use the suffix $_v$ to denote md-vtrees $w = (v, \psi)$ with vtree v.*

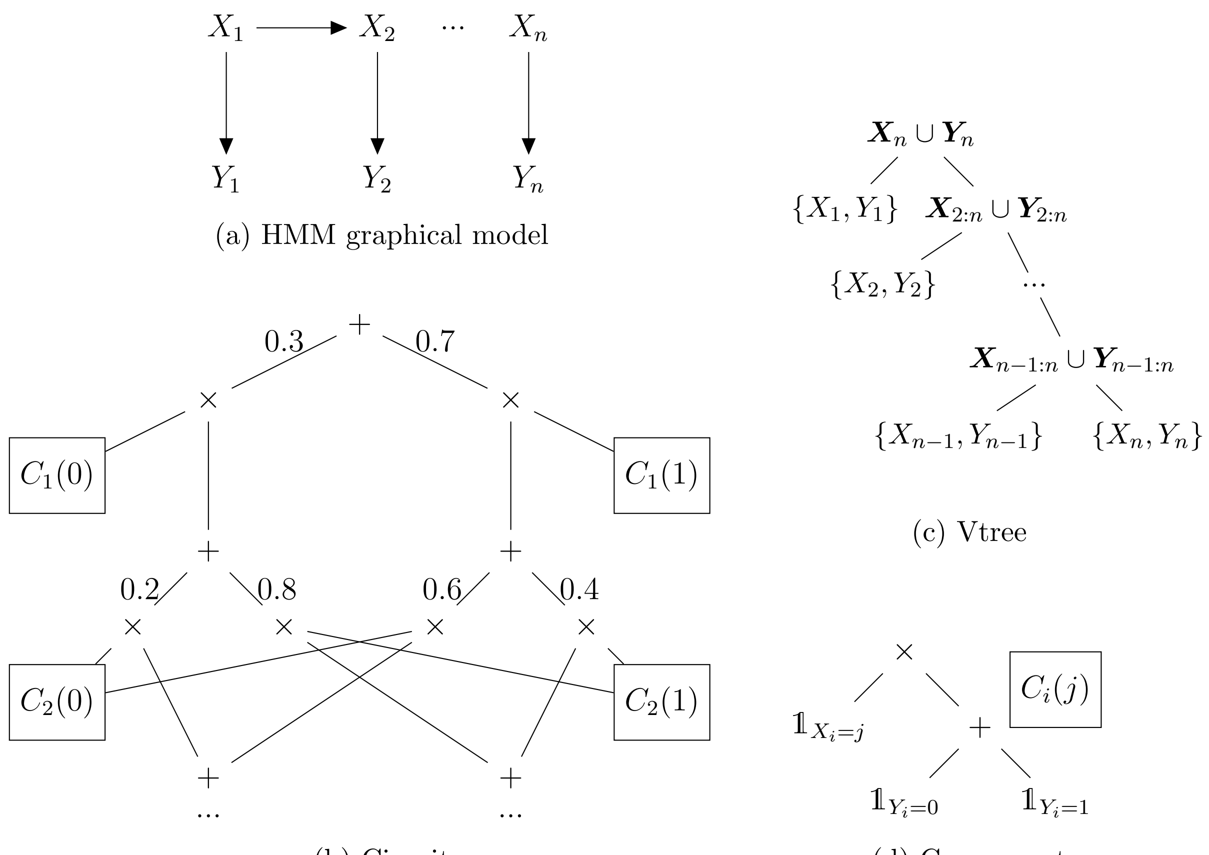
- ***Marginal Determinisms**: We use the suffix $_{\boldsymbol{S}}$ to denote md-vtrees $w = (v, \psi)$ which imply all marginal determinisms in $\boldsymbol{S}$, i.e. $\mathcal{Q}(w) \supseteq \boldsymbol{S}$.*

For example, $\boldsymbol{\mathcal{C}}_{v,\text{str}}$ is the class of all circuits respecting some md-vtree $w = (v, \psi^{(str)})$, where $\psi^{(str)}$ is a labelling function for v corresponding to strong determinism. Now, we need a means of comparing the succinctness of different circuit classes. Informally, a class of circuits $\boldsymbol{\mathcal{C}}'$ is at least as succinct as another class of circuits $\boldsymbol{\mathcal{C}}$, if the function $p_{\mathcal{C}}$ computed by any circuit $\mathcal{C} \in \boldsymbol{\mathcal{C}}$ is also computed by a circuit $\mathcal{C}' \in \boldsymbol{\mathcal{C}}$ with size at most polynomial in the size of $\mathcal{C}$ [69, 50]. In our case, for reasons which will become apparent shortly, we use a slightly different definition that compares two sequences of circuit classes:

Definition 5.19 (Succinctness)**.** *Let $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}}$ and $(\boldsymbol{\mathcal{C}}'_n)_{n\in\mathbb{N}}$ be two sequences of circuit classes. Then we say $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}}$ is at least as succinct as $(\boldsymbol{\mathcal{C}}'_n)_{n\in\mathbb{N}}$, written $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}} \leq (\boldsymbol{\mathcal{C}}'_n)_{n\in\mathbb{N}}$ if there exists a polynomial q such that for all $n \in \mathbb{N}$ and $\mathcal{C} \in \boldsymbol{\mathcal{C}}_n$, there exists $\mathcal{C}' \in \boldsymbol{\mathcal{C}}'_n$ computing the same function, with $|\mathcal{C}'| \leq q(|\mathcal{C}|)$.*

We will also say that a sequence of circuit classes is strictly more succinct than another sequence, written $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}} < (\boldsymbol{\mathcal{C}}'_n)_{n\in\mathbb{N}}$, if it is the case that $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}} \leq (\boldsymbol{\mathcal{C}}'_n)_{n\in\mathbb{N}}$, but $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}} \not\geq (\boldsymbol{\mathcal{C}}'_n)_{n\in\mathbb{N}}$. For example, it is known that $(\boldsymbol{\mathcal{C}}_n)_{n\in\mathbb{N}} < (\boldsymbol{\mathcal{C}}_{n,\text{det}})_{n\in\mathbb{N}}$, where $\boldsymbol{\mathcal{C}}_n$ is the class of all n-variable decomposable and smooth circuits and $\boldsymbol{\mathcal{C}}_{n,str}$ is the class of all n-variable decomposable, smooth and deterministic circuits [31].

Using this notion of succinctness, there are several ways in which we can compare the succinctness of strongly deterministic md-vtrees as compared to general md-vtrees. One is to compare the succinctness of subclasses of circuits satisfying a single marginal determinism $\boldsymbol{Q}$; this is practically important for symbolic conditioning on $\boldsymbol{Q}$, for instance. Another is to compare the relative succinctness for the same vtree, in order to isolate the effect of the labelling function. The following result states that in



(a) HMM graphical model

(b) Circuit

(c) Vtree

(d) Component

Figure 5.6: Illustration of PC computing hidden Markov model (HMM)

both of these cases, the general md-vtree class is strictly more succinct than strongly deterministic md-vtrees.

Theorem 5.4 (Exponential Separation)**.** *The following separation results hold:*

- $\exists$ *a sequence* $(\boldsymbol{Q}_n)_{n\in\mathbb{N}}$ *such that* $(\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\}})_{n\in\mathbb{N}} < (\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\},str})_{n\in\mathbb{N}}$
- $\exists$ *a sequence* $(v_n)_{n\in\mathbb{N}}$ *such that* $(\boldsymbol{\mathcal{C}}_{v_n})_{n\in\mathbb{N}} < (\boldsymbol{\mathcal{C}}_{v_n,str})_{n\in\mathbb{N}}$

Proof. Firstly, note that $(\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\}})_{n\in\mathbb{N}} \leq (\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\},\text{str}})_{n\in\mathbb{N}}$ trivially holds for any sequence $(\boldsymbol{Q}_n)_{n\in\mathbb{N}}$, as $\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\}} \supseteq \boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\},\text{str}}$. Similarly, $(\boldsymbol{\mathcal{C}}_{v_n})_{n\in\mathbb{N}} \supseteq (\boldsymbol{\mathcal{C}}_{v_n,\text{str}})_{n\in\mathbb{N}}$ for any sequence $(v_n)_{n\in\mathbb{N}}$. It remains to show strictness of the succinctness relation, that is, that $\exists$ a sequence $(\boldsymbol{Q}_n)_{n\in\mathbb{N}}$ such that $(\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\},\text{str}})_{n\in\mathbb{N}} \not\leq (\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\}})_{n\in\mathbb{N}}$, and that $\exists$ a sequence $(v_n)_{n\in\mathbb{N}}$ such that $(\boldsymbol{\mathcal{C}}_{v_n,\text{str}})_{n\in\mathbb{N}} \not\leq (\boldsymbol{\mathcal{C}}_{v_n})_{n\in\mathbb{N}}$.

We can show this by constructing a distribution (function) that can be computed by a polynomially sized circuit in the class corresponding to general md-vtrees, but for which the smallest circuit in the class corresponding to strongly deterministic md-vtrees is exponentially sized. Each of these examples will also provide insights into how marginal determinism can be constructed through using labellings other than strong determinism labelling functions.

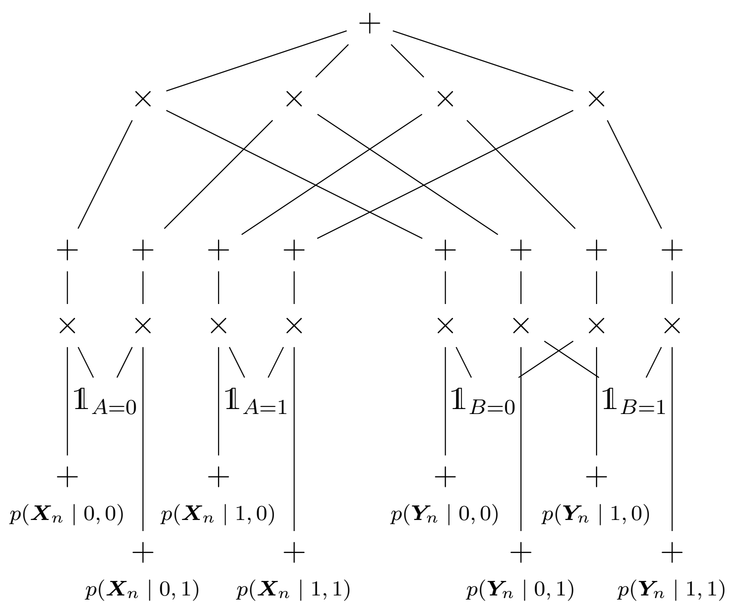
We begin with the first result. In this case, we consider representing the distribution given by a hidden Markov model (HMM) over (hidden) variables $\boldsymbol{X}_n = \{X_1, ..., X_n\}$ and (observed) variables $\boldsymbol{Y}_n = \{Y_1, ..., Y_n\}$, as depicted in Figure 5.6a. We take the required marginal determinism to be $\boldsymbol{Q}_n := \boldsymbol{X}_n$. If we have a PC computing $p(\boldsymbol{X}_n, \boldsymbol{Y}_n)$ that is marginally deterministic w.r.t. $\boldsymbol{Q}_n$, then it becomes tractable to obtain a circuit representing $p(\boldsymbol{Y}_n|\boldsymbol{X}_n)$, or compute the marginal MAP $\max_{\boldsymbol{X}_n} p(\boldsymbol{X}_n)$ over the hidden variables. The latter task in particular is a well-known task that can be solved by the classical Viterbi algorithm based on dynamic programming (DP). In this case applying the marginal MAP algorithm to a $\boldsymbol{Q}_n$-deterministic circuit would effectively execute this DP algorithm.

Figure 5.6b shows a structured decomposable circuit that computes the hidden Markov model distribution, where the components $C_i(j)$ have scope $\{X_i, Y_i\}$. The corresponding vtree (with nodes notated using their scopes) is shown in Figure 5.6c.

By applying the $\boldsymbol{S}$-constrained label with $\boldsymbol{S} = \{\boldsymbol{X}_n\}$, we obtain a labelling function for the vtree such that the resulting mdvtree w implies $\boldsymbol{X}_n$-determinism. This labelling function is given by $\psi_{\boldsymbol{S}}(m) = \phi(m) \cap \boldsymbol{X}_n$, or, in other words, imposes the condition that the children of sum nodes correspond to distinct values of the X variables in the scope of the sum node. This holds in the example circuit, as the components $C_i(j)$ for fixed i and distinct j correspond to different values of X_i. Thus, this circuit is in $\boldsymbol{\mathcal{C}}_{\{\boldsymbol{Q}_n\}}$. Further, the circuit size is clearly linear in n.

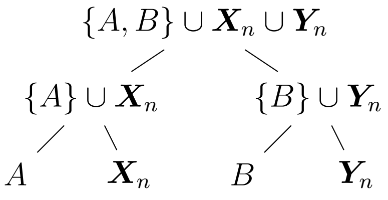
It remains to show that for any strongly deterministic md-vtree that implies $\boldsymbol{Q}_n$-determinism, the smallest circuit respecting that md-vtree and computing the HMM distribution is exponential in size. Explicitly, we will choose a HMM distribution with $p(Y_i|X_i) = \mathbb{1}_{Y_i=X_i}$. Let $w^{(\mathrm{str})}$ be a strongly deterministic md-vtree that implies $\boldsymbol{X}_n$-determinism. By the definition of strong determinism, for any vtree node m with $\phi(m) \cap \boldsymbol{X}_n \neq \emptyset$, we must have $\boldsymbol{X}_n \supseteq \psi(m) = \phi(m_1)$. In other words, if a vtree node contains any X variables in its scope, then no $\boldsymbol{Y}$ variables can be contained in the scope of its first child m_1. Let $m_{2,1}$ denote the root vtree node with scope $\phi(m_{2,1}) = \boldsymbol{X}_n \cup \boldsymbol{Y}_n$, and the sequence $m_{2,1}, ..., m_{2,K}$ be the sequence of nodes obtained by starting at the root vtree node and successively taking the second child of each vtree node. As only X variables can be "removed" while the scope of the vtree node contains any X variables, there must be some node in this sequence, say $m_{2,k}$, with scope $\phi(m_{2,k}) = \boldsymbol{Y}_n$. Now, let circuit $\mathcal{C}$ be a circuit respecting the md-vtree $w^{(\mathrm{str})}$, and computing the HMM distribution. Let $\boldsymbol{T}_k$ be the set of sum nodes with scope $\boldsymbol{Y}_n$ in the circuit[7]. Each node $T_{k,j} \in \boldsymbol{T}_k$ represents a distribution $p_{T_{k,j}}(\boldsymbol{Y}_n)$ over $\boldsymbol{Y}_n$. Now, for any value $\boldsymbol{x}_n$ of $\boldsymbol{X}_n$, consider the distribution $p(\boldsymbol{x}_n, \boldsymbol{Y}_n) = p_{\mathcal{C}}(\boldsymbol{x}_n, \boldsymbol{Y}_n) = p_{\mathcal{C}}(\boldsymbol{x}_n)\mathbb{1}_{\boldsymbol{Y}_n=\boldsymbol{x}_n}$. By the semantics of PCs, this distribution will be given by a weighted sum $p_{\mathcal{C}}(\boldsymbol{x}_n, \boldsymbol{Y}_n) = \sum_{T_{k,j} \in \boldsymbol{T}_k} c_j p_{T_{k,j}}(\boldsymbol{Y}_n)$ where c_j are some non-negative scalars (determined by the rest of the circuit). This means that it must be possible to construct $\mathbb{1}_{\boldsymbol{Y}_n=\boldsymbol{x}_n}$ (up to a constant) using a weighted sum of the sum nodes $\boldsymbol{T}_k$. Further, this must be possible for all values of $\boldsymbol{x}_n \in \{0,1\}^n$. This implies that the set $\boldsymbol{T}_k$ must contain at least 2^n sum nodes, and as such the circuit has exponential size in n.

[7]In general, for strongly deterministic circuits, these represent the conditional distribution of $\boldsymbol{Y}_n$ given some (logical formula over) $\boldsymbol{X}_n$; we do not explicitly prove these semantics here, but the interested reader can refer to [89] for more details.



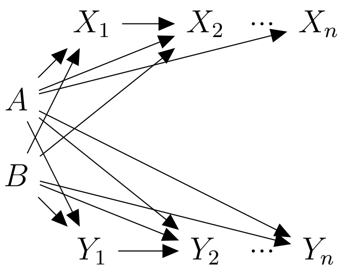
+
×
$\mathbb{1}_{A=0}$
$\mathbb{1}_{A=1}$
$\mathbb{1}_{B=0}$
$\mathbb{1}_{B=1}$
$p(\boldsymbol{X}_n \mid 0,0)$
$p(\boldsymbol{X}_n \mid 1,0)$
$p(\boldsymbol{Y}_n \mid 0,0)$
$p(\boldsymbol{Y}_n \mid 1,0)$
$p(\boldsymbol{X}_n \mid 0,1)$
$p(\boldsymbol{X}_n \mid 1,1)$
$p(\boldsymbol{Y}_n \mid 0,1)$
$p(\boldsymbol{Y}_n \mid 1,1)$

(a) Circuit



$\{A, B\} \cup \boldsymbol{X}_n \cup \boldsymbol{Y}_n$
$\{A\} \cup \boldsymbol{X}_n$
$\{B\} \cup \boldsymbol{Y}_n$
A
$\boldsymbol{X}_n$
B
$\boldsymbol{Y}_n$

(b) Vtree



(c) Graphical Model

Figure 5.7: Illustration of PC computing graphical model

It is important to note that we are *not* proving that strongly deterministic circuits (e.g. PSDDs) cannot represent HMMs with $\boldsymbol{Q}_n$-determinism. In fact, this is not true, as the circuit shown in Figure 5.6b is actually a strongly deterministic circuit (i.e. respects w_{str}). Rather, the result states that for any strongly deterministic md-vtrees such that *all* circuits respecting the md-vtree are $\boldsymbol{Q}_n$-deterministic, the smallest circuit respecting that md-vtree and computing the HMM distribution is exponential. This has important implications, for example, for learning: if we were to learn a circuit (e.g. PSDD) respecting w_{str}, there is no guarantee that the resulting circuit would be $\boldsymbol{Q}_n$-deterministic, while such a guarantee does exist for circuits respecting w. This first example shows that even when strongly deterministic md-vtrees can enforce the required marginal determinisms, this may come at the cost of succinctness.

The second result states, informally, that for the same vtree, the class of marginally deterministic circuits is exponentially more succinct than strongly deterministic circuits. Note that unlike the previous result, we do not impose any requirement on marginal determinisms. The graphical model we wish to represent (Figure 5.7c) contains two separate chain graphs, $X_1 \to X_2 \to ... \to X_n$ and $Y_1 \to Y_2 \to ... \to Y_n$, with two confounders A, B. In Figure 5.7b, we show a (partial) vtree over variables A, B,

$\boldsymbol{X}_n = \{X_1, ...X_n\}$ and $\boldsymbol{Y}_n = \{Y_1, ..., Y_n\}$. We also show a (partial) circuit in Figure 5.7a which computes the distribution given by the graphical model, and which respects the given vtree. To represent the graphical model distribution, the root node of the circuit has four children, representing the four different values of (A, B). These are then connected to the sum nodes at the bottom, which compute the conditional distributions of $\boldsymbol{X}_n$ or $\boldsymbol{Y}_n$ given A, B (each of these is just a chain graph distribution, so can be represented with a circuit with linear size in n). This circuit is $\{A, B\}$-deterministic, respecting a md-vtree with labelling function $\psi(m_{\text{root}}) = \{A, B\}$ for the root vtree node. It is also linear in the size of n.

In contrast, for this vtree, the strongly deterministic labelling of the root node is $\psi(m_{\text{root}}) = \{A\} \cup \boldsymbol{X}_n$, giving a md-vtree w_{str}. However, the shown circuit would not respect such a md-vtree. In fact, any circuit respecting w_{str} and computing the distribution of the graphical model is exponential in size. To show this, let $\boldsymbol{P}$ be the set of immediate product node descendants of the root sum node (i.e. product nodes such that all paths between the root and product node contain only sum nodes). By $(\{A\} \cup \boldsymbol{X}_n)$-determinism, all of the product nodes $\{P_j\}_{j=1}^{K}$ must have distinct marginalized support w.r.t $(\{A\} \cup \boldsymbol{X}_n)$. This means that given any value $\{a\} \cup \boldsymbol{x}_n$ of $(\{A\} \cup \boldsymbol{X}_n)$, the corresponding distribution $p_{\mathcal{C}}(\{a\} \cup \boldsymbol{x}_n, \{B\} \cup \boldsymbol{Y}) = c \times p_{P_k}(\{a\} \cup \boldsymbol{x}_n, \{B\} \cup \boldsymbol{Y})$ where c is some non-negative scalar and P_k is a product node in $\boldsymbol{P}$. However, for the graphical model, in general $p(\{a\} \cup \boldsymbol{x}_n, \{B\} \cup \boldsymbol{Y})$ will be different for every value $\{a\} \cup \boldsymbol{x}_n$. As such, there must be a different product node corresponding to each value $\{a\} \cup \boldsymbol{x}_n$, and so the circuit has size at least 2^{n+1}. The reason the circuit in Figure 5.7a achieves linear size is because there we are allowed to have different product nodes with support over the same value $\{a\} \cup \boldsymbol{x}_n$ (this holds for the left two product nodes, and the right two product nodes, in the top product layer).

$\square$

To summarize, we have shown two ways in which general md-vtrees result in more succinct circuits compared with strongly deterministic md-vtrees. In the first case, we saw the benefits of general labelling functions, in that they are decoupled from the scopes of the vtree nodes. This means there are a lot more vtrees using general labelling functions that can imply $\boldsymbol{Q}_n$-determinism, as compared to strongly deterministic labelling functions. In the second case, we saw further how it can be beneficial to select labels $\psi(m)$ for a vtree node that contain variables from the scopes $\phi(m_1), \phi(m_2)$ of *both* the children of m. This enables the given graphical model distribution to be encoded using the given vtree, where it would not be possible for strongly deterministic labellings (or indeed, any labelling which only included variables from the scope of only one of the children of m).

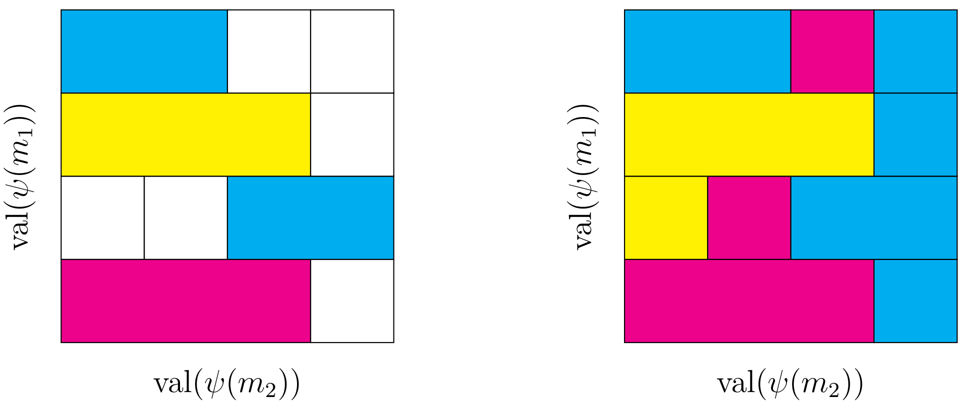
5.3 MDNet Architecture and Learning

In this section, we show how to construct and learn a probabilistic circuit that respects a particular md-vtree. It is worth noting that, as special cases of md-vtrees, we can use existing architectures and learning algorithms for PCs such as PSDDs [103] and structured decomposable and deterministic circuits [41, 53]. However, we have seen that PSDDs are not optimally succinct, and to enforce tractability, we may need to target md-vtrees that do not fall into these existing categories, such as those generated using the Y-constrained label function in Definition 5.17. We thus propose a novel PC architecture, MDNet, which enforces a given regular md-vtree *by design*.

5.3.1 Understanding Regular Md-vtrees and MDNets

Before explaining the architecture of MDNets, we begin by discussing the intuition and motivation behind the chosen architecture. The key desiderata that we set for MDNets is to be able to enforce *arbitrary* md-vtrees and to do so in a *scalable* manner. By scalable, we mean that it should be possible to easily trade off expressivity/complexity of the architecture with the size of the model (and thus inferential complexity) by configuring hyperparameters; this is a key feature of the most successful modern PC architectures [139, 137, 107], drawing inspiration from neural networks.

To better illustrate the nature of regular md-vtrees, we show in Figure 5.8 a partitioning of the values of $\psi(m_1) \cup \psi(m_2)$. Let m be a vtree node with children m_1, m_2 in the md-vtree. Suppose we have sum nodes T_1, T_2 with scopes $\phi(T_1) = \phi(m_1), \phi(T_2) = \phi(m_2)$. We can represent the marginalized support $\mathrm{supp}_{\psi(m_1)}(T_1) \subseteq$



(a) Mixing layer (b) Synthesizing layer

Figure 5.8: Illustration of support partitioning in mixing and synthesizing layers, for a vtree node m and its children m_1, m_2. Vertical/horizontal axes represent different values of $\psi(m_1)/\psi(m_2)$ respectively. Colors represent partitioning of support for sum nodes within a group.

$\text{val}(\psi(m_1)), \text{supp}_{\psi(m_2)}(T_2) \subseteq \text{val}(\psi(m_2))$ as *segments along the vertical/horizontal axis* respectively. Given this, a product node P with children T_1, T_2 will then have marginalized support $\text{supp}_{\psi(m_1)\cup\psi(m_2)}(P)$ given by the Cartesian product of $\text{supp}_{\psi(m_1)}(T_1), \text{supp}_{\psi(m_2)}(T_2)$, represented pictorially by a *rectangle* in the Figure.

Now, recall that regular md-vtrees w have the property that for a vtree node m and its children m_1, m_2, the label of m satisfies $\psi(m) = \emptyset$, $\psi(m) = \psi(m_1)$, $\psi(m) = \psi(m_2)$, or $\psi(m) = \psi(m_1) \cup \psi(m_2)$. Let us consider a sum node T with scope $\phi(T) = \phi(m)$ in a circuit $\mathcal{C}$, and which has a number of product node children $P_1, ..P_n$, each of which in turn has two sum node children $T_{i,1}, T_{i,2}$ with scopes $\phi(m_1), \phi(m_2)$ respectively. In order for $\mathcal{C}$ to respect w, we need T to satisfy $\psi(m)$-determinism. We consider each of the cases of the label separately:

- $\psi(m) = \emptyset$: As we have previously seen, to enforce an empty label we need to ensure that T only has a single child P_1, i.e. represents a factorized distribution. In the Figure, this would correspond to selecting a single rectangle.

- $\psi(m) = \psi(m_1)$ or $\psi(m) = \psi(m_2)$: Without loss of generality, we assume that $\psi(m) = \psi(m_1)$. We need to ensure that the product node children $P_1, ..., P_n$ have distinct marginalized support $\text{supp}_{\psi(m_1)}(P_i)$ with respect to $\psi(m_1)$. This is somewhat similar to *strong determinism* in PSDDs, except that the required marginal determinism is not the scope $\phi(m_1)$ as it is for PSDDs, but rather $\psi(m_1)$ which is a subset of $\phi(m_1)$.

To visualize this, in Figure 5.8a we show a possible partitioning of the values of $\psi(m_1) \cup \psi(m_2)$. In order for T to satisfy the required marginal determinism property, each product node child must have distinct marginalized support w.r.t $\psi(m_1)$; that is, they must cover different values along the vertical axis. For example, T could have four product node children corresponding to each of the colored rectangles, but would not be able to add any other product node without violating $\psi(m_1)$-determinism.

- $\psi(m) = \psi(m_1) \cup \psi(m_2)$: In this case, we need to ensure that the product node children $P_1, ..., P_n$ has distinct marginalized support $\mathrm{supp}_{\psi(m_1)\cup\psi(m_2)}(P_i)$ with respect to $\psi(m_1) \cup \psi(m_2)$.

  To visualize this, in Figure 5.8b we again show the partitioning of the support of $\psi(m_1) \cup \psi(m_2)$. This time, however, the children of T need only correspond to different values of $\psi(m_1) \cup \psi(m_2)$; visually, this means that they need only not overlap. For example, T could have product node children corresponding to each of the rectangles.

The analysis above suggests a strategy for implementing the required marginal determinisms in $\mathcal{C}$, that restricts the product nodes that T is connected to, i.e. selecting an appropriate subset of the rectangles. However, this relies implicitly on the sum nodes $\{T_{i,1}\}_{i=1}^n$ having disjoint marginalized support $\mathrm{supp}_{\psi(m_1)}(T_{i,1})$, and $\{T_{i,2}\}_{i=1}^n$ having disjoint marginalized support $\mathrm{supp}_{\psi(m_2)}(T_{i,2})$, such that the product nodes (rectangles) do not overlap. This motivates our key insight: namely, the concept of a sum node *group*. Intuitively, a group is a set of sum nodes with the same scope, corresponding to some vtree node m, which have disjoint marginalized support w.r.t. $\psi(m)$. By implementing groups corresponding to m_1 and m_2, we can ensure that the products of sum nodes from these groups have no overlap. Importantly, we do not need to know the functions that the sum nodes in the groups represent, or even their (marginalized) support, in order to be able to construct the PC structure.

5.3.2 MDNet Architecture: Definition and Illustration

We are now ready to define the architecture of MDNets. Given a md-vtree m, a MDNet consists of a *layer* of sum nodes for each vtree node m in w. Each layer in turn consists of a set of G_m groups, each of which contain K_m sum nodes. To ensure structured decomposability (that is, that the sum nodes in a layer have scope $\phi(m)$), each sum node in a layer corresponding to m has only product node children, which in turn have two sum node children in layers corresponding to m_1 and m_2, where m_1 and m_2 are the children of m in the vtree.

The key modular component of MDNets is the *node group*, which is a vector of sum nodes with the same scope (i.e. corresponding to the same vtree node m) with the property that the nodes in the group have disjoint marginalized support $\mathrm{supp}_{\psi(m)}(N)$. We use groups as an invariant in order to enforce the required marginal determinisms throughout the circuit. More formally, suppose that we have a non-leaf vtree node m, and let m_1, m_2 be its children. In the layer corresponding to m, we have G_m groups $\boldsymbol{T}_1, ...\boldsymbol{T}_{G_m}$ to the layer for vtree node m, and similarly $\boldsymbol{T}_1^{(1)}, ...\boldsymbol{T}_{G_1}^{(1)}$, $\boldsymbol{T}_1^{(2)}, ...\boldsymbol{T}_{G_2}^{(2)}$ to the layers for m_1, m_2. For regular md-vtrees, the label $\psi(m)$ is either equal to $\emptyset$, or is $\psi(m_1)$ or $\psi(m_2)$, or is their union. We handle these cases separately, as *product*, *mixing* and *synthesizing* layers.

The difference between these layers lies in the restrictions that we place on the connectivity of sum nodes within a group. For each group $\boldsymbol{T}_i = (T_{i,1}, ..., T_{i,K_m})$, and for each sum node $T_{i,k}$ in the group, we assign a vector of product nodes $\boldsymbol{P}_{i,k}$ to $T_{i,k}$. Each product node has two children; the first child being a node from $\boldsymbol{T}_1^{(1)}, ...\boldsymbol{T}_{K_1}^{(1)}$ and the second child being a node from $\boldsymbol{T}_1^{(2)}, ...\boldsymbol{T}_{K_2}^{(2)}$. We write $\boldsymbol{T}_{i,k}^{(1)}$ for the vector of first children of $\boldsymbol{P}_{i,k}$, and $\boldsymbol{T}_{i,k}^{(2)}$ for the vector of second children of $\boldsymbol{P}_{i,k}$.

Product Layer If $\psi(m) = \emptyset$, we implement a *product* layer. Product layers impose the restriction that there is only one sum node in each group, i.e. $|\boldsymbol{T}_i| = 1$, and that sum node has only one product node child, i.e. $|\boldsymbol{P}_{i,1}| = 1$.

Mixing Layer If $\psi(m) = \psi(m_1)$ or $\psi(m_2)$, we implement a *mixing* layer. Without loss of generality, we assume $\psi(m) = \psi(m_1)$. Mixing layers impose the restriction that, for a given group of sum nodes in layer m, all product node children must have *different* sum nodes in the *same* group in layer m_1 as children. More formally, no sum node can appear more than once in $\boldsymbol{T}_{i,1}^{(1)}, ..., \boldsymbol{T}_{i,K_m}^{(1)}$, and the sum nodes must all come from a single group, i.e $\cup_k \boldsymbol{T}_{i,k}^{(1)} \subseteq \boldsymbol{T}_{k'}^{(1)}$ for some k'. We call this group in layer m_1 the *reference group.* Note that there is no restriction on which sum nodes in layer m_2 the product nodes are connected to. This ensures that all of the sum nodes are $\psi(m) = \psi(m_1)$-deterministic, and further that each group $\boldsymbol{T}_i$ satisfies the invariant.

As an example, in Figure 5.8a, we show a single group $\boldsymbol{T}_i$ of a mixing layer, consisting of three sum nodes $T_{i,1}, T_{i,2}, T_{i,3}$. Each sum node corresponds to a distinct color, with each of the colored rectangles corresponding to a product node. Notice that the product nodes all correspond to different values of $\psi(m_1)$, resulting in the sum nodes also having this property.

Synthesizing Layer If $\psi(m) = \psi(m_1) \cup \psi(m_2)$, we implement a *synthesizing* layer. Synthesizing layers also require that, for a given group of sum nodes in layer m, the product node children have a corresponding reference group. However, there are two differences. Firstly, the reference group can be in either layer m_1 or layer m_2, whereas for mixing layers this is determined by the value of $\psi(m)$. Secondly, the restriction that the product nodes must have *different* sum nodes as children (in the reference group) is relaxed. Instead, we impose the restriction that if two product nodes have the same child sum node in the reference group, then the other child of the product nodes must be different sum nodes in the same group.

As an example, in Figure 5.8b, we show a single group $\boldsymbol{T}_i$ of a synthesizing layer, consisting of three sum nodes $T_{i,1}, T_{i,2}, T_{i,3}$. Each sum node corresponds to a distinct color, with each of the colored rectangles corresponding to a product node. Notice that the product nodes all correspond to different values of $\psi(m_1) \cup \psi(m_2)$, resulting in the sum nodes also having this property.

We show explicit examples of each of these types of layers in Figures 5.9-5.11, which consist of three layers corresponding to m, m_1, m_2. We have two sum node groups in layer m, each of which has a set of product node children which implement the

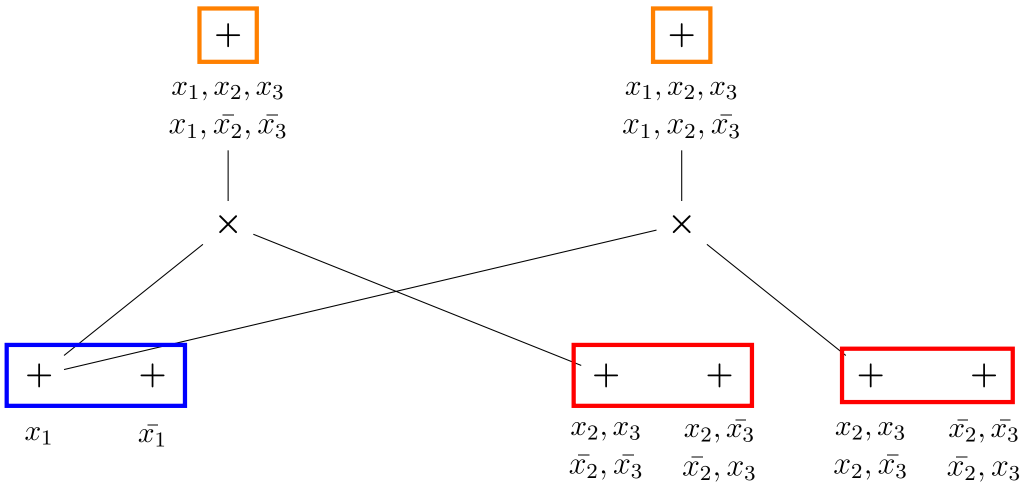


Figure 5.9: Example of product layer. Groups are highlighted in boxes, with the top two groups corresponding to layer m, the bottom left group corresponding to layer m_1, and the bottom right groups corresponding to layer m_2. Sum nodes labelled with their support. Sum node weights omitted for clarity.

restrictions described. As the restrictions become less strenuous, the connectivity of the PC can become more dense. Intuitively, mixing layers achieve their marginal determinism by "copying" the marginal determinism of one of their child layers, while mixing over groups in the other child layer. On the other hand, synthesizing layers enforce marginal determinism by combining, or synthesizing, the marginal determinism properties of both of their children. This allows synthesizing layers to increase the size or "expressivity" of groups, as the number of available product nodes is equal to the product of the group sizes in the child layers.

The scalability of the MDNet architecture comes from the fact that we can control the size of the architecture by choosing the hyperparameters G_m (number of groups) and K_m (number of nodes per group) for each vtree node m. For simplicity, we propose to learn MDNets exploiting recent advancements in random structures for PC learning [137, 139, 53]: in particular, we propose to choose the MDNet structure randomly within the constraints, and then learn the parameters using standard MLE estimation if the md-vtree implies ($\boldsymbol{V}$-)determinism, or use expectation-maximization otherwise [134].

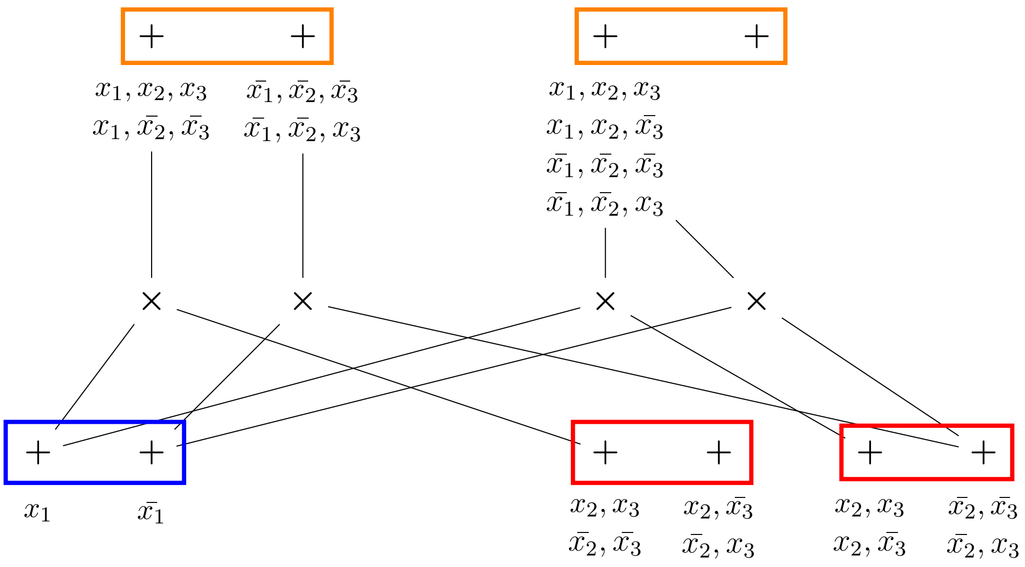


Figure 5.10: Example of mixing layer. Groups are highlighted in boxes, with the top two groups corresponding to layer m, the bottom left group corresponding to layer m_1, and the bottom right groups corresponding to layer m_2. Sum nodes labelled with their support. Sum node weights omitted for clarity.

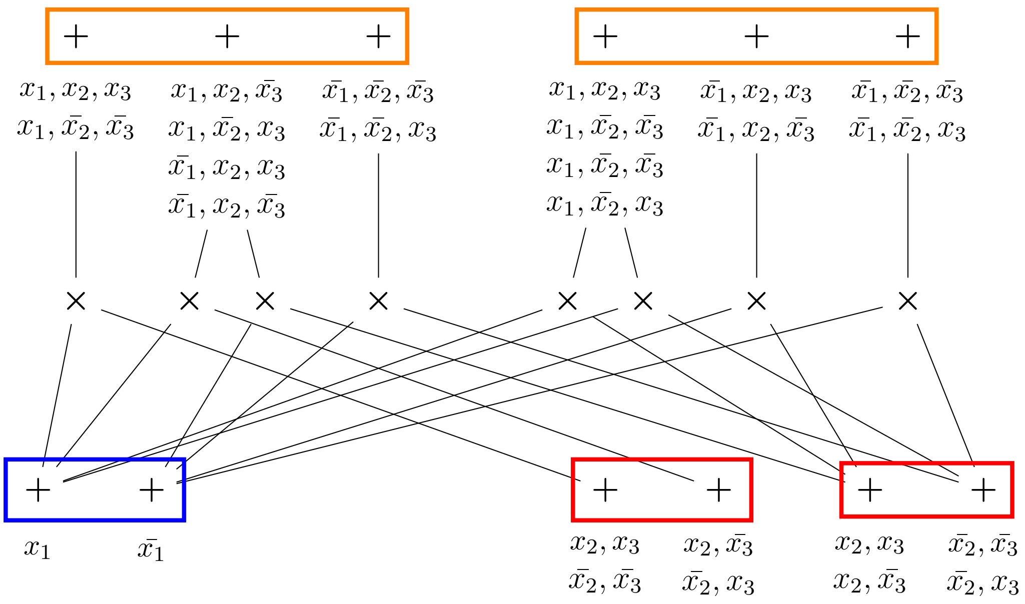


Figure 5.11: Example of synthesizing layer. Groups are highlighted in boxes, with the top two groups corresponding to layer m, the bottom left group corresponding to layer m_1, and the bottom right groups corresponding to layer m_2. Sum nodes labelled with their support. Sum node weights omitted for clarity.

5.4 Compositional Inference using Structured Marginal Determinism

With the theory of structured marginal determinism in hand, we now turn to our original goal of analyzing tractability for *compositions* of operations such as marginalization, products and conditioning. That is, given an (arbitrary) composition of operations, we would like to determine conditions/properties on the input circuit that enable tractable computation, and an algorithm for applying the composition when it is tractable. Compared to analyzing tractability of individual operations, the additional challenge is that we need to consider not only the *input* properties needed for tractability of each operation, but also the properties that the *output* of the operations satisfy. This is particularly problematic for support properties such as ($\boldsymbol{V}$-)determinism, where we have seen that marginalizing out a subset of variables results in determinism being lost.

In this section, we will describe a methodology that exploits our md-vtree framework as a language for deriving tractability conditions for arbitrary compositions of basic operations on probabilistic circuits. We begin by defining the set of basic operations that we work with, and elaborating on the challenges associated with analysing support properties in compositions of these operations. Next, we derive algorithms for transforming md-vtrees through each operation. These algorithms are an abstract implementation of the corresponding algorithms for implementing the operation on circuits, such that the output md-vtree precisely characterizes the properties that the output circuit is guaranteed to respect. Finally, we derive a set of rules called the MD-calculus, which enable us to derive tractability conditions for compositions of operations by propagating marginal determinisms back through the compositional pipeline.

5.4.1 Support Properties in Compositional Inference

In Table 5.1, we define a collection of basic probabilistic inference operations, including marginalization, products, instantiation, conditioning, powers, maximization, and logarithms, along with the properties (*requirements*) under which there exist efficient

Operation	Input Condition	Output Encodes	Complexity
MARG$(\mathcal{C}; \boldsymbol{W})$	-	$\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{V})$	$O(\|\mathcal{C}\|)$
INST$(\mathcal{C}; \boldsymbol{w})$	-	$p_{\mathcal{C}}(\boldsymbol{w}, \boldsymbol{V} \setminus \boldsymbol{W})$	$O(\|\mathcal{C}\|)$
PROD$(\mathcal{C}_1, \mathcal{C}_2)$	Str.Dec. w/ Compatible Vtrees	$p_{\mathcal{C}_1}(\boldsymbol{V}) \times p_{\mathcal{C}_2}(\boldsymbol{V})$	$O(\|\mathcal{C}_1\|\|\mathcal{C}_2\|)$
COND$(\mathcal{C}; \boldsymbol{W})$	$\boldsymbol{W}$-Det	$p_{\mathcal{C}}(\boldsymbol{V} \setminus \boldsymbol{W} \mid \boldsymbol{W})\|_{\text{supp}_{\boldsymbol{W}}(\mathcal{C})}$	$O(\|\mathcal{C}\|)$
POW$(\mathcal{C}; \alpha)$	Det	$p_{\mathcal{C}}(\boldsymbol{V})^{\alpha}\|_{\text{supp}(\mathcal{C})}$	$O(\|\mathcal{C}\|)$
MAX$(\mathcal{C})$	Det	$\max_{\boldsymbol{V}} p_{\mathcal{C}}(\boldsymbol{V})$	$O(\|\mathcal{C}\|)$
LOG$(\mathcal{C})$	Det	$\log p_{\mathcal{C}}(\boldsymbol{V})\|_{\text{supp}(\mathcal{C})}$	$O(\|\mathcal{C}\|)$

Table 5.1: Definitions of basic operations.

(polytime) algorithms for computing them on PCs [185] and the complexity; note that we assume decomposability and smoothness by default. These operations produce a circuit encoding the specified function (or scalar in the case of `MAX`). We use $|$ to denote the *restricted* conditional/power/logarithm, with the output circuit $\mathcal{C}'$ defined to take the value $p_{\mathcal{C}'}(\boldsymbol{V}) = 0$ for any value of $\boldsymbol{V}$ where the function is not well defined.

Each of the operations in the top half of the table is tractable for structured decomposable and smooth circuits (respecting compatible vtrees for products). That is, they do not require support properties for tractability. On the other hand, the tractability of the operations in the bottom half, which we term *deterministic* operations, depends on the input circuit being (marginally) deterministic. In particular, without determinism, `MAX` is known to be `NP`-complete (`MAX`), while `POW` and `LOG` are `#P`-hard [31, 185].

As we have previously discussed in Section 5.1.1, probabilistic marginal inference corresponds to a composition of marginalization and product operations, and does not require any deterministic operations; this means that we do not need to consider support properties. More generally, in [185] it is also shown how to compositionally analyze a range of other inference queries, when deterministic operations appear *before* non-determinstic operations in the composition. However, for classes of inference queries that involve deterministic operations at arbitrary points in the composition, more careful analysis is required. Even though `MARG`, `INST`, `PROD` do not require support properties on the input circuit, they *do* affect the support properties that the output circuit satisfies. As a result, if we have a compositional inference query that consists of one of these operations followed by a deterministic operation, we need to make sure that the output satisfies the required support property.

As an example, let us again consider the napkin formula, expressed as a composition of operations:

$$\mathcal{C}_2 = \texttt{COND}(\mathcal{C}_1; \boldsymbol{W} \cup \boldsymbol{Z}); p_{\mathcal{C}_2}(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p(\boldsymbol{X}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{Z}) \tag{5.8}$$

$$\mathcal{C}_3 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z}); p_{\mathcal{C}_3}(\boldsymbol{W}) = p(\boldsymbol{W}) \tag{5.9}$$

$$\mathcal{C}_4 = \texttt{PROD}(\mathcal{C}_2; \mathcal{C}_3); p_{\mathcal{C}_4}(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p(\boldsymbol{X}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{Z}) p(\boldsymbol{W}) \tag{5.10}$$

$$\mathcal{C}_5 = \texttt{MARG}(\mathcal{C}_4; \boldsymbol{W}); p_{\mathcal{C}_5}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \sum_{\boldsymbol{W}} p(\boldsymbol{X}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{Z}) p(\boldsymbol{W}) \tag{5.11}$$

$$\mathcal{C}_6 = \texttt{COND}(\mathcal{C}_5; \boldsymbol{X} \cup \boldsymbol{Z}); p_{\mathcal{C}_6}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \frac{\sum_{\boldsymbol{W}} p(\boldsymbol{X}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{Z}) p(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p(\boldsymbol{X} | \boldsymbol{W}, \boldsymbol{Z}) p(\boldsymbol{W})} \tag{5.12}$$

In order to be able to compute the final conditional, we require $\mathcal{C}_5$ to be $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic. However, $\mathcal{C}_5$ is itself defined as a marginalization of a product of other circuits; and it is not clear how, or even if, we can enforce properties on the original circuit $\mathcal{C}_1$ to ensure that $\mathcal{C}_5$ is $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic.

5.4.2 Operations on md-vtrees

To tackle these challenges, we apply our md-vtree framework as a unified language for scope and support in structured decomposable circuits. The first step is to understand how each basic operation "transforms" md-vtrees. In other words, if we have input circuit(s) that respect some given md-vtree(s), can we obtain a md-vtree that the output of a basic operation is guaranteed to respect? In Algorithms 5.2-5.4 we show algorithms for applying `MARG`, `INST`, `COND` to md-vtrees. These algorithms are based on, and follow the structure of, the corresponding algorithms for circuits, but act at the level of vtree nodes rather than sum nodes.

The key component of these algorithms is the labelling function assigned to the output md-vtree. For `MARGMDV`, we have that the label for each node is unchanged from the input md-vtree if no variable in the original label is marginalized out, and is set to the universal label otherwise. This reflects the fact that for a $\psi(m)$-deterministic sum node T, where each child has distinct marginalized support over $\psi(m)$, marginalizing any variable in $\psi(m)$ means that we can no longer separate the children in this way.

Algorithm 5.2: `MARGMDV`$(w, \boldsymbol{W})$

```
Input: Input md-vtree $w = (v, \psi)$; set of variables to be marginalized $\boldsymbol{W}$
Result: Output md-vtree $w' = (v', \psi')$
1  $m \leftarrow$ root$(v)$;
2  $m' \leftarrow$ newnode$()$;
3  if $m$ is leaf then
4      $v' \leftarrow$ createvtree$(m')$;              // create vtree with single node
5  else
6      $m_1, m_2 \leftarrow$ children$(m)$;
7      $v'_1, \psi'_1 \leftarrow$ MARGMDV$((v_{m_1}, \psi), \boldsymbol{W})$;
8      $v'_2, \psi'_2 \leftarrow$ MARGMDV$((v_{m_2}, \psi), \boldsymbol{W})$;
9      $v', \psi' \leftarrow v'_1 \cup v'_2, \psi'_1 \cup \psi'_2$;         // combine the vtrees/labelling fn
10     $v' \leftarrow$ addnode$(v'; m')$; $v' \leftarrow$ addchildren$(v'; m',$ root$(v'_1),$ root$(v'_2))$;
11 $\phi'(m') \leftarrow \phi(m) \setminus \boldsymbol{W}$;                         // Update scope function
12 if $\psi(m) \cap \boldsymbol{W} = \emptyset$ then                      // Update labelling function
13     $\psi'(m') \leftarrow \psi(m)$;
14 else
15     $\psi'(m') \leftarrow U$;
16 Return $(v', \psi')$
```

On the other hand, for `INSTMDV`, we remove the instantiated variables from the original label for each node. It can thus be seen that, while marginalization and instantiation affect the scope of the vtree in the same way, their effect on support properties is very different. In particular, given the same input md-vtree, the output md-vtree provides more stringent constraints (smaller labels) on the output circuit in the instantiation case.

For `PRODMDV`, we combine the labels of vtree nodes from the input md-vtrees, by taking their union in Line 11. Given two sum nodes $T^{(1)}, T^{(2)}$, the product algorithm on circuits creates a new sum node, T', that has a child corresponding to every combination of children from $T^{(1)}$ and $T^{(2)}$. The intuition behind this labelling, then, is that if any two children of $T^{(1)}$ correspond to distinct values of $\psi^{(1)}(m)$, and any two children of $T^{(2)}$ correspond to distinct values $\psi^{(2)}(m)$, then any two children of T' will correspond to distinct values of either $\psi^{(1)}(m)$ or $\psi^{(2)}(m)$. For simplicity of presentation, we have shown the product algorithm on md-vtrees in the case where the input vtrees are the same; however, it can be extended to compatible vtrees with some more effort, the details of which we defer to Appendix B.2.

Algorithm 5.3: INSTMDV($w, \boldsymbol{W}$)

```
Input: Input md-vtree $w = (v, \psi)$; set of variables to be instantiated $\boldsymbol{W}$
Result: Output md-vtree $w' = (v', \psi')$;
1  $m \leftarrow$ root($v$);
2  $m' \leftarrow$ newnode();
3  if $m$ is leaf then
4      $v' \leftarrow$ createvtree($m'$);          // create vtree with single node
5  else
6      $m_1, m_2 \leftarrow$ children($m$);
7      $v'_1, \psi'_1 \leftarrow$ INSTMDV(($v_{m_1}, \psi$), $\boldsymbol{W}$);
8      $v'_2, \psi'_2 \leftarrow$ INSTMDV(($v_{m_2}, \psi$), $\boldsymbol{W}$);
9      $v', \psi' \leftarrow v'_1 \cup v'_2, \psi'_1 \cup \psi'_2$;         // combine the vtrees/labelling fn
10     $v' \leftarrow$ addnode($v'; m'$); $v' \leftarrow$ addchildren($v'; m'$, root($v'_1$), root($v'_2$));
11 $\phi'(m') \leftarrow \phi(m) \setminus \boldsymbol{W}$;                          // Update scope function
12 $\psi'(m') \leftarrow \psi(m) \setminus \boldsymbol{W}$;                      // Update labelling function
13 Return $(v', \psi')$
```

Now, given any pipeline, and input md-vtree(s), we can determine if the compositional query is tractable, simply by propagating md-vtree(s) *forward* through the composition, and checking that the input md-vtree(s) to any intermediate operation satisfy the requirements in Table 5.1. Importantly, this can be done *without doing the computation of the query itself*; all of our algorithms run in polytime in the number of variables $|\boldsymbol{V}|$, which is much smaller than the circuits themselves. If the composition is tractable, then we can apply the corresponding circuit algorithms according to the composition, which are polytime in the sizes of the input circuits.

5.4.3 The MD-calculus

So far, we have now reduced reasoning about tractability given the input circuits, to reasoning about tractability given the input md-vtrees. However, when learning circuits from data, we have the freedom to choose the input md-vtree(s), for instance through MDNets, in order to enable tractable inference. That is, rather than asking "*is the backdoor query tractable given input md-vtree w?*", we now consider the question "*for which input md-vtrees w is the backdoor query tractable?*". This requires *backward* analysis through a composition, to find what input md-vtrees will lead to a given output md-vtree. Unfortunately, none of the algorithms shown for marginalization,

Algorithm 5.4: PRODMDV($\mathcal{C}^{(1)}, \mathcal{C}^{(2)}$)

```
Input: Input md-vtrees $w^{(1)} = (v, \psi^{(1)}), w^{(2)} = (v, \psi^{(2)})$ with same vtree $v$
Result: Output circuit $w' = (v', \psi')$
1  $m \leftarrow$ root($v$);
2  $(m_1, m_2) \leftarrow$ children($m$);                    // null if $m$ is leaf
3  $m' \leftarrow$ newvtreenode();
4  if $m$ is leaf then
5      $v', \psi' \leftarrow$ createvtree(m');
6  else
7      $v'_1, \psi'_1 \leftarrow$ PRODMDV($(v_{m_1}, \psi^{(1)}), (v_{m_1}, \psi^{(2)})$);
8      $v'_2, \psi'_2 \leftarrow$ PRODMDV($(v_{m_2}, \psi^{(1)}), (v_{m_2}, \psi^{(2)})$);
9      $v', \psi' \leftarrow v'_1 \cup v'_2, \psi'_1 \cup \psi'_2$;      // combine the vtrees/labelling fn
10 $\phi'(m') \leftarrow \phi(m)$;                          // Update scope function
11 $\psi'(m') \leftarrow \psi^{(1)}(m) \cup \psi^{(2)}(m)$;           // Update labelling function
12 $v' \leftarrow$ addnode($v'; m'$);
13 $v' \leftarrow$ addchildren($v'; m'$, root($v'_1$), root($v'_2$));
14 Return $(v', \psi')$
```

instantiation, or products are invertible, making this a challenging combinatorial problem.

Thus, instead of explicitly enumerating all input md-vtrees that enable tractable computation of the compositional query, we propose a simpler approach that is sound (that is, always derives tractable input md-vtrees), but not necessarily complete (may fail to find all tractable input md-vtrees). We observe that the key barriers to tractability in compositions are the deterministic operations, which require the input circuit to be (marginally) deterministic. We thus propose to propagate sets of *required* marginal determinism backwards through the composition. To this end, in Table 5.2 we show a set of input-output conditions called the *MD-calculus*. These are sufficient conditions on the input(s) to an operation to guarantee that the output is $\boldsymbol{Q}$-deterministic. The MD-calculus forms a set of rules that we can apply backwards from deterministic operations (which require $\boldsymbol{V}$-determinism), in order to determine a sufficient set of marginal determinisms $\boldsymbol{S}$ for each intermediate circuit. Finally, we can enforce those marginal determinisms on the input md-vtree(s) using Definition 5.17.

Theorem 5.5 (MD-calculus)**.** *The conditions in Table 5.2 hold.*

Operation	**Input Condition**	**Output**
$\texttt{MARG}(\mathcal{C};\boldsymbol{W})$	$\boldsymbol{Q}$-det	$\boldsymbol{Q}$-det
$\texttt{INST}(\mathcal{C};\boldsymbol{w})$	$\exists \boldsymbol{W}' \subseteq \boldsymbol{W} : (\boldsymbol{Q} \cup \boldsymbol{W}')$-det	$\boldsymbol{Q}$-det
$\texttt{PROD}(\mathcal{C}^{(1)},\mathcal{C}^{(2)})$	$\exists \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} : \boldsymbol{Q}^{(1)}$-det, $\boldsymbol{Q}^{(2)}$-det, and: • Either (a) $\boldsymbol{Q} \subseteq \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$ and $\boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)} = \boldsymbol{Q}$; • Or (b) $\boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} \supseteq \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$ and $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)}$	$\boldsymbol{Q}$-det
$\texttt{COND}(\mathcal{C};\boldsymbol{W})$	$\boldsymbol{Q}$-det	$\boldsymbol{Q}$-det
$\texttt{POW}(\mathcal{C};\alpha)$	$\boldsymbol{Q}$-det	$\boldsymbol{Q}$-det
$\texttt{MAX}(\mathcal{C})$	N/A	N/A
$\texttt{LOG}(\mathcal{C})$	-	-

Table 5.2: MD-calculus: sufficient input-output conditions for each basic operation

Proof. (Sketch) We provide here proofs of these results for marginalization, instantiation, and products (in the case of same vtrees); the rest of the proof can be found in Appendix B.1. For each, we need to show that, assuming the input md-vtree satisfies the *input condition*, applying the *algorithm* on md-vtrees results in an output md-vtree satsifying the *output condition*.

$\texttt{MARG}(\cdot;\boldsymbol{W})$ For the marginalization operation, the output md-vtree is over variables $\boldsymbol{V} \setminus \boldsymbol{W}$. Thus, let $\boldsymbol{Q}$ be any subset of $\boldsymbol{V} \setminus \boldsymbol{W}$.

- *Input Condition:* The input condition requires that the input md-vtree w implies $\boldsymbol{Q}$-determinism; that is, for every vtree node m, either $\phi(m) \cap \boldsymbol{Q} = \emptyset$, or else $\boldsymbol{Q} \supseteq \psi(m)$.
- *Algorithm:* In Algorithm 5.2, every vtree node m' in the output md-vtree corresponds to a vtree node m' in the input md-vtree, such that $\phi'(m') = \phi(m) \setminus \boldsymbol{W}$, and $\psi'(m') = \psi(m)$ if $\psi(m) \cap \boldsymbol{W} = \emptyset$, or $\psi'(m') = U$ otherwise.
- *Proof for Output Condition:* For each vtree node m', if $\phi'(m') \cap \boldsymbol{Q} \neq \emptyset$, then,

we have that:

$$\begin{aligned}
&(\phi(m) \cap \boldsymbol{Q}) \setminus \boldsymbol{W} \neq \emptyset && \text{(by effect of algorithm)}\\
\implies &\phi(m) \cap \boldsymbol{Q} \neq \emptyset && \text{(weaker statement)}\\
\implies &\psi(m) \subseteq \boldsymbol{Q} && \text{(by input condition)}\\
\implies &\psi'(m') \subseteq \boldsymbol{Q}
\end{aligned}$$

The last line follows since $\boldsymbol{Q} \cap \boldsymbol{W} = \emptyset$, so $\psi(m) \cap \boldsymbol{W} = \emptyset$, and so we are in the algorithm case where the label is "copied". Thus, we have shown that the output md-vtree implies $\boldsymbol{Q}$-determinism, as required.

`INST`$(\cdot; \boldsymbol{w})$ For the instantiation operation, the output md-vtree is over variables $\boldsymbol{V} \setminus \boldsymbol{W}$. Thus, let $\boldsymbol{Q}$ be any subset of $\boldsymbol{V} \setminus \boldsymbol{W}$.

- *Input Condition:* The input condition requires that the input md-vtree w implies $(\boldsymbol{Q} \cup \boldsymbol{W}')$-determinism for some $\boldsymbol{W}' \subseteq \boldsymbol{W}$; that is, for every vtree node m, either $\phi(m) \cap (\boldsymbol{Q} \cup \boldsymbol{W}') = \emptyset$, or else $\boldsymbol{Q} \cup \boldsymbol{W}' \supseteq \psi(m)$.
- *Algorithm:* In Algorithm 5.3, every vtree node m' in the output md-vtree corresponds to a vtree node m' in the input md-vtree, such that $\phi'(m') = \phi(m) \setminus \boldsymbol{W}$, and $\psi'(m') = \psi(m) \setminus \boldsymbol{W}$.
- *Proof for Output Condition:* For each vtree node m', if $\phi'(m') \cap \boldsymbol{Q} \neq \emptyset$, then, we have that:

$$\begin{aligned}
&(\phi(m) \cap \boldsymbol{Q}) \setminus \boldsymbol{W} \neq \emptyset && \text{(by effect of algorithm)}\\
\implies &\phi(m) \cap \boldsymbol{Q} \neq \emptyset && \text{(weaker statement)}\\
\implies &\phi(m) \cap (\boldsymbol{Q} \cup \boldsymbol{W}') \neq \emptyset && \text{(weaker statement)}\\
\implies &\psi(m) \subseteq \boldsymbol{Q} \cup \boldsymbol{W}' && \text{(by input condition)}\\
\implies &\psi'(m') \subseteq \boldsymbol{Q}
\end{aligned}$$

Here, the last line follows since the new label $\psi'(m') = \psi(m) \setminus \boldsymbol{W}$ removes all elements of $\boldsymbol{W}$, and thus $\boldsymbol{W}'$, from $\psi(m)$. Thus, we have shown that the output md-vtree implies $\boldsymbol{Q}$-determinism, as required.

PROD$(\cdot, \cdot)$ For the product operation, in the case the input vtrees are the same, the output md-vtree is over variables $\boldsymbol{V}^{(1)} \cup \boldsymbol{V}^{(2)} = \boldsymbol{V} \cup \boldsymbol{V} = \boldsymbol{V}$. Thus, let $\boldsymbol{Q}$ be any subset of $\boldsymbol{V}$.

- *Input Condition:* The input condition requires that the first input md-vtree $w^{(1)}$ implies $\boldsymbol{Q}^{(1)}$-determinism, and the second input md-vtree $w^{(2)}$ implies $\boldsymbol{Q}^{(2)}$-determinism, where *one* of the following holds:

  (a) $\boldsymbol{Q} \subseteq \boldsymbol{V}$ and $\boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)} = \boldsymbol{Q}$;

  (b) $\boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} \supseteq \boldsymbol{V}$ and $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)}$

  Here, case (b) is not relevant (the only possible case of $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)}$ is also captured by (a)).

- *Algorithm:* In Algorithm 5.4, every vtree node m' in the output md-vtree corresponds to a vtree node m in the input md-vtrees respectively, such that $\phi'(m') = \phi(m)$. The label is given by the union $\psi'(m') = \psi^{(1)}(m) \cup \psi^{(2)}(m)$.

- *Proof for Output Condition:* We need to show that for either input condition (a), (b), the condition for implied $\boldsymbol{Q}$-determinism holds on m'; that is, if $\phi'(m') \cap \boldsymbol{Q} \neq \emptyset$, then $\psi'(m') \subseteq \boldsymbol{Q}$. If $\phi'(m') \cap \boldsymbol{Q} \neq \emptyset$, we have:

$$\begin{aligned}
&\phi(m) \cap \boldsymbol{Q} \neq \emptyset && \text{(as scope does not change} \\
\implies &\psi^{(1)}(m) \supseteq \boldsymbol{Q}^{(1)} \text{ and } \psi^{(2)}(m) \supseteq \boldsymbol{Q}^{(2)} && \text{(by marginal determinisms)} \\
\implies &\psi^{(1)}(m) \cup \psi^{(2)}(m) \supseteq \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)} && \text{(combining previous statements)} \\
\implies &\psi'(m) \supseteq \boldsymbol{Q} \neq \emptyset && \text{(rewriting both sides)}
\end{aligned}$$

Thus we have shown the md-calculus rule for PROD when the input vtrees are the same, which essentially says that if both input md-vtrees imply a $\boldsymbol{Q}$-determinism, then so does the output md-vtree.

$\square$

5.4.4 Examples

We conclude this section by demonstrating the application of the MD-calculus to previously studied compositional probabilistic inference queries, where it provides additional insight. Firstly, we consider the marginal MAP query $\max_{\boldsymbol{Q}} p_{\mathcal{C}}(\boldsymbol{Q}, \boldsymbol{e})$. This can be written as a composition $\texttt{MAX}(\texttt{MARG}(\texttt{INST}(\mathcal{C}; \boldsymbol{e}), \boldsymbol{V} \setminus (\boldsymbol{Q} \cup \boldsymbol{E})))$. In order for the maximization query to be tractable, we need its input to be deterministic. Applying the MD-calculus backwards through the composition, we obtain:

1. **Requirement:** $\mathcal{C}_1 = \texttt{MARG}(\texttt{INST}(\mathcal{C}; \boldsymbol{e}), \boldsymbol{V} \setminus (\boldsymbol{Q} \cup \boldsymbol{E}))$ is $\boldsymbol{Q}$-deterministic.

2. $\mathcal{C}_2 = \texttt{INST}(\mathcal{C}; \boldsymbol{e})$ is $\boldsymbol{Q}$-deterministic $\implies$ $\mathcal{C}_1$ is $\boldsymbol{Q}$-deterministic.

3. **Sufficient Condition:** $\exists \boldsymbol{W}' \subseteq \boldsymbol{E}$ s.t. $\mathcal{C}$ is $(\boldsymbol{Q} \cup \boldsymbol{W}')$-deterministic $\implies$ $\mathcal{C}_2$ is $\boldsymbol{Q}$-deterministic.

That is, the input circuit $\mathcal{C}$ needs to be $(\boldsymbol{Q} \cup \boldsymbol{W}')$-deterministic for some $\boldsymbol{W}' \subseteq \boldsymbol{E}$. This is a weaker condition than was previously known for tractable computation of marginal MAP, which instead required the circuit to be $\boldsymbol{Q}$-deterministic regardless of the evidence [128, 77, 33, 32]. The practical implication is that a $\boldsymbol{W}$-deterministic circuit can answer marginal MAP queries for any $\boldsymbol{Q}, \boldsymbol{E}$ such that $\boldsymbol{Q} \subseteq \boldsymbol{W} \subseteq \boldsymbol{Q} \cup \boldsymbol{E}$.

As a second example, we consider the mutual information between two sets of variables:

$$\sum_{\boldsymbol{X}, \boldsymbol{Y}} p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Y}) \log \frac{p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Y})}{p_{\mathcal{C}}(\boldsymbol{X}) p_{\mathcal{C}}(\boldsymbol{Y})} \tag{5.13}$$

For mutual information, applying a similar approach we obtain that $\mathcal{C}$ should be both $\boldsymbol{X}$-deterministic and $\boldsymbol{Y}$-deterministic. However, this would require that the root sum node is both $\boldsymbol{X}$-deterministic and $\boldsymbol{Y}$-deterministic, and we have seen in Proposition 5.3 that this is not possible without restricting support.

5.5 Causal Inference using MD-Calculus

In this section, we apply our md-vtree framework to analyse tractability conditions for exact causal inference for PCs. In particular, we constructively derive tractability conditions for exactly computing interventional distributions for the backdoor, front-door, and (extended) napkin formulae. The resulting algorithms constitute the first poly-time algorithms for causal inference on probabilistic circuits that do not rely on a compilation assumption, and can be used whenever the interventional distribution is identified by one of these formulae. Representing the interventional distribution as a (structured decomposable and smooth) circuit then enables us to reason about causal queries of interest, such as average causal effects, as the moments (e.g. expectation) of such a circuit are tractable [33]. Throughout this section, we assume *positivity*, that is, $p(\boldsymbol{V}) > 0$; this is a standard assumption in causal inference to ensure that the interventional distributions are well-defined [168].

5.5.1 MD-calculus for Causal Formulae

5.5.1.1 Backdoor

We begin with the backdoor formula. Recall that in cases where there is a valid backdoor adjustment set, such as in Figure 5.2a, we have the following formula for the interventional distribution:

$$p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{Z}) \tag{5.14}$$

Before continuing, it is worth noting that the expression above is valid for any value of $\boldsymbol{X}, \boldsymbol{Y}$, though in causal inference it is more typical that we are interested in the interventional distribution $p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y})$ for specific values $\boldsymbol{x}$ of $\boldsymbol{X}$, i.e. a specific intervention, or a small set of interventions. This distinction is important as we will see, interestingly, that instantiating $\boldsymbol{X}$ makes the query more tractable in the sense that the marginal determinism requirements for the circuit $\mathcal{C}$ are more relaxed. Now, given a circuit $\mathcal{C}$ representing the observational distribution $p(\boldsymbol{V})$, the interventional distribution can be expressed using the following composition:

$$\mathcal{C}_1 = \texttt{MARG}(\mathcal{C}; \boldsymbol{V} \setminus (\boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z})); p_{\mathcal{C}_1}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) \tag{5.15}$$

$$\mathcal{C}_2 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{X} \cup \boldsymbol{Y}); p_{\mathcal{C}_2}(\boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z}) \tag{5.16}$$

$$\mathcal{C}_3 = \texttt{COND}(\mathcal{C}_1; \boldsymbol{X} \cup \boldsymbol{Z}); p_{\mathcal{C}_3}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{X}, \boldsymbol{Z}) \tag{5.17}$$

$$\mathcal{C}_4 = \texttt{PROD}(\mathcal{C}_2, \mathcal{C}_3); p_{\mathcal{C}_4}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{X}, \boldsymbol{Z}) \tag{5.18}$$

$$\mathcal{C}_5 = \texttt{MARG}(\mathcal{C}_4; \boldsymbol{Z}); p_{\mathcal{C}_5}(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{X}, \boldsymbol{Z}) \tag{5.19}$$

Finally, we have $p_{\mathcal{C}_5}(\boldsymbol{X}, \boldsymbol{Y}) = p_{\mathcal{C}, \boldsymbol{X}}(\boldsymbol{Y})$. To apply the MD-calculus, we first identify the deterministic operations in the pipeline. For the backdoor query, the only deterministic operation is the $\texttt{COND}(\cdot; \boldsymbol{X} \cup \boldsymbol{Z})$ operation, which requires $\boldsymbol{X} \cup \boldsymbol{Z}$-determinism, and is applied to $\mathcal{C}_1$. Then, we can work *backwards* through the pipeline from $\mathcal{C}_1$ in order to derive tractability conditions on $\mathcal{C}$.

1. **Requirement**: $\mathcal{C}_1 = \texttt{MARG}(\mathcal{C}; \boldsymbol{V} \setminus (\boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z}))$ is $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic.

2. **Sufficient Condition**: $\mathcal{C}$ is $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic.

This simple derivation shows that it suffices for $\mathcal{C}$ to be $(X \cup Z)$-deterministic to compute the backdoor query. Now, let us consider the case in which we instantiate $\boldsymbol{X}$ with a specific value $\boldsymbol{x}$. In this case, the composition becomes:

$$\mathcal{C}_1 = \texttt{INST}(\mathcal{C}; \boldsymbol{x}); p_{\mathcal{C}_1}(\boldsymbol{V} \setminus \boldsymbol{X}) = p_{\mathcal{C}}(\boldsymbol{V} \setminus \boldsymbol{X}, \boldsymbol{x}) \tag{5.20}$$

$$\mathcal{C}_2 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{V} \setminus (\boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z})); p_{\mathcal{C}_2}(\boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{Z}) \tag{5.21}$$

$$\mathcal{C}_3 = \texttt{MARG}(\mathcal{C}; \boldsymbol{V} \setminus \boldsymbol{Z}); p_{\mathcal{C}_3}(\boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z}) \tag{5.22}$$

$$\mathcal{C}_4 = \texttt{COND}(\mathcal{C}_2; \boldsymbol{Z}); p_{\mathcal{C}_4}(\boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{x}, \boldsymbol{Z}) \tag{5.23}$$

$$\mathcal{C}_5 = \texttt{PROD}(\mathcal{C}_3, \mathcal{C}_4); p_{\mathcal{C}_4}(\boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{x}, \boldsymbol{Z}) \tag{5.24}$$

$$\mathcal{C}_6 = \texttt{MARG}(\mathcal{C}_5; \boldsymbol{Z}); p_{\mathcal{C}_6}(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{x}, \boldsymbol{Z}) \tag{5.25}$$

We have that $p_{\mathcal{C}_6}(\boldsymbol{Y}) = p_{\mathcal{C}, \boldsymbol{x}}(\boldsymbol{Y})$. In the composition above, note in particular that for $\mathcal{C}_4$ we only condition with respect to $\boldsymbol{Z}$, and not $\boldsymbol{X} \cup \boldsymbol{Z}$, as $p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{x}, \boldsymbol{Z}) = \frac{p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{Z})}{p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Z})} = \frac{p_{\mathcal{C}_2}(\boldsymbol{X}, \boldsymbol{Z})}{p_{\mathcal{C}_2}(\boldsymbol{Z})}$. Here, employing the MD-calculus starting from $\mathcal{C}_2$ gives us the following:

1. **Requirement**: $\mathcal{C}_2 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{V} \setminus (\boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z}))$ is $\boldsymbol{Z}$-deterministic.

2. $\mathcal{C}_1 = \texttt{INST}(\mathcal{C}; \boldsymbol{x})$ is $\boldsymbol{Z}$-deterministic.

3. **Sufficient Condition**: $\mathcal{C}$ is $(\boldsymbol{X}' \cup \boldsymbol{Z})$-deterministic for some $\boldsymbol{X}' \subseteq \boldsymbol{X}$.

This shows that the instantiated backdoor adjustment is tractable for a wider range of circuits than if we insisted on a circuit encoding $p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y})$ as a function of $\boldsymbol{X}$ (and $\boldsymbol{Y}$).

Proposition 5.7 (Tractable Backdoor Adjustment). *Let $\mathcal{C}$ be a structured decomposable and smooth circuit. Then, in the backdoor case:*

- *Computing the interventional distribution $p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^2)$ time, if $\mathcal{C}$ is additionally $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic;*

- *Computing the instantiated interventional distribution $p_{\mathcal{C},\boldsymbol{x}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^2)$ time, if $\mathcal{C}$ is additionally $(\boldsymbol{X}' \cup \boldsymbol{Z})$-deterministic for some $\boldsymbol{X}' \subseteq \boldsymbol{X}$.*

5.5.1.2 Frontdoor

$$p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}|\boldsymbol{X}) \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y}|\boldsymbol{X}', \boldsymbol{Z}) \tag{5.26}$$

We can similarly derive results for the frontdoor formula, where again it is the case that instantiating $\boldsymbol{X}$ leads to weaker conditions for tractability.

Proposition 5.8 (Tractable Frontdoor Adjustment). *Let $\mathcal{C}$ be a structured decomposable and smooth circuit. Then, in the frontdoor case:*

- *Computing the interventional distribution $p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^3)$ time, if $\mathcal{C}$ is additionally $\boldsymbol{X}$-deterministic and $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic;*

- *Computing the instantiated interventional distribution $p_{\mathcal{C},\boldsymbol{x}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^3)$ time, if $\mathcal{C}$ is additionally $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic.*

Dataset	$\|\boldsymbol{Z}\|$	**Error**		**Time**	
		MD	Counting	MD	Counting
Asia	4	0.269	**0.0143**	**0.5**	1.0
Sachs	4	0.180	**0.0219**	1.7	**0.7**
Child	13	**0.0802**	0.135	1.9	**0.9**
Win95pts	59	**0.0044**	0.0511	3.2	**0.9**
Andes	202	**0.0382**	0.0982	7.9	**1.3**

Table 5.3: Backdoor Estimation (averaged over 10 runs)

5.5.1.3 Napkin

$$p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y}) = \frac{\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{W}, \boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{X}|\boldsymbol{W}, \boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{W})} \tag{5.27}$$

Finally, for the napkin formula, it turns out that we cannot derive a valid tractability condition for $p_{\boldsymbol{X}}(\boldsymbol{Y})$ using the MD-calculus. The reason is basically that we would require $p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{W}, \boldsymbol{Z})p_{\mathcal{C}}(\boldsymbol{W})$ to satisfy $(\boldsymbol{X} \cup \boldsymbol{Z})$-determinism, which cannot be propagated through the product according to the MD-calculus product rule. However, if we instantiate $\boldsymbol{X}$, and take advantage of the fact that the napkin formula is valid for any value of $\boldsymbol{Z}$, then computing the interventional distribution becomes possible:

Proposition 5.9 (Tractable Napkin). *Let $\mathcal{C}$ be a structured decomposable and smooth circuit. Then, in the napkin case:*

- *Computing the instantiated interventional distribution $p_{\mathcal{C},\boldsymbol{x}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^2)$ time, if $\mathcal{C}$ is additionally $(\boldsymbol{W} \cup \boldsymbol{X}' \cup \boldsymbol{Z})$-deterministic for some $\boldsymbol{X}' \subseteq \boldsymbol{X}$.*

5.6 Experiments

In this section, we empirically evaluate our tractable algorithm for backdoor adjustment derived using MD-calculus. We generate datasets by sampling 1000 datapoints from the (discrete variable) Bayesian network (BN) models in the *bnrepository* [160], and learn a MDNet over all variables $\boldsymbol{V}$ from data. For each Bayesian network (causal graph), we select a single treatment variable X and single outcome variable Y, as well as a set of variables $\boldsymbol{Z}$ forming a valid backdoor adjustment set for (X, Y), and seek to estimate $\sum_{\boldsymbol{Z}} p(Y|X, \boldsymbol{Z})p(\boldsymbol{Z})$. We manually select a vtree that splits the scope into $(X \cup \boldsymbol{Z})$ and Y at the root, and thereafter generates the rest of the vtree randomly. Given a vtree, we use the $\boldsymbol{S}$-constrained labelling (Definition 5.17) with $\boldsymbol{S} = \boldsymbol{X} \cup \boldsymbol{Z}$. The required tractability properties for backdoor adjustment are then enforced through the structure of the corresponding MDNet.

The results are shown in Table 5.3; for comparison, we also show results for the *counting* approach, where we estimate $p(Y|X, \boldsymbol{Z})$ as $\frac{N_{Y,X,\boldsymbol{Z}}}{N_{X,\boldsymbol{Z}}}$, where N refers to the number of datapoints with the subscripted assignment of variables (this is set to 0 if $N_{X,\boldsymbol{Z}} = 0$). It can be seen that, while the counting approach is generally more robust in lower dimensions, the advantage in terms of learning a full model becomes apparent with the higher-dimensional adjustment sets, as shown by the lower error on the Win95pts and Andes datasets. Remarkably, as the size of the adjustment set $\boldsymbol{Z}$ increases, the time taken for the algorithm based on the MD-calculus increases only approximately *linearly* in the dimension. This illustrates the attractiveness of tractable probabilistic modelling, in that we can systematically control the computational cost of exact inference by restricting the size of the PC model.

5.7 Discussion

To summarize, this Chapter has focused on two related problems; performing causal reasoning given a learned model of the observational distribution, and designing classes of circuits where such queries (and other similar queries) are tractable. For the latter, we introduced the md-vtree framework for compactly specifying classes of structured decomposable circuits according to their support properties. The framework naturally provides insight into the properties of previously proposed PC classes such as strongly deterministic circuits, as well as inspiring our newly designed MDNet architecture for constructing circuits according to a md-vtree. We further showed how to derive tractability conditions, in terms of the md-vtree, for compositions of basic operations on circuits. This includes, in particular, prominent causal identification formulae, but the compositional framework is general and can, in theory, be applied to derive tractability conditions for many inference queries beyond those we have discussed.

Though we have presented the framework with discrete variables, the theory applies also to continuous variables, with the only modification being to replace summation in marginalization operations with integration.

On the causal reasoning side, we began this Chapter with the motivation of decoupling the learning or specification of probabilistic models (of the observational probability distribution) from the causal assumptions (diagram) that enable identifiability of interventional probabilities. Our first result showed that there is no existing class of PCs that enables tractable computation of interventional probabilities in general. Investigating further, we found that marginal determinism is the key support property that enables tractable computation of common do-calculus formulae such as the backdoor, frontdoor, and napkin formulae. That is, from the perspective of computational difficulty, there is still a very real gap between reasoning on the observational distribution represented by a PC (which typically only requires decomposability and smoothness), and transferring this information and reasoning on an interventional distribution. Further, in each of these cases, we found a different marginal determinism condition for tractability, and further that this condition depends on the identities of the variables in the causal diagram. The consequence is that we still require some knowledge of the underlying causal diagram before designing/learning the circuit from data.

Though we have chosen to focus on exact computation of the interventional distribution, as is typically demanded of TPMs, a promising future direction is to leverage the insights from our exact analysis to aid designing effective approximate inference routines. For example, the KL-divergence between two structured decomposable and smooth circuits with compatible vtrees is tractable [105, 185]. Thus, if we have learned a PC representing the observational distribution, but which does not satisfy the required marginal determinisms for tractability of some interventional query, we can employ a md-vtree which does imply the required marginal determinisms (and with the same vtree) as a tractable variational family. This could resolve the need to know the identity of the causal variables in advance.

We have taken the view in this Chapter that PCs are just a model for the observational probability distribution, and not inherently causal. However, it is also interesting to consider whether circuits can be used as a causal model of a system; that is, expressing causality without the need for an external causal diagram [126, 201]. We have already seen an example of such a "causal probabilistic circuit", in the form of arithmetic circuits compiled from Bayesian networks in the last Chapter. In this case, applying interventions is as simple as instantiating the value of symbolic parameters in the circuit. However, for a generic probabilistic circuit, whose structure and/or parameters may have been learned from data, there is no corresponding reference point to a graphical model [31].

This observation has spurred recent efforts towards finding interpretations of probabilistic circuits as latent variable models [210, 136, 19], by assigning discrete latent variables corresponding to the mixture components (children) of sum nodes; this interpretation has proven fruitful for designing new learning algorithms for circuits [209, 107]. Given this interpretation, one might wonder if it is possible to interpret a general probabilistic circuit as a *causal* graphical model, such as a causal Bayesian network or structural causal model. Unfortunately, most latent-variable interpretation schemes are typically not particularly human-interpretable or necessarily representative of underlying structure in the domain, due to the large number of latent variables introduced. More pertinently for causality, the resulting graphical models have a bipartite structure such that there are no edges between endogenous variables, leading to a lack of causal *expressivity*: in particular, intervening on any endogenous variable has no effect on the distribution of any other endogenous variable [126][8]. Even if it were possible to introduce some causal semantics for computing interventional distributions into a general PC (as in compiled PCs), such a causal PC would not be *learnable* (identifiable) from observational data, as is the case with neural probabilistic models [198]. Our work in this Chapter identifies a third challenge; namely that, even if there exists a sufficiently causally *expressive* interpretation of PCs, which is *learnable* (for example, using both observational and experimental data), inference on the causal PC may not be *tractable*.

[8]The notable exception is the decompilation algorithm of [19]; however, this assumes that the PC is compiled from a Bayesian network using a reverse topological order and so is not applicable unless the graphical model (BN) was previously known.

Chapter 6

Tractable Causal Reasoning with Structural Uncertainty

Contents

6.1 Bayesian Causal Reasoning

In previous Chapters, we focused on models of the observed probability distribution $p(\boldsymbol{V})$, whilst assuming that sufficient information about the causal graph was either

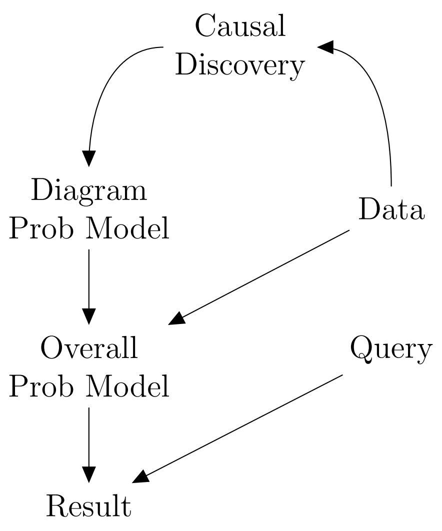


Figure 6.1: Reasoning under diagram uncertainty

known a priori, or would be available at the time of reasoning to identify the query of interest. On the other hand, in this Chapter, we turn to the problem of causal reasoning in the setting where we are fundamentally uncertain about the underlying causal graph. This is significantly more challenging from a computational perspective, as we need both a means of representing or modelling this uncertainty over graphs, as well as reasoning to make inferences about not only the causal graph, but also downstream quantities such as interventional probabilities.

Given the conceptual separation between the causal graph and the observational distribution in causal inference, it is tempting to try modelling the distributions on the graph and variables independently to reduce complexity. For example, for the former, we could learn a MDNet for $p(\boldsymbol{V})$, while for the latter, we could separately specify or learn a distribution $p(G)$ over graphs. However, this is arguably not the right approach when we have uncertainty over graphs, for two reasons. Firstly, from a *learning* perspective, causal graphs have testable implications on the observational distribution $p(\boldsymbol{V}|G)$ that they generate. This includes conditional independences that follow from the graphical structure, as well as additional signals if parametric assumptions about the local conditional probability distributions are made [167]. The consequence is the domain variables $\boldsymbol{V}$ and graph G over those variables are correlated, which cannot be learned if we specify separate models for $p(\boldsymbol{V})$ and $p(G)$. Secondly, from the *reasoning* perspective, a significant issue is that there is no known probabilistic model, including MDNets, that enables tractable causal reasoning for all causal graphs G. As such, it is likely that we will need to specialize the specification of the observational distribution to particular graphs (or sets of graphs). For these reasons, we need to model the joint distribution $p(G, \boldsymbol{V})$ over graphs and variables.

The overall schematic of the learning and reasoning framework we propose is shown in Figure 6.1. Given data (and any domain knowledge specified in the prior), we obtain a probabilistic model that represents our uncertainty over the causal diagram. This can then be combined with the distribution on the domain variables given a graph, in order to form an overall probabilistic model $p(G, \boldsymbol{V})$. Finally, given a causal reasoning query, such as an interventional probability, we return a result that accounts for the uncertainty over the graphical structure. The key challenge that we aim to address is how to efficiently represent, learn and reason over these probabilistic models, which fundamentally involve discrete graphical structures.

6.1.1 Bayesian Structure Learning

We begin by setting up the learning problem. Recall that, given a causal Bayesian network $\mathcal{CBN} = (G, Pr)$, the joint distribution over variables $\boldsymbol{V}$ is given by the factorization:

$$p(\boldsymbol{V}|G, Pr) = \prod_{i=1}^{d} Pr_i(V_i|G_i) \tag{6.1}$$

where we have emphasized the dependence of the distribution on the graph G and local conditional probability distributions Pr, and written G_i to denote the set of parents $\mathrm{pa}_G(V_i)$ of variable V_i in DAG G. Now, given a dataset $\mathcal{D} = \{\boldsymbol{v}^{(1)}, ..., \boldsymbol{v}^{(n)}\}$ of observations of the variables, in Bayesian structure learning, the goal is to learn a distribution that represents uncertainty over the space of causal graphs. This requires specifying priors for the BN graph and CPDs, in addition to the likelihood above.

- **Prior over Graphs** $p_{prior}(G)$: This should capture our prior belief about the causal graph. It is possible to either place a non-informative prior on the graph (e.g. a sparsity that penalizes more dense graphs), or we can impose domain knowledge which indicates the presence (or absence) of certain edges. If the prior decomposes as $p_{prior}(G) = \prod_{i=1}^{d} p_{prior,i}(G_i)$, then we say it is *modular.*

- **Prior over CPDs** $p(Pr|G)$: We also need to specify a prior over the distribution of the Bayesian network, given the graph. For example, in the common case of linear Gaussian Bayesian networks, we have $V_i = b_j + \sum_j B_{ij} V_j + \epsilon_i$, with $\epsilon_i \sim \text{Gaussian}(0, \Sigma_{ii}^2))$, where $b \in \mathbb{R}^d, B \in \mathbb{R}^{d \times d}, \Sigma \in \mathbb{R}^{d \times d}$ are parameters. The dependence on the graph comes from the fact that we have $B_{ij} = 0$ whenever $G_{ij} = 0$, i.e. there is no edge from V_i to V_j. The prior $p(Pr|G)$ over CPDs can then be expressed using a prior over the parameters $p(\boldsymbol{b}, B, \Sigma|G)$. If the CPD prior decomposes as $p(Pr|G) = \prod_{i=1}^{d} p_i(Pr_i|G_i)$, then we say it is modular.

Together, this specifies a joint distribution $p(G, Pr, \boldsymbol{V}) = p_{prior}(G)p(Pr|G)p(\boldsymbol{V}|G, Pr)$. We can marginalize out Pr to obtain a distribution over just graph and variables:

$$p(G, \boldsymbol{V}) = p_{prior}(G) \int_{Pr} p(\boldsymbol{V}|G, Pr)p(Pr|G)dPr \tag{6.2}$$

The quantity $p_{lh}(\boldsymbol{V}|G) := \int_{Pr} p(\boldsymbol{V}|G, Pr)p(Pr|G)dPr$ is known as the *marginal likelihood.* If the prior over CPDs is modular, then the marginal likelihood can also be written in a modular fashion, as:

$$\begin{aligned} p_{lh}(\boldsymbol{V}|G) &= \int_{Pr} p(\boldsymbol{V}|G, Pr)p(Pr|G)dPr && (6.3)\\ &= \int_{Pr} \prod_{i=1}^{d} Pr_i(V_i|G_i)p_i(Pr_i|G_i)dPr && (6.4)\\ &= \prod_{i=1}^{d} p_{lh,i}(V_i|G_i) && (6.5) \end{aligned}$$

where we define $p_{lh,i}(V_i|G_i) := \int_{Pr_i} Pr_i(V_i|G_i)p_i(Pr_i|G_i)dPr_i$ for the marginal likelihood of G_i given the datapoint $\boldsymbol{V}$.

Now, given an observed dataset $\mathcal{D} = \{\boldsymbol{v}^{(1)}, ..., \boldsymbol{v}^{(n)}\}$, and writing $p_{lh}(\mathcal{D}|G) = \int_{Pr} \left(\prod_{j=1}^{n} p(\boldsymbol{v}^{(j)}|G, Pr)\right) p(Pr|G)dPr$ for the marginal likelihood of the dataset, we can apply Bayes' theorem to obtain a posterior distribution over graphs:

$$p_G(G|\mathcal{D}) \propto p_{prior}(G)p_{lh}(\mathcal{D}|G) \tag{6.6}$$

Given a modular graph prior and CPD prior, this can be written as:

$$\begin{aligned}
p_G(G|\mathcal{D}) &\propto p_{prior}(G)p_{lh}(\mathcal{D}|G) && (6.7)\\
&= \left(\prod_{i=1}^{d} p_{prior,i}(G_i)\right) \int_{Pr} \left(\prod_{j=1}^{n}\prod_{i=1}^{d} Pr_i(v_i^{(j)}|G_i)\right)\left(\prod_{i=1}^{d} p_i(Pr_i|G_i)\right) dPr && (6.8)\\
&= \left(\prod_{i=1}^{d} p_{prior,i}(G_i)\right) \prod_{i=1}^{d} \int_{Pr_i} \left(\prod_{j=1}^{n} Pr_i(v_i^{(j)}|G_i)\right) p_i(Pr_i|G_i) dPr_i && (6.9)\\
&= \prod_{i=1}^{d} p_{prior,i}(G_i) p_{lh,i}(\mathcal{D}|G_i) && (6.10)\\
&= \prod_{i=1}^{d} p_{G_i}(G_i|\mathcal{D}) && (6.11)
\end{aligned}$$

where we write $p_{lh,i}(\mathcal{D}|G_i) = \int_{Pr_i} \left(\prod_{j=1}^{n} Pr_i(v_i^{(j)}|G_i)\right) p_{prior,i}(Pr_i|G_i)dPr_i$ for the marginal likelihood of G_i given the dataset, and $p_{G_i}(G_i|\mathcal{D})$ for the (unnormalized) posterior distribution on G_i. For example, for linear Gaussian models, we can employ the BGe score [95], which is a closed-form expression for the marginal likelihood of a variable given its parent set $p_{lh,i}(\mathcal{D}|G_i)$.

It may seem that the posterior is now a fully factorized distribution, with independent components for each parent set G_i. However, the distribution is defined over the space of directed acyclic graphs, and not all directed graphs; as a result, two parent sets G_i, G_j will still be correlated due to the acyclicity condition. This leads to the posterior being highly complex and multi-modal in general.

6.1.2 Statistical and Causal Uncertainty

Where does the uncertainty over graphs in the posterior come from? Firstly, as we only have a finite dataset $\mathcal{D}$, many graphs could have generated the dataset, albeit with different (positive) likelihoods. The posterior belief is given by the prior belief of a graph, reweighted by the likelihood of the graph given the observed data. We call this *statistical uncertainty.* However, there may be a second source of uncertainty that does not go away even in the limit of infinite data. This occurs when two different graphs have the same marginal likelihood $p(\boldsymbol{V}|G)$, and thus we cannot distinguish them with certainty from observational data. We call this *causal uncertainty.*

In structure learning, this is known as *non-identifiability*[1], and occurs when two different graphs imply the same conditional independences in the data; the graphs form equivalence classes known as Markov equivalence classes (MECs), which cannot be distinguished from any amount of data without additional (e.g. parametric) assumptions. Even for certain parametric families of distributions, such as discrete BNs and linear Gaussian BNs, different graphs G, G' in a MEC can generate the same observed distributions i.e. $p(\boldsymbol{V}|G) \equiv p(\boldsymbol{V}|G')$. Of course, if we were only interested in the observational distribution that the graphs induced, then this would not be a problem. However, causal quantities such as interventional probabilities differ between members of an equivalence class.

This is an example of the more general problem of *model identifiability* in frequentist statistics [21], where non-identifiability precludes us from determining the correct model. In Bayesian statistics, however, non-identifiability is not, conceptually, an issue, as the posterior distribution is well-defined regardless of whether the likelihood may be the same for different hypotheses (e.g. graphs) [106, 194]. This is because the prior provides information on hypotheses even when the likelihood is (partially) uninformative. For example, given infinite data, a Bayesian posterior would assign weight to each graph in a MEC proportional to their weights in the prior. However, this comes at two practical costs. Firstly, the posterior can become highly reliant on the information contained within the prior. Secondly, even when it is not important to distinguish between equivalent models that have different parameterizations, this can lead to very multi-modal posteriors (i.e. posteriors with multiple local maxima), which are notoriously difficult to infer; this has been observed in Bayesian mixture models [175, 118], and Bayesian neural networks [82], to name a couple of examples.

For the first problem, a recent study [55] examined a number of possible prior choices for the Bayesian structure learning problem, finding that sparsity favouring priors (i.e those which assign lower probability to DAGs with more edges) strike a good balance in practice, assigning greater probability to DAGs with smaller in-degrees (number of incoming edges to a variable) a priori but also demonstrating the ability to

[1]Note that in causal structure learning, identifiability is typically used to mean identifiability of the causal graph, rather than identifiability of a specific causal inference query given the graph as is the case in causal inference.

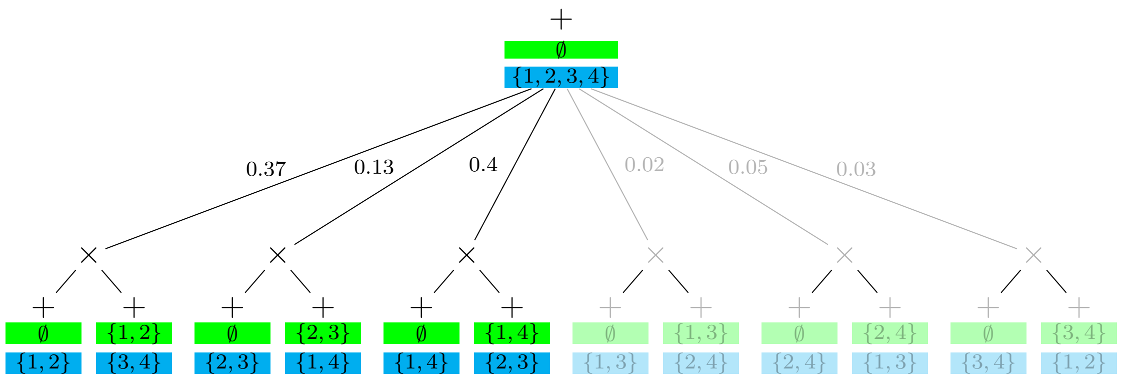


Figure 6.2: Width-limited approximation to posterior distribution

adapt to larger in-degrees given enough data. In particular, we will use the *Fair* prior [62], which assigns $p_{prior}(G) \propto \prod_{i=1} 1/\binom{d}{|G_i|}$, where $\binom{d}{|G_i|}$ is a binomial coefficient, and G_i is the *set of parents of node* V_i. Intuitively, this prior is uniform over the *number of parents* that a node has. As for the second problem of multi-modal posteriors, we will introduce in the next sections a hierarchical decomposition and approximation of the posterior that enables us to efficiently express these multi-modal distributions.

6.1.3 Hierarchical Conditional Independences

In this chapter, we consider Bayesian structure learning over the joint space of topological orders and DAGs, where each order σ is a permutation of $\{1, ..., d\}$. Let $\sigma^{<i}$ be the set of variables preceding variable i in σ. We say that a parent set G_i is consistent with an order σ if $G_i \subseteq \sigma^{<i}$, and that graph G is consistent if all of its parent sets are consistent (written $G \models \sigma$). It follows that any DAG is consistent with at least one order, and further any directed graph consistent with an order must be acyclic. Thus we can specify a joint distribution over orders and DAGs as follows:

$$p(\sigma, G|\mathcal{D}) \propto p_G(G|\mathcal{D})\mathbb{1}_{G\models\sigma} \propto p_{prior}(G)p_{lh}(\mathcal{D}|G)\prod_{i=1}^{d}\mathbb{1}_{G_i\subseteq\sigma^{<i}}$$

Notice that the marginal posterior over graphs $p(G|\mathcal{D}) = \sum_G p(\sigma, G|\mathcal{D})$ is not the same as the original posterior $p_G(\mathcal{D}|G) \propto p_{prior}(G)p_{lh}(\mathcal{D}|G)$, as p will assign higher relative probability graphs which are consistent with more orders as compared to p_G. This imparts a bias for learning with respect to p_G. On the other hand, the space of orders is much smaller than the space of DAGs, enabling more efficient exploration of the distribution [62]. In the case where the prior and marginal likelihood are modular, the resulting distribution p is said to be *order-modular*. In this case, $p_G(G)$ factorizes as $p_G(G) = \prod_i p_{G_i}(G_i)$, giving:

$$p(\sigma, G) \propto p_G(G)\mathbb{1}_{G \models \sigma} = \prod_{i=1}^{d} p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq \sigma^{<i}} \tag{6.12}$$

where we have now omitted the dependence on the dataset and write $p(\sigma, G)$ for the Bayesian posterior. Unfortunately, the representation of the order-modular distribution in Equation 6.12 is not *tractable*: we cannot easily sample from it, nor can we efficiently deduce, for instance, the marginal probability of a given edge. Our goal is thus to obtain a representation approximating this distribution which does possess tractability properties. The key idea is that by exploiting exact conditional independences (CIs) in the distribution, we can *hierarchically* break the approximation of the original distribution into smaller subproblems.

To illustrate this, we first define some notation. Given any subset $S \subseteq \{1, ..., d\}$ of the variable indices, let σ_S denote an ordering (permutation) over S, and $G_S \triangleq \{G_i : i \in S\}$ denote the set of parent sets for each variable V_i for $i \in S$. Now, suppose we partition the set of indices $\{1, ..., d\}$ into two subsets (S_1, S_2), and consider conditioning on the event that all indices in S_1 come before S_2 in the ordering, that is, the order partitions as $\sigma = (\sigma_{S_1}, \sigma_{S_2})$. In this case, the conditional distribution can be written as:

$$\begin{aligned} p(\sigma, G \mid \sigma = (\sigma_{S_1}, \sigma_{S_2})) &\propto \prod_{i=1}^{d} p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq (\sigma_{S_1}, \sigma_{S_2})^{<i}} && (6.13) \\ &= \prod_{i \in S_1} p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq \sigma_{S_1}^{<i}} \prod_{i \in S_2} p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq S_1 \cup \sigma_{S_2}^{<i}} && (6.14) \end{aligned}$$

In other words, conditioning on the order partition, variables corresponding to indices in S_1 can only (potentially) have parents also in S_1, while variables corresponding to indices in S_2 can have parents in either S_1 or S_2. Notice that the distribution has factorized into two terms, which respectively mention only (σ_{S_1}, G_{S_1}) and (σ_{S_2}, G_{S_2}). This motivates defining the following function over (σ_{S_2}, G_{S_2}), which intuitively is the (unnormalized) density of (σ_{S_2}, G_{S_2}) given that S_1 (and only S_1) can be parents of variables in S_2:

Definition 6.1 (Partial Distributions). *Given any disjoint subsets $S_1, S_2 \subseteq \{1, ..., d\}$, we define the partial distribution:*

$$\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}) \triangleq \prod_{i \in S_2} p_{G_i}(G_i) \mathbb{1}_{G_i \subseteq S_1 \cup \sigma_{S_2}^{<i}} \tag{6.15}$$

The full posterior $p(\sigma, G)$ corresponds to a special case of this definition where $S_1 = \emptyset$ and $S_2 = \{1, ..., d\}$; then $p(\sigma, G) = \tilde{p}_{\emptyset,\{1,...,d\}}(\sigma, G)$. However, this generalized definition of partial distributions allows us to additionally express the factorization above purely in terms of these partial distributions:

$$p(\sigma, G \mid \sigma = (\sigma_{S_1}, \sigma_{S_2})) = p_{\emptyset,\{1,...,d\}}(\sigma, G \mid \sigma = (\sigma_{S_1}, \sigma_{S_2})) \tag{6.16}$$

$$\propto \tilde{p}_{\emptyset,S_1}(\sigma_{S_1}, G_{S_1}) \tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}) \tag{6.17}$$

Thus, we have shown that the distribution on orders and graphs (σ, G) can be decomposed into a product of two distributions on (σ_{S_1}, G_{S_1}) and (σ_{S_2}, G_{S_2}) respectively, conditional on $\sigma = (\sigma_{S_1}, \sigma_{S_2})$, i.e. the event that all indices in S_1 come before S_2 in the ordering. Using these conditional independences, we can exactly represent the order-modular posterior, by summing over mutually exclusive partitions of the variable indices $\{1, ..., d\}$. For example, when $d = 4$, we have that:

$$p(\sigma, G) = \sum_{\substack{S_1 \in \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\} \\ S_2 = \{1,2,3,4\} \backslash S}} p(\sigma = (\sigma_{S_1}, \sigma_{S_2})) p(\sigma, G \mid \sigma = (\sigma_{S_1}, \sigma_{S_2})) \tag{6.18}$$

$$= \sum_{\substack{S_1 \in \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\} \\ S_2 = \{1,2,3,4\} \backslash S}} p(\sigma = (\sigma_{S_1}, \sigma_{S_2})) \tilde{p}_{\emptyset,S_1}(\sigma_{S_1}, G_{S_1}) \tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}) \tag{6.19}$$

In Figure 6.2, we represent this expression as a decomposable, smooth and deterministic probabilistic circuit, where each sum node is labelled with its associated S_1, S_2. The weights of the edges then correspond to the probability of each partition $p(\sigma = (\sigma_{S_1}, \sigma_{S_2}))$. As we will later see, we can exploit these circuit properties to tractably reason about posterior over graphs. In higher dimensions, however, it will be infeasible to represent the posterior in this way; for example, if we consider partitions which split the variables into two equally sized sets, the number of possible partitions is given by $\binom{d}{d/2}$. On the other hand, many of these partitions will have very low proability under the posterior. This motivates the idea of using a *width-limited* approximation, where we only keep a subset of the possible partitions.

The second component that enables us to scale to higher dimensions is to *hierarchically* decompose the posterior distribution. In Equation 6.17, we showed how to decompose the distribution over orders on $\{1, ..., d\}$, to distributions on orders on S_1 and S_2. We now present the generalization of this result, which allows us to decompose $\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})$ for any disjoint sets S_1, S_2. Let us consider arbitrary disjoint subsets $S_1, S_2 \subseteq \{1, ..., d\}$, and the distribution $\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})$. We can apply similar reasoning to conditionally decompose $\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})$ into distributions over S_{21}, S_{22}, where S_{21}, S_{22} partition S_2, given by the following Proposition:

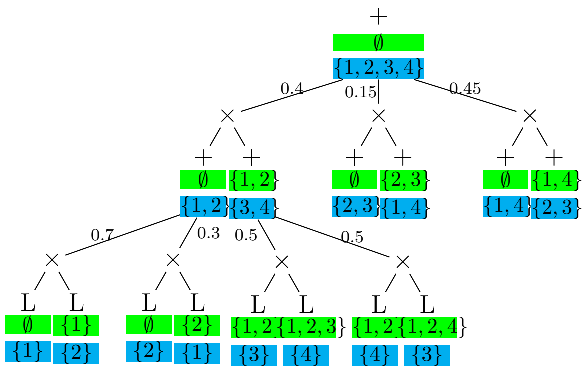
Proposition 6.1 (Hierarchical Conditional Independences)**.** *Let* $p(\sigma, G) \propto p_G(G)\mathbb{1}_{G\models\sigma}$ *be an order-modular distribution. Suppose that* S_1, S_2 *are any disjoint subsets of the variables* $\{1, ..., d\}$*, and let* (S_{21}, S_{22}) *be a partition of* S_2*. Then the following CI holds:*

$$\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}|\sigma_{S_2} = (\sigma_{S_{21}}, \sigma_{S_{22}})) \propto \tilde{p}_{S_1,S_{21}}(\sigma_{S_{21}}, G_{S_{21}})\tilde{p}_{S_1\cup S_{21},S_{22}}(\sigma_{S_{22}}, G_{S_{22}})$$

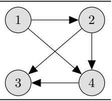
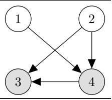
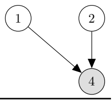
These conditional independencies suggest an approximation strategy: select K partitions (S_1, S_2) of $\{1, ..., d\}$ to form the approximation and, conditional on a partition, then independently approximate the resulting distributions $\tilde{p}_{\emptyset,S_1}(\sigma_{S_1}, G_{S_1}), \tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})$, which are simpler problems of dimensions $|S_1|, |S_2|$, respectively. The decomposition process can be viewed as a rooted tree, where we alternate between nodes that select partitions, and those which decompose the conditional distribution. Using Proposition 6.1, this can be done recursively, until we obtain as a base case distributions where S_2 is a singleton $\{i\}$, where:

$$\tilde{p}_{S_1,\{i\}}(\sigma_{\{i\}}, G_i) = p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1\cup\sigma_{\{i\}}^{<i}} = p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1} \tag{6.20}$$

Note that $\sigma_{\{i\}}$ is an order over a single item, and so can be considered as a constant. This base case then corresponds to the posterior distribution over G_i (parents of variable V_i), conditional on G_i being a subset of S_1. In general, for each $i \in \{1, ..., d\}$, there will be many different base distributions $\tilde{p}_{S_1,\{i\}}(\sigma_{\{i\}}, G_i)$ over G_i, indexed by different sets $S_1 \subseteq \{1, ..., d\} \setminus \{i\}$.



(a) Regular OrderSPN, with expansion factors $\boldsymbol{K} = (3, 2)$. Each sum/leaf node is labelled with its associated (S_1, S_2). Only one expansion beyond the first level is shown for clarity.

(S_1, S_2)	Example Orders	Example Graph
($\emptyset$, $\{1,2,3,4\}$)	**(1, 2, 4, 3)** (2, 3, 1, 4) (4, 1, 3, 2)	
($\{1,2\}$, $\{3,4\}$)	(3, 4) **(4, 3)**	
($\{1,2\}$, $\{4\}$)	**(4)**	

(b) Example orders and graphs for 3 sum/leaf nodes. Graphs only include parent sets of S_2 (filled) variables.

Figure 6.3: Example of regular OrderSPN for $d = 4$. Best viewed in color.

6.2 Tractable Representations for Bayesian Structure Learning

6.2.1 OrderSPNs

Inspired by the above discussion on approximating order-modular posteriors, we now formally propose a class of tractable circuit models, called OrderSPNs, that express distributions over orders, and directed acyclic graphs respecting those orders. OrderSPNs exploit hierarchical conditional independence assumptions (product nodes) in order to represent distributions over orders and graphs more compactly.

Definition 6.2 (OrderSPN)**.** *An OrderSPN $\mathcal{C}$ is a probabilistic circuit over (σ, G), in which each node N is associated with a tuple $\rho(N)$, and which satisfies the following structure:*

- ***Leaf:*** *Each leaf node L is associated with $\rho(L) = (S_1, \{i\})$, for some subset S_1 of $\{1, ..., d\}$ and $i \notin S_1$, and has scope $\phi(L) = (\sigma_{\{i\}}, G_i)$.*

  In addition, the leaf node distribution $p_L(\sigma_{\{i\}}, G_i)$ must have support only over graphs $G_i \subseteq S_1$, i.e. $p_L(\sigma_{\{i\}}, G_i) = 0$ for $G_i \not\subseteq S_1$.

- ***Sum:*** *Each sum node T is associated with $\rho(T) = (S_1, S_2)$, where S_1, S_2 are disjoint subsets of $\{1, ..., d\}$ with $|S_2| > 1$, and has scope $\phi(T) = (\sigma_{S_2}, G_{S_2})$.*

It has K_T children and weights $\omega_{T,i}$ for $i = 1, ..., K_T$, where the i^{th} child is a product node P associated with $\rho(P) = (S_1, S_{21,i}, S_{22,i})$ for some partition $(S_{21,i}, S_{22,i})$ of S_2.

- ***Product:*** *Each product node P is associated with $\rho(P) = (S_1, S_{21}, S_{22})$, where S_1, S_{21}, S_{22} are disjoint subsets of $\{1, ...d\}$, and has scope $\phi(P) = (\sigma_{S_{21} \cup S_{22}}, G_{S_{21} \cup S_{22}})$, where $\sigma_{S_{21} \cup S_{22}} = (\sigma_{S_{21}}, \sigma_{S_{22}})$.*

  It has two children, where the first child is associated with (S_1, S_{21}), and the second with $(S_1 \cup S_{21}, S_{22})$. These children are either sum nodes or leaves.

We can interpret each sum (or leaf) node T associated with (S_1, S_2) as representing a distribution over DAGs over variables S_2, where these variables can additionally have parents from among S_1. In other words, every sum node represents a (smaller) Bayesian structure learning problem over a set of variables S_2 and a set of potential confounders S_1. Each product node then represents a product of independent distributions over DAGs/orders over 1) S_{21}, where the variables can additionally have parents from among S_1; and 2) S_{22}, where the variables can additionally have parents from among $S_1 \cup S_{21}$. The topology of the OrderSPN (in particular, the splits of S_2 into $S_{2,1}$ and $S_{2,2}$ at a product node) induces the distribution over orders.

In practice, we propose to organize the OrderSPN into alternating layers of sum and product nodes, starting with the root sum node. In the j^{th} sum layer, we create a fixed number K_j of children for each sum node T in the layer. Further, for each child i of each sum node T, we choose $(S_{21,i}, S_{22,i})$ such that $|S_{21,i}| = \lfloor \frac{|S_2|}{2} \rfloor$, $|S_{22,i}| = \lceil \frac{|S_2|}{2} \rceil$, and further require that the partitions are distinct for different children i of T. Under these conditions, the OrderSPN will have $\lceil \log_2(d) \rceil$ sum (and product) layers. This allows us to easily control the size of the representation, and further enables efficient tensorized computation over layers. We call such OrderSPNs *regular*, and the associated list $\boldsymbol{K}$ of numbers of children for each layer are called the *expansion factors*. An example of a regular OrderSPN is shown in Figure 6.3. At the top sum layer, we create a child for $K_1 = 3$ different partitions of $\{1, 2, 3, 4\}$ into equally sized subsets, each of which has an associated weight. In the second sum layer, we set $K_2 = 2$ partitions for each sum nodes. Sum and product layers alternate until we reach the leaf nodes.

The leaf nodes L represent distributions over some column of the graph: if L is associated with (S_1, i), then it expresses a distribution over the parents G_i of variable i. The interpretation of S_1 is that this distribution should only have support over sets $G_i \subseteq S_1$. This restriction ensures that OrderSPNs are consistent, in the sense that they represent distributions over valid (σ, G) pairs (in particular, all graphs in the support of an OrderSPN are guaranteed to be acyclic):

Proposition 6.2 (Consistency of Graph and Order). *Let $\mathcal{C}$ be an OrderSPN. Then, for all pairs (σ, G) in the support of $\mathcal{C}$ (i.e. $p_{\mathcal{C}}(\sigma, G) > 0$), it holds that $G \models \sigma$.*

By design, (regular) OrderSPNs satisfy the standard PC properties that make them an efficient representation for inference, which we show in the following Proposition. In the following sections, we will exploit these properties for query computation, as well as for learning the PC parameters.

Proposition 6.3 (Properties of OrderSPNs). *Any OrderSPN is smooth and decomposable, and regular OrderSPNs are additionally deterministic.*

The key advantage of OrderSPNs is in their compactness, or expressive efficiency, obtained by encoding hierarchical conditional independence assumptions. For regular OrderSPNs, we can precisely characterize the compactness of the representation, in terms of the expansion factors.

Proposition 6.4 (Compactness of OrderSPNs). *Given a regular OrderSPN $\mathcal{C}$ over $d = 2^l$ variables, with l sum (and product) layers and expansion factors $(K_0, ...K_{l-1})$ as above, then we have that:*

- *The size (number of edges) of $\mathcal{C}$ is given by: $\sum_{i=1}^{l}(2^i + 2^{i-1}) \prod_{j<i} K_j$*
- *The size (number of orders) of the support of $\mathcal{C}$ is given by: $\prod_{i=0}^{l-1} K_i^{2^i}$*

For example, if we used the same expansion factor K for all layers, then the size of the circuit is $O(K^l)$, while the size of the support is $O(K^d)$. In other words, OrderSPNs can be *exponentially* more compact than maintaining a list/sample of orders. Unfortunately, K^d is still much less than the super-exponential number of possible orders, meaning that practically sized OrderSPNs will only have support on a small subset of possible orders. We will later use OrderSPNs as a variational family for approximating the posterior distribution. The limited support of the OrderSPN is the price that is paid for tractability and respecting the acyclicity constraint; we will soon see how, compared to other intractable variational families for the Bayesian structure learning problem [110, 40], this leads to advantages both in terms of reasoning and learning.

6.2.2 Reasoning on Leaf Distributions

We now have a class of models, OrderSPNs, for modelling uncertainty over causal graphs. Further, OrderSPNs satisfy the properties of decomposability, smoothness, and determinism, which are sufficient for tractable marginal and MAP inference in PCs. However, this overlooks the assumption in PCs that reasoning on the functions represented by the leaf nodes is tractable (in constant time). In the case of OrderSPNs, each leaf node represents a different distribution on the parent set of a variable, and it is not at all obvious that e.g. marginal or MAP queries on these distributions can be computed efficiently. Thus, we investigate in this section how to implement these distributions efficiently.

Given a leaf node L associated with $\rho(L) = (S_1, i)$, corresponding to a distribution on G_i, the definition of OrderSPNs imposes the restriction that the distribution must have support only on graphs $G_i \subseteq S_1$. The question is then how to select a leaf node distribution that best approximates the posterior over orders/graphs, within these constraints. Given that we are approximating an order-modular distribution $p(\sigma, G) \propto \prod_i p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq \sigma^{<i}}$, the natural choice of (unnormalized) leaf distributions is:

$$p_L(G_i) = p_{S_1,\{i\}}(G_i) = \frac{p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq S_1}}{\sum_{G_i \subseteq S_1} p_{G_i}(G_i)} \tag{6.21}$$

That is, we simply reallocate the weight of the disallowed parent sets G_i, in proportion to the probability of each of the allowed parent sets. We will provide formal justification for this choice in Proposition 6.7 when deriving the evidence lower-bound (ELBO) for the OrderSPN as a variational approximation to the posterior.

Unfortunately, as the dimension d increases, evaluating the density $p_{S_1,\{i\}}(G_i)$ is computationally challenging due to the (exponential) sum over subsets in the normalizing constant. Further, different leaves of the OrderSPNs (for the same variable i) will in general have different sets S_1, due to the conditions on variable ordering imposed by the OrderSPN structure. Thus, following previous work [62, 96], we globally limit the parents of variable i to a *candidate set* C_i. That is, for each leaf node for variable i with distribution $p_{S_1,\{i\}}(G_i)$, we replace S_1 with $S_1 \cap C_i$. While this inevitably restricts the coverage of the distribution over DAGs, we can choose the candidate parents C_i in such a way as to preserve as much posterior mass as possible[2]. In contrast to previous work, we are interested not just in the densities/normalizing constants, but also more complex forms of inference; as we will shortly see, this restriction also enables us to design precomputation schemes that then allow for these inference queries on $p_{S_1,\{i\}}(G_i)$ to be answered efficiently for *any* $S_1 \subseteq C_i$.

In order to be able to compute inference queries on the overall distribution over (σ, G) efficiently, we require that the corresponding inference queries for the leaf distributions are tractable. In particular, we will be interested in three types of tasks: marginal/conditional inference, MAP inference, and (conditional) sampling. To formalize this, let $a_{i,j}$ be a Boolean variable indicating whether $j \in G_i$, i.e. j is a parent of i. Further, let c_i be any logical conjunction of the corresponding positive or negative literals, i.e. $a_{i,j}$ or $\neg a_{i,j}$. For instance, $c_i = a_{i,0} \wedge a_{i,1} \wedge \neg a_{i,2}$ represents the event that $0, 1$ are parents of i, but not 2. Then, the task of marginal inference is to evaluate the probability $p_{S_1,\{i\}}(c_i = 1)$. Conditional inference is the task of $p_{S_1,\{i\}}(c_i = 1 | c_i' = 1)$ for two conjunctions c_i, c_i'. MAP inference is $\max_{G_i} p_{S_1,\{i\}}(G_i | c_i = 1)$, while conditional sampling is the task of sampling from $p_{S_1,\{i\}}(G_i | c_i = 1)$.

The key component is to precompute the following function, previously proposed in the Appendix of [188] (for a different purpose):

$$f_i(A_i, A_i') = \sum_{G_i \models \left(\bigwedge_{j \in A_i} a_{i,j} \wedge \bigwedge_{j \in A_i'} \neg a_{i,j}\right)} p_{G_i}(G_i) \tag{6.22}$$

where A_i, A_i' are disjoint subsets of C_i. Intuitively, this is the (unnormalized) probability that all variables in A_i are parents of i, and all those in A_i' are not parents of i.

[2][188] studied a number of different strategies for selecting these candidate parents; we use the *Greedy* heuristic, which was found empirically to be most effective.

This function can be precomputed in time and space $O(3^{|C_i|})$ as follows. In the base case where A_i, A'_i partition C_i, then we simply have:

$$f_i(A_i, A'_i) = p_{G_i}(A_i) \tag{6.23}$$

since A_i, A'_i fully specify the parents of i. In any other case, we have the recurrence:

$$f_i(A_i, A'_i) = f_i(A_i \cup \{b\}, A'_i) + f_i(A_i, A'_i \cup \{b\}) \tag{6.24}$$

for any $b \in C_i \setminus (A_i \cup A'_i)$. This can be seen from the definition of f_i; the RHS corresponds to conditioning on the cases where b either is or is not a parent of i. Notice that, for each A_i, A'_i, we need just a constant-time addition; thus the overall complexity is given by the number of partitions of C_i into three subsets, i.e. $O(3^{|C_i|})$.

Now, let c_i be any conjunction of (positive or negative) literals of the atoms $\{a_{i,j} : j \in C_i\}$, i.e. a partial specification of which edges can and can't be included. We now propose methods for performing marginal, conditional, MAP, and sampling inferences:

- **Marginal/Conditional**: For distribution $p_{S_1,\{i\}}$, the marginal for formula c_i is given by:

$$p_{S_1,\{i\}}(c_i = 1) = \frac{\sum_{G_i \models c_i} p_{G_i}(G_i) \mathbb{1}_{G_i \subseteq S_1}}{\sum_{G_i \subseteq S_1} p_{G_i}(G_i)} = \frac{\sum_{G_i \models \left(c_i \wedge \bigwedge_{j \in C_i \setminus S_1} \neg a_{i,j}\right)} p_{G_i}(G_i)}{\sum_{G_i \models \bigwedge_{j \in C_i \setminus S_1} \neg a_{i,j}} p_{G_i}(G_i)} \tag{6.25}$$

  where we have expressed the condition that $G_i \subseteq S_1$ as the logical formula $\bigwedge_{j \in C_i \setminus S_1} \neg a_{i,j}$. Notice that both the numerator and denominator are of the form of the precomputed f_i, so we can compute the marginal probability simply by two lookups, i.e. $O(1)$ per query.

  Any conditional probability $p_{S_1,\{i\}}(c_i = 1 | c'_i = 1)$ can be computed from marginals as $p_{S_1,\{i\}}(c_i = 1 | c'_i = 1) = \frac{p_{S_1,\{i\}}(c_i \wedge c'_i = 1)}{p_{S_1,\{i\}}(c'_i = 1)}$.

- **MAP**: For distribution $p_{S_1,\{i\}}$, the MAP for formula c_i is given by:

$$p_{S_1,\{i\}}(c_i = 1) = \max_{G_i} p_{S_1,\{i\}}(G_i | c_i = 1) = \max_{G_i \models c_i} p_{S_1,\{i\}}(G_i | c_i = 1) \tag{6.26}$$

$$= \frac{\max_{G_i \models c_i \wedge \bigwedge_{j \in C_i \setminus S_1} \neg a_{i,j}} p_{G_i}(G_i)}{\sum_{G_i \models c_i \wedge \bigwedge_{j \in C_i \setminus S_1} \neg a_{i,j}} p_{G_i}(G_i)} \tag{6.27}$$

The maximum is over G_i satisfying a logical conjunction, similarly to how f_i expresses sums over G_i satisfying logical conjunctions. Thus, we propose to precompute another function $f_i^{\max}$, which is entirely similar to f_i except that the recurrence is given by:

$$f_i^{\max}(A_i, A_i') = \max(f_i^{\max}(A_i \cup \{b\}, A_i'), f_i^{\max}(A_i, A_i' \cup \{b\})) \tag{6.28}$$

Analogously to f_i, $f_i^{\max}$ computes the maximal probability $p_{G_i}(G_i)$ for all G_i satisfying the logical formula. Thus, once this function is precomputed, we can compute any MAP query through a lookup of $f_i^{\max}$ and a lookup of f_i, i.e. $O(1)$ per query.

- **(Conditional) Sampling**: Given the condition c_i, we would like to sample G_i from $p_{S_1,\{i\}}(G_i|c_i = 1)$. Let $B \subseteq C_i$ contain the variables that c_i does not specify (as either definitely being a parent, or definitely not being a parent).

  Then, given any ordering $b_1, ...b_K$ of the elements of B, we can sample whether b_k is present sequentially. When sampling b_k, let $d_i^{(k)}$ be a conjunction formula representing the sampling of $b_1, ...b_{k-1}$, e.g. $d_i = a_{i,b_1} \wedge \neg a_{i,b_2} \wedge ... \wedge \neg a_{i,b_{k-1}}$. Then we have:

$$p_{S_1,\{i\}}(a_{b_k} = 1|d_i^{(k)} = 1, c_i = 1) = p_{S_1,\{i\}}(a_{b_k} = 1|d_i^{(k)} \wedge c_i = 1) \tag{6.29}$$

  This takes the form of a conditional probability, which we can compute in constant time. We must apply this operation $K = O(|C_i|)$ times, which leads to an overall complexity of $O(|C_i|)$ per sampling query.

To summarize, we can perform a precomputation with $O(d3^{|C_i|})$ time and space complexity to obtain the functions/tables f_i, f_i^{max}, after which all of these queries require just a $O(1)$ lookup, except (conditional) sampling, which takes time $O(|C_i|)$. The tractability of inference queries on the leaf nodes of the OrderSPN is crucial to enable tractability of queries for the OrderSPN as a whole, which we investigate in the next section.

6.2.3 Tractable Queries on OrderSPNs

We now turn to reasoning over the OrderSPN as a whole. In Table 6.1, we show a collection of queries, together with their time and space complexity (where M is the size of the OrderSPN). The first four queries correspond purely to the distribution over orders and graphs $p_{\mathcal{C}}(\sigma, G)$ represented by the OrderSPN. The algorithms (and complexity) for these queries are a simple application of standard inference routines on decomposable, smooth and deterministic probabilistic circuits. On the other hand, the last two involve reasoning about the joint distribution over graphs and variables, and in particular *causal reasoning* about the effect of interventions. In the following, we describe how to compute each of these queries, together with their interpretation in the context of structure learning, when the OrderSPN is an approximation to the posterior. Below we will write $p_{\mathcal{C}}(G) := \sum_{\sigma} p_{\mathcal{C}}(\sigma, G)$ to denote the marginal of G in $p_{\mathcal{C}}(\sigma, G)$.

Marginal and conditional inference Let c_i, c_i' be conjunctions over the graph column G_i. Then the *marginal inference* problem is to compute $p_{\mathcal{C}}(\bigwedge_{i=1}^{d} c_i)$. This can be interpreted as the probability of any arbitrary combination of edges (direct causal relations) simultaneously being present. As marginal/conditional inference on decomposable and smooth PCs is tractable in linear time, and since marginal inference for the individual leaves requires just a constant-time lookup, the overall complexity is $O(M)$.

MAP inference MAP inference is the problem of finding the most likely instantiation of the variables, given some evidence $\bigwedge_{i=1}^{d} c_i$. More precisely, we wish to compute $\max_{\sigma, G} p_{\mathcal{C}}(\sigma, G | \bigwedge_{i=1}^{d} c_i)$, which allows us to, for instance, find the most likely extension of a partially specified DAG. This is tractable in linear time for regular OrderSPNs as they are deterministic. As MAP inference on the individual leaves requires just a constant-time lookup, the overall complexity is once again $O(M)$. Note, however, that computing $\max_G p_{\mathcal{C}}(G | \bigwedge_{i=1}^{d} c_i)$ (i.e. with order marginalized out) is not tractable. This would require the sum nodes to be marginally deterministic with respect to the

Query	**Time Complexity**	**Space Complexity**
Marginal/Conditional	$O(M)$	$O(M)$
MAP	$O(M)$	$O(M)$
Sampling	$O(d^2)$	$O(d^2)$
Conditional Sampling	$O(d^2 + M)$	$O(d^2 + M)$
(Interventional) Data Density	$O(dM)$	$O(M)$
Linear Pairwise Causal Effects	$O(d^3M)$	$O(d^2M)$

Table 6.1: Per-query complexity for OrderSPNs, for d variables and OrderSPN of size M

graph variables, which does not hold for OrderSPNs (for example, the graph with no edges is consistent with any order).

Sampling Unconditional sampling from the OrderSPN is straightforward and efficient; we traverse the circuit top-down, randomly choosing one child of each sum-node in proportion to its weight, and all children of each product-node, until we reach the leaf nodes, taking linear time in d. Coupled with the cost of sampling the leaf-node distributions, the overall complexity is $O(d \max_i |C_i|)$ per sample. Conditional sampling is more involved, and requires an $O(M)$ bottom-up computation which updates the OrderSPN weights/probabilities according to the evidence, before sampling via top-down traversal [186].

Data Density We now turn to the computation of other types of queries specific to the causal reasoning setting. Whereas the previous queries were concerned entirely with the posterior distribution over graphs (approximated by the OrderSPN), here we are concerned with the *posterior predictive* distribution over the variables, defined by:

$$p(\boldsymbol{V}|\mathcal{D}) = \sum_G p(G|\mathcal{D}) p_{lh}(\boldsymbol{V}|G) = \mathbb{E}_{G \sim p(G|\mathcal{D})}[p_{lh}(\boldsymbol{V}|G)] \tag{6.30}$$

where $p(G|\mathcal{D})$ is the posterior over graphs, and $p_{lh}(\boldsymbol{V}|G)$ is the marginal likelihood. Given the OrderSPN $p_{\mathcal{C}}(G)$ as an approximation to $p(G|\mathcal{D})$, we can approximate the posterior predictive as:

$$p_{\mathcal{C}}(\boldsymbol{V}|\mathcal{D}) := \sum_G p_{\mathcal{C}}(G) p_{lh}(\boldsymbol{V}|G) \tag{6.31}$$

Recall that the marginal likelihood $p_{lh}(\boldsymbol{V}|G)$ takes the form $p_{lh}(\boldsymbol{V}|G) = \prod_i p_{lh,i}(V_i|G_i)$. This factorized distribution enables us to express the posterior predictive of any sum node in the OrderSPN, and any product node in the OrderSPN, as a function of the posterior predictives of their children. More formally, for any node N in the OrderSPN with scope $\phi(N) = (\sigma_S, G_S)$, define $p_N(\boldsymbol{V}|\mathcal{D}) := \sum_{G_S} p_N(G_S) \prod_{i \in S} p_{lh,i}(V_i|G_i)$.

Now, let T be a sum node associated with $\rho(T) = (S_1, S_2)$, and with children $N_1, ...N_K$ with corresponding weights $\omega_1, ..., \omega_K$. Then we have that:

$$p_T(\boldsymbol{V}|\mathcal{D}) = \sum_{G_{S_2}} p_T(G_{S_2}) \prod_{i \in S_2} p_{lh,i}(V_i|G_i) = \sum_{G_{S_2}} \sum_{i=1}^{k} \omega_i p_{N_i}(G_{S_2}) \prod_{i \in S_2} p_{lh,i}(V_i|G_i) \tag{6.32}$$

$$= \sum_{i=1}^{k} \omega_i \sum_{G_{S_2}} p_{N_i}(G_{S_2}) \prod_{i \in S_2} p_{lh,i}(V_i|G_i) = \sum_{i=1}^{k} \omega_i p_N(\boldsymbol{V}|\mathcal{D}) \tag{6.33}$$

For product nodes P associated with $\rho(P) = (S_1, S_{21}, S_{22})$, where the first child N_1 is associated with $\rho(N_1) = (S_1, S_{21})$ and the second N_2 is associated with $\rho(N_2) = (S_1 \cup S_{21}, S_{22})$, we have that:

$$p_P(\boldsymbol{V}|\mathcal{D}) = \sum_{G_{S_2}} p_P(G_{S_2}) \prod_{i \in S_2} p_{lh,i}(V_i|G_i) = \sum_{G_{S_2}} p_{N_1}(G_{S_{21}}) p_{N_2}(G_{S_{22}}) \prod_{i \in S_2} p_{lh,i}(V_i|G_i) \tag{6.34}$$

$$= \sum_{G_{S_{21}}} \sum_{G_{S_{22}}} \left(p_{N_1}(G_{S_{21}}) \prod_{i \in S_{21}} p_{lh,i}(V_i|G_i) \right) \left(p_{N_2}(G_{S_{22}}) \prod_{i \in S_{22}} p_{lh,i}(V_i|G_i) \right) \tag{6.35}$$

$$= p_{N_1}(\boldsymbol{V}|\mathcal{D}) p_{N_2}(\boldsymbol{V}|\mathcal{D}) \tag{6.36}$$

This means that we can compute the posterior predictive for a circuit by performing a standard evaluation of the circuit. The caveat is that we need to be able to compute the leaf posterior predictives given by $p_L(\boldsymbol{V}|\mathcal{D}) = \sum_{G_i} p_L(G_i) p_{lh,i}(V_i|G_i) = \mathbb{E}_{p_L}[p_{lh,i}(V_i|G_i)]$ when $\rho(L) = (S_1, \{i\})$. Unfortunately, it is not possible to compute this exactly in constant time. Instead, we propose to approximate the expectation with a Monte-Carlo estimate by sampling parent sets from p_L. As each sample from a leaf node takes time $O(|C_i|)$, the overall time complexity is $O(dM)$ (assuming that the sample size is held constant).

The above derivation concerned evaluating the posterior predictive *observational* distribution. However, it is straightforward to extend the method to posterior predictive *interventional* distributions. For example, if we wanted to intervene to set a variable V_i to a specific value v_i', then we can compute the posterior predictive *interventional* distribution using the same method as above, but replacing the evaluation of $p_L(\boldsymbol{V}|\mathcal{D})$ with $\mathbb{1}_{V_i=v_i'}$ for any leaf node with $\rho(L) = (S_1, \{i\})$. This reflects the fact that we have "fixed" the mechanism for V_i by intervention.

In other words, we can tractably evaluate interventional probabilities/densities for a complete instantiation of $\boldsymbol{V}$, averaged over the (approximate posterior) distribution over graphs given by the OrderSPN. Given this, we might wonder whether it is possible to evaluate more complex quantities that involve reasoning over the interventional distribution, such as interventional marginals. Unfortunately, this is not possible in general, for similar reasons to those we discussed in the previous chapter; namely, that the same probabilistic model cannot be tractable for all causal graphs. However, it turns out that there is a specific case where we can tractably reason about pairwise causal effects, which we explain next.

Causal effects In linear Gaussian Bayesian networks the distribution is given by the structural equation $\boldsymbol{V} = \boldsymbol{V}B + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \text{Gaussian}(\boldsymbol{b}, \Sigma)$ represents the Gaussian noise. Here, the parameters of the Bayesian network consist of the edge coefficients $B \in \mathbb{R}^{d\times d}$, the bias term $\boldsymbol{b} \in \mathbb{R}^d$, and the diagonal matrix of noise variances $\Sigma \in \mathbb{R}^{d\times d}_{\geq 0}$. For a given DAG G, we have $B_{ij} = 0$ for all i, j such that i is not a parent of j in G.

In this setting, one of the most important quantities for causal inference is pairwise causal effects, first studied in [197] as the"method of path coefficients". In particular, for a given graph G and weights B, the causal effect of V_i on V_j, written $E_{ij}(B)$, is given by summing the weight of all directed paths from i to j, where the weight of a path is given by the product of the weights of the edges along that path. Notice that, in cases where i is not an ancestor of j, $E_{ij}(B) = 0$. Now, a priori, when we do not know the graph or weights, the causal effect is a random variable given by:

$$E_{ij}(B) := \sum_{\pi \in F(\{1,\dots,d\}\setminus\{i,j\})} B_{i,\pi_1} B_{\pi_{|\pi|},j} \prod_{i=1}^{|\pi|-1} B_{\pi_i,\pi_{i+1}} \tag{6.37}$$

where $F(S)$ is the family of all ordered subsets of the variables S.

Pairwise causal effects are often of great practical interest, as they allow us to understand, on average, how increasing or decreasing a variable V_i by intervention (e.g. dosage of a treatment) affects another variable V_j (e.g. patient outcome). For a given causal graph, evaluating this effect is possible in $O(d^3)$ time. However, if we do not know the causal graph, then from the Bayesian perspective, the natural approach is to employ Bayesian model averaging to estimate the causal effect, in the hope that the result will be more robust than if we committed to a single (e.g. most likely) graph. To perform Bayesian model averaging, we take the expectation of E_{ij} with respect to posterior over graphs $G \sim p(G|\mathcal{D})$, and then with respect to the posterior over edge coefficients $B \sim p(B|G, \mathcal{D})$ given the graph.

$$\mathrm{BCE}(i,j) := \mathbb{E}_{G\sim p(G|\mathcal{D})}\left[\mathbb{E}_{B\sim p(B|G,\mathcal{D})}[E_{ij}(B)]\right] \tag{6.38}$$

Given the approximate posterior $G \sim p_{\mathcal{C}}(G)$, we can approximate this as:

$$\mathrm{BCE}(i,j)^{(\mathcal{C})} := \mathbb{E}_{G\sim p_{\mathcal{C}}(G)}\left[\mathbb{E}_{B\sim p(B|G,\mathcal{D})}[E_{ij}(B)]\right] \tag{6.39}$$

Proposition 6.5 (Pairwise Causal Effects). *Given an OrderSPN representation $\mathcal{C}$ of the distribution over DAGs, the matrix of all pairwise Bayesian averaged causal effects $BCE(i,j)^{(\mathcal{C})}$ can be computed in $O(d^3M)$ time and $O(d^2M)$ space, where M is the size of the OrderSPN.*

Proof. Recall that all nodes N in the OrderSPN can be associated with the variable subsets $\rho(N) = (S_1, S_2)$, and represent a distribution over the set of edges G_{S_2} (in the case of product nodes, we define $S_2 = S_{21} \cup S_{22}$). Thus, they also define a distribution over causal effects for any distinct $i \in S_1 \cup S_2$, $j \in S_2$, given by:

$$E_{ij}^{(N)}(B_{S_2}) = \sum_{\pi\in F(S_2\setminus\{j\})} B_{i,\pi_1} B_{\pi_{|\pi|},j} \prod_{i=1}^{|\pi|-1} B_{\pi_i,\pi_{i+1}} \tag{6.40}$$

Notice that this only counts paths from i to j which immediately enter (and stay in) S_2; thus all edges are in G_{S_2}. As such, this causal effect depends only on B_{S_2}. By taking the expectation, we can similarly define Bayesian averaged causal effects for node N:

$$\mathrm{BCE}(i,j)^{(N)} := \mathbb{E}_{G_{S_2}\sim p_N(G_{S_2})}\left[\mathbb{E}_{B\sim p(B_{S_2}|G_{S_2},\mathcal{D})}[E_{ij}^{(T)}(B_{S_2})]\right] \tag{6.41}$$

Given this, we now show how it is possible to decompose the computation of $\text{BCE}(i, j)$ through the structure of the OrderSPN.

Given a sum node T, with children nodes $N_1, .., N_K$ and corresponding weights $\omega_1^{(T)}, ...\omega_K^{(T)}$ we simply have that:

$$\begin{aligned}
\text{BCE}(i, j)^{(T)} &= \mathbb{E}_{G_{S_2} \sim p_T(G_{S_2})} \left[\mathbb{E}_{B \sim p(B_{S_2} | G_{S_2}, \mathcal{D})} [E_{ij}^{(T)}(B_{S_2})] \right] && (6.42) \\
&= \sum_{c=1,...,K} \omega_c^{(T)} \mathbb{E}_{G_{S_2} \sim p_{N_c}(G_{S_2})} \left[\mathbb{E}_{B \sim p(B_{S_2} | G_{S_2}, \mathcal{D})} [E_{ij}^{(N_c)}(B_{S_2})] \right] && (6.43) \\
&= \sum_{c=1,...,K} \omega_c^{(T)} \text{BCE}(i, j)^{(N_c)} && (6.44)
\end{aligned}$$

where we have used linearity of expectations to bring the sum outside.

Given a product node P, then it has two children N_1, N_2, which are associated with variable subsets $\rho(N_1) = (S_1, S_{21})$, $\rho(N_2) = (S_1 \cup S_{21}, S_{22})$, respectively. We now consider three separate cases, depending on where $i \in S_1 \cup S_2, j \in S_2$ are located within the subsets.

- If $i \in S_{22}$, $j \in S_{21}$, then $\text{BCE}(i, j)^{(P)} = 0$ since by construction edges (and by extension paths) from S_{22} to S_{21} are disallowed.

- If $i \in S_1 \cup S_{21}$ and $j \in S_{21}$, or alternatively $i \in S_{22}$ and $j \in S_{22}$, then notice that all paths between i, j must stay within S_{21} or S_{22} respectively, since there are no edges from S_{22} to S_{21}. Thus, we have that $E_{ij}^{(P)} = E_{ij}^{(N_1)}$ or $E_{ij}^{(N_2)}$ (respectively) and
$$\text{BCE}(i, j)^{(P)} = \text{BCE}(i, j)^{(N_1)} \text{ or } \text{BCE}(i, j)^{(N_2)} \tag{6.45}$$

- In the final case, $i \in S_1 \cup S_{21}$ while $j \in S_{22}$. Here we must consider all possible paths between i and j. To do so, we will condition on the last variable in $S_1 \cup S_{21}$

("exit-point") k along a path. Then we have:

$$\begin{aligned}
E_{ij}^{(P)} &= \sum_{\pi \in F(S_2 \backslash \{j\})} B_{i,\pi_1} B_{\pi_{|\pi|},j} \prod_{i=1}^{|\pi|-1} B_{\pi_i,\pi_{i+1}} && (6.46)\\
&= \sum_{k \in F(S_{21})} \left(\sum_{\pi \in F(S_{21} \backslash \{k\})} B_{i,\pi_1} B_{\pi_{|\pi|},k} \prod_{i=1}^{|\pi|-1} B_{\pi_i,\pi_{i+1}} \right) && (6.47)\\
&\qquad \left(\sum_{\pi \in F(S_{22} \backslash \{j\})} B_{k,\pi_1} B_{\pi_{|\pi|},j} \prod_{i=1}^{|\pi|-1} B_{\pi_i,\pi_{i+1}} \right) && (6.48)\\
&= \sum_{k \in F(S_{21})} E_{ik}^{(N_1)} E_{kj}^{(N_2)} && (6.49)
\end{aligned}$$

The last equality follows as the two summations are precisely the causal effects $i \to k$ and $k \to j$ for N_1, N_2, respectively, which correspond to variable subsets (S_1, S_{21}) and $(S_1 \cup S_{21}, S_{22})$. Now, by linearity of expectations, and the independence of $E_{ik}^{(N_1)}, E_{kj}^{(N_2)}$, this gives the matrix multiplication:

$$\mathrm{BCE}(i,j)^{(P)} = \sum_{k \in F(S_{21} \backslash \{i\})} \mathrm{BCE}(i,k)^{(N_1)} \mathrm{BCE}(k,j)^{(N_2)}$$

Finally, we consider the leaf nodes L of the OrderSPN, where $|S_2|= 1$ (say, $S_2 = \{j\}$). In such cases, the causal effect reduces to $E_{ij}^{(L)} = B_{i,j}$, and the expectation is given by:

$$\mathrm{BCE}(i,j)^{(L)} = \mathbb{E}_{G_j \sim p_L(G_j)}[\mathbb{E}_{B_j \sim p(B_j|G_j,\mathcal{D})}[B_{i,j}]]$$

Given the graph column (set of parents) G_j, the posterior distribution of B_j, $p(B_j|G_j, \mathcal{D})$, is given by a multivariate t-distribution [188], and so the inner expectation can be computed exactly for a given G_j. The outer expectation can be approximated using sampling from the leaf distribution. Though this involves sampling, the crucial aspect of our method is that the expectation through the OrderSPN (and thus through different orders) is exact. We will later see in our experiments that this can have a significant impact on the accuracy of the causal effect estimates.

Finally, analysing the complexity, we see that at each node N corresponding to variable subsets (S_1, S_2), we must maintain an array $BCE(i,j)^{(N)}$ for $i \in S_1 \cup S_2, j \in S_2$, i.e. of size $(|S_1|+|S_2|) \times |S_2|< d^2$. Computations at any sum or product node N take linear time in the number of children (outgoing edges) of the node, except for the matrix multiplication at product nodes, which takes $(|S_1|+|S_{21}|) \times |S_{21}|\times|S_{22}|< d^3$ time. At leaf nodes, we have to sample graph columns G_i, which takes $|C_i|< d$ time. Thus, the overall space and time complexity is $O(d^2M)$ and $O(d^3M)$ respectively.

□

The factor of $O(d^3)$ in the time complexity is unsurprising and arises from the inference cost of causal effects in linear Gaussian BNs [93]; the significance is in the linear complexity in the size of the OrderSPN, given that $\text{BCE}(i, j)$ averages over potentially exponentially more DAGs. Intuitively, this is achieved by "summing-out" over different causal (directed) paths between variables i, j at each node.

6.3 Learning OrderSPNs

In this section, we propose a two-stage method for learning regular OrderSPNs from data, which we refer to as Trust (TRactable Uncertainty for STructure learning). In the first stage, we learn the structure of OrderSPN by hierarchically choosing partitions for each sum node. This characterizes the support of the distribution over orders and graphs. In the second stage, we optimize the parameters of the circuit using a variational inference scheme.

6.3.1 Learning OrderSPN structures

In Section 6.1.3, we introduced the idea of a (hierarchical) width-limited approximation to the graph posterior, in which we choose a subset of the possible partitions of a variable set S_2 into S_{21}, S_{22}. In a regular OrderSPN, for each sum node T in layer j we must choose the K_j partitions $(S_{21,i}, S_{22,i})$ of S_2. A reasonable strategy is to choose to keep the partitions (set of orders) which have the greatest posterior probability $p(\sigma_{S_2} = (\sigma_{S_{21}}, \sigma_{S_{22}}))$. However, we cannot actually implement this strategy in practice, as this posterior probability is intractable to compute.

To define the problem more concretely, we need access to an oracle $\mathcal{O}$ which takes as input the dataset $\mathcal{D}$, disjoint sets S_1, S_2, and a number of partitions K, and returns K partitions $(S_{21,i}, S_{22,i})$ of S_2. The goal of the oracle is to maximize coverage of the posterior distribution, i.e. the posterior mass over orders and graphs consistent with the sampled partitions. In practice, we can instantiate the oracle with any structure learning method that can 1) produce a set of high-probability DAGs, rather than a single DAG; and 2) can be modified to produce a DAG over S_2, which can additionally have parents from S_1, for arbitrary sets S_1, S_2. In our experiments, we adapt two recent Bayesian structure learners, DIBS [110] and GADGET [188], which can produce samples of DAGs approximately from the posterior. Given such a method, we can define the oracle by (i) taking K samples of such DAGs; (ii) for each sample, choosing a random ordering consistent with the DAG; and (iii) splitting the ordering into a partition. Each partition $(S_{21,i}, S_{22,i})$ influences the support of the corresponding child of T, by restricting that S_{21} comes before S_{22} in the ordering.

The proposed strategy involves calling the oracle $\mathcal{O}$ for each sum node in the OrderSPN. This improves exploration of the space over the base structure learning method (oracle), by recursively exploring subspaces of DAGs over smaller subsets of variables $S_2 \subseteq \{1, ..., d\}$. However, it also appears to introduce a computational challenge since the number of sum nodes in the circuit could be very large. Thankfully, though each successive sum layer has $2K_j$ times more sum nodes than the previous layer, the dimension of the DAG space is halved, meaning that the oracle requires much less time. In practice, we ensure efficient implementation by the following methods: (i) we set a time budget appropriately for the oracle in each layer, so that the time spent by the oracle in each layer is roughly the same; (ii) for small dimensions $|S_2| \leq d'$ (chosen to be 4), we avoid the constant-time overhead of each oracle run by instead explicitly enumerating over all partitions.

6.3.2 Parameter Learning via Variational Inference

Given an OrderSPN structure, we now consider the task of learning the parameters/weights of the OrderSPN. We formulate this as a discrete variational inference (VI) problem. Given an unnormalized order-modular distribution $\tilde{p}(\sigma, G) = p_G(G)\mathbb{1}_{G \models \sigma}$, the evidence lower bound (ELBO) is given by:

$$\text{ELBO}(\mathcal{C}) = \mathbb{E}_{p_{\mathcal{C}}(\sigma,G)}[\log \tilde{p}(\sigma, G)] + H(p_{\mathcal{C}}(\sigma, G)) \tag{6.50}$$

where $H(p_{\mathcal{C}}(\sigma, G)) = -\mathbb{E}_{p_{\mathcal{C}}(\sigma,G)}[\log p_{\mathcal{C}}(\sigma, G)]$ is the entropy of the OrderSPN $\mathcal{C}$. The goal of VI is then to maximize the ELBO with respect to the OrderSPN parameters ω. Typically, such discrete VI problems are very difficult, since the ELBO requires computing (gradients of) the expectation using high-variance estimators such as REINFORCE, leading to unstable optimization (due to the discrete space, the reparameterization trick is not applicable). However, for OrderSPNs, the ELBO can be computed exactly due to the tractability of the circuit model. The following result is similar to the corresponding result (Thm 1) from [166] for using deterministic PCs as a variational approximation to discrete graphical models, and we defer the full proof to Appendix C.1.

Proposition 6.6 (Tractable ELBO). *The ELBO and its gradients for any regular OrderSPN $\mathcal{C}$ and order-modular distribution p can be computed in linear time in the size of the circuit.*

Aside from showing that the ELBO can be propagated through sum and product nodes, to prove this result we need to be able to compute the ELBO for the leaf node distributions. For a leaf node L associated with (S_1, i), which represents a distribution $p_L(G_i)$ over the parents of variable V_i, the ELBO is defined to be:

$$\begin{aligned}\mathrm{ELBO}(L) &:= \mathbb{E}_{p_L}[\log \tilde{p}(\sigma_{\{i\}}, G_i)] + H(p_L(\sigma_{\{i\}}, G_i)) \\ &= \mathbb{E}_{p_L}[\log p_{G_i}(G_i)] + H(p_L(G_i)) \qquad (6.51)\end{aligned}$$

Recall that, for OrderSPNs, it is required that $p_L(G_i)$ has support only over $G_i \subseteq S_1$. Previously, in Equation 6.21, we chose to set $p_L(G_i) \propto p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq S_1}$. We now provide justification for this choice:

Proposition 6.7 (Optimal Leaf Distribution). *$p_L(G_i) \propto p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq S_1}$ maximizes (6.51) subject to the support condition.*

Proof. The ELBO for a leaf distribution (6.51) can be written as:

$$\begin{aligned}\mathrm{ELBO}(L) &= \mathbb{E}_{p_L}[\log p_{G_i}(G_i)] + H(p_L(G_i))) \\ &= \mathbb{E}_{p_L}[\log p_{G_i}(G_i)] - \mathbb{E}_{p_L}[\log p_L(G_i)] \\ &= -KL(p_L || p_{G_i})\end{aligned}$$

where KL is the KL-divergence. Thus, to maximize the ELBO, we need to minimize this KL-divergence. Let $C = \sum_{G_i \subseteq S_1} p_{G_i}(G_i)$. Assuming L satisfies the support condition, this can be written as:

$$\begin{aligned} KL(p_L || p_{G_i}) &= \mathbb{E}_{p_L}\left[\log \frac{p_L(G_i)}{p_{G_i}(G_i)}\right] \\ &= \mathbb{E}_{p_L}\left[\log \frac{p_L(G_i)}{p_{G_i}(G_i)}\right] \\ &= \mathbb{E}_{p_L}\left[\log \frac{p_L(G_i)}{p_{G_i}(G_i)/C}\right] - \log C \end{aligned}$$

This KL-divergence is minimized by $p_L(G_i) \propto p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq S_1}$, as required. In this case, the ELBO is given by:

$$\begin{aligned} \text{ELBO}(L) &= \log C - KL(p_L || \frac{p_{G_i}\mathbb{1}_{G_i \subseteq S_1}}{C}) \\ &= \log C \end{aligned}$$

□

We see that, with this choice of p_L, the ELBO is a constant $\log C$ that we can precompute using the methods for computation of leaf distribution described in Section 6.2.2. Thus, the computation of ELBO for leaf distributions can be done in an $O(1)$ lookup, and the overall ELBO computation is linear in the size of the OrderSPN (in particular, independent of the dimension). This allows us to learn the weights using gradient-based optimization of the ELBO, leveraging tensor learning frameworks and hardware acceleration.

6.4 Experiments

In this section, we perform an empirical investigation and evaluation of the TRUST framework. In particular, we implement two state-of-the-art Bayesian structure learning methods, (marginal) DIBS [110][3] and GADGET [188], and compare them

[3]There are two variants of DIBS; namely *joint* DIBS and *marginal* DIBS. While joint DIBS is generally more performant, we use marginal DIBS here so that all models are using a marginal likelihood, for fairness of comparison; [150] indicated that joint likelihood might exploit features that are unavailable to (score-equivalent) marginal likelihoods.

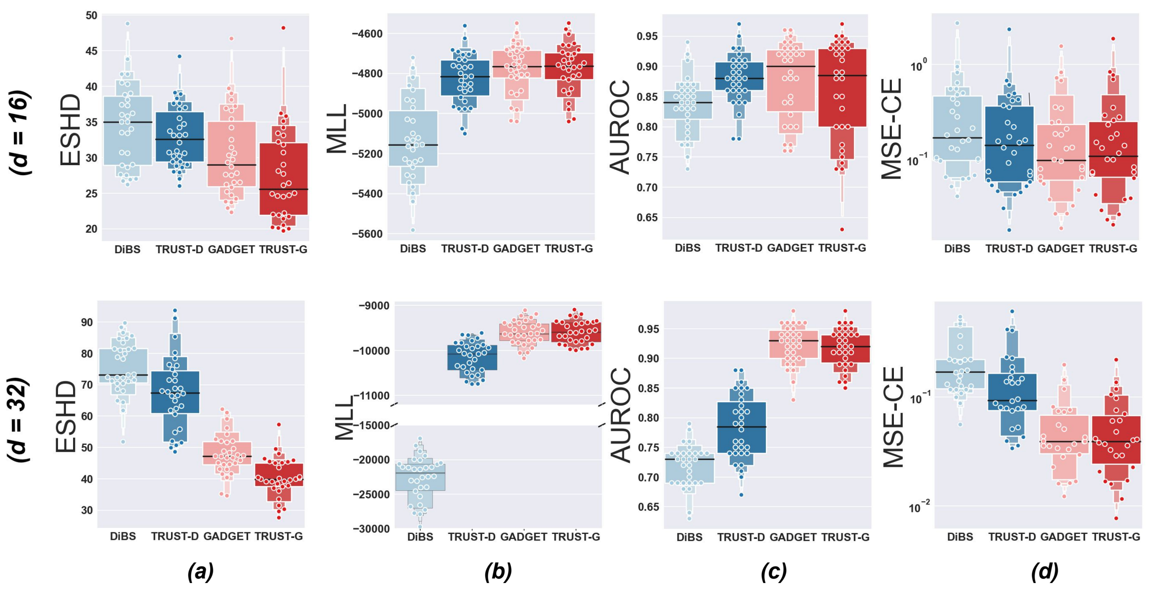


Figure 6.4: Performance evaluation of the Trust framework. We find that across all metrics and for both dimensionalities that the Trust framework outperforms the seed method, in some instances considerably. **Top Row:** Learning structures with $d = 16$. **Bottom Row:** Learning structures with $d = 32$. **(a)** Expected Structural Hamming Distance, lower is better. **(b)** Marginal Log Likelihood (higher is better). **(c)** Area Under the Receiver Operator Characteristic curve (higher is better). **(d)** MSE of Causal Effects (lower is better).

with their Trust-enhanced counterparts, Trust-D and Trust-G, which use the respective method as the oracle for learning the structure of the OrderSPN. We apply each inference method to synthetic structure learning problems, where the ground truth causal structures are Erdős-Rényi random graphs with dimension $d \in \{16, 32\}$ and $2d$ expected edges, and the Bayesian network distribution is linear Gaussian. All methods tested employ the BGe marginal likelihood score [95] for linear Gaussian networks. For each experiment, a dataset $\mathcal{D}^{\text{train}}$ of $N = 100$ datapoints is generated for each graph for inference, in order to capture the regime where there is uncertainty due to limited data. We defer full experimental details to Appendix C.3.

6.4.1 Learning Performance

We begin by evaluating the quality of the inferred posterior $p_{\mathcal{C}}(G)$ for each inference method. Unfortunately, the true posterior $p(G)$ is intractable, meaning that we cannot use it as a reference point to compare. Instead, we use a variety of standard metrics for measuring the posterior quality. As the graphs are synthetically generated, we have access to the true graph and edge coefficients (of the linear Gaussian BN), which we denote with G, B respectively. Further, we write $\mathcal{D}^{\text{test}}$ to denote a held-out dataset of 1000 datapoints generated independently from the training data.

- The *expected structural Hamming distance* E-SHD$(p_{\mathcal{C}}, G)$ measures the expected number of edge changes (SHD) between the essential graphs of G and G', where G' is sampled from the inferred posterior p:

$$\text{E-SHD}(p, G) = \mathbb{E}_{G' \sim p_{\mathcal{C}}}[\text{SHD}(\text{essential}(G'), \text{essential}(G))] \tag{6.52}$$

- The *area under the receiver operating characteristic curve* AUROC(p, G) for Bayesian structure learning [62] is computed using marginal edge probabilities $p(G'_{ij} = 1)$ for each potential edge G'_{ij}, while varying the confidence threshold to construct the ROC curve.

- The *marginal log-likelihood* MLL$(p, G, \mathcal{D}^{\text{test}})$ measures how well the posterior fits the held-out test data [123, 110], using the BGe marginal likelihood p_{lh}:

$$\text{MLL}(p, \mathcal{D}^{\text{test}}) = \mathbb{E}_{G' \sim p}[\log p_{lh}(\mathcal{D}^{\text{test}} | G')] \tag{6.53}$$

- Finally, the *mean-squared error of causal effects* MSE-CE(p, B) measures the squared difference between the expected posterior causal effect BCE$(i, j)^{(p)}$, and the true causal effect $E_{ij}(B)$ (for variable pair i, j) [140]. This is then averaged over all (distinct) pairs i, j:

$$\text{MSE-CE}(p, B) = \frac{1}{d(d-1)} \sum_{i \neq j} |BCE(i, j)^{(p)} - E_{ij}(B)|^2 \tag{6.54}$$

For the sample-based posterior approximations Dibs and Gadget, the expectations over p are computed by explicitly averaging over all samples. For Trust-d and Trust-g, we also sample from $p_{\mathcal{C}}$ to approximate the expectation. However, for AUROC and MSE-CE, we employ exact computation of the expectations (with respect to the OrderSPN distribution $p_{\mathcal{C}}$) using our query answering algorithms for marginal edge probabilities/causal effects respectively.

We show the results in Figure 6.4 for all methods. TRUST-D and TRUST-G match or outperform their counterparts across all metrics, with especially strong performance on E-SHD, where TRUST-G is best by a clear margin for both $d = 16, 32$. This indicates that hierarchical exploration can lead to more accurate discovery, compared to sampling directly on the full space of DAGs. Another interesting phenomenon that we observe is that TRUST-D significantly improves upon the marginal log-likelihood of DIBS, which the authors in [110] hypothesized was due to high variance in their variational inference procedure. This suggests that the exact parameter learning in the second stage of TRUST helps to correct for errors in the relative weighting of samples.

6.4.2 Ablation Study on OrderSPN Learning

In Section 6.3, we proposed to use a two-step procedure for learning OrderSPNs, in which we (i) propose a structure for the OrderSPN using an oracle method; and (ii) further learn the parameters of the OrderSPN via variational inference. We now perform an ablation study to examine each of these steps and their impact on performance. In particular, we evaluate four different versions of TRUST, including TRUST-G, and the following variants:

- **Random** In this case, instead of using an oracle method $\mathcal{O}$ to split S_2 into a partition $(S_{21,i}, S_{22,i})$, we instead perform this split *randomly* throughout the OrderSPN. We also do not perform any parameter learning, instead setting the parameters at each sum-node in the OrderSPN to be equal (e.g. if a sum-node has 4 children, we set each parameter to 0.25).

- **Parameter Only** We randomly propose the structure as above, but do perform parameter learning using VI in the second stage.

- **Structure Only** We do perform structure learning using GADGET as an oracle, but do not learn parameters using VI in the second stage.

The first step of structure learning determines the support of the OrderSPN, i.e. the orders and DAGs to which it assigns positive probability, while the second step of parameter learning aims to optimize the fit to the posterior given the support constraints imposed by the first step. By randomizing one (or both) of these steps, we can see how this affects the approximation.

The results are shown in Figure 6.5. As expected, the fully random method performs by far the worst, on all metrics. Performing parameter learning only and structure learning only provide significant improvements, but interestingly on different metrics. Structure learning only performs quite well on AUROC, while parameter learning only performs comparatively better on E-SHD and MLL (even outperforming Gadget on E-SHD). The performance of using parameter learning only is quite remarkable, given that the graphs covered by the OrderSPN were chosen *at random*. We hypothesize that this can be attributed to the compactness and capacity of OrderSPNs as a representation; as a result, even the randomly chosen OrderSPN structure will contain some orders/DAGs in its support which are close to the ground truth DAG both in structure (E-SHD) and marginal likelihood (MLL). Nonetheless, adding OrderSPN structure learning as well, as in Trust-g, does provide the best overall performance, and shows that both steps are important to obtain the best possible representation.

6.4.3 Exact and Approximate Computation

One of the key features of OrderSPNs is the tractability of the representation, eliminating error from statistical sampling. In particular, we saw that for linear causal models, it is possible to compute the matrix of pairwise causal effects $BCE_q(i,j)$, averaged over the posterior over graphs induced by the OrderSPN. Alternatively, we could sample a set of graphs from the OrderSPN, and obtain an empirical average $\widehat{BCE_q(i,j)}$ of pairwise causal effects. In Figure 6.6, we compare the exact and approximate methods in terms of their mean squared error (MSE-CE) compared to the weight matrix of the true, data-generating graph (averaged over multiple runs/data-generating graphs). For the approximate estimate, we use 10000 graph samples. Along the horizontal axis, we additionally vary the mean of edge coefficients in generating random linear Gaussian BNs. The rationale is that larger weights will lead to larger pairwise causal effects (on average), making the estimation task more challenging. For example, for a

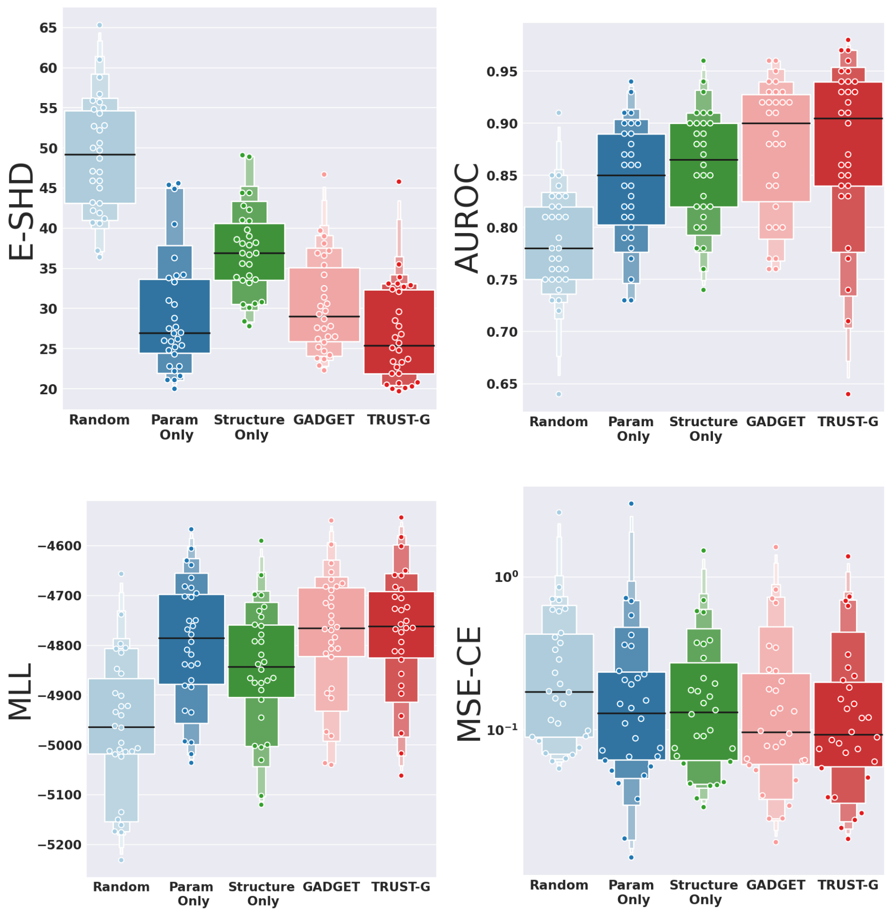


Figure 6.5: Ablation study evaluating performance of different variants of TRUST-G (and GADGET), for $d = 16$.

chain causal graph with an edge $V_i \to V_j$ whenever $j = i + 1$, and $d = 16$, the pairwise causal effect of V_1 on V_{16} would be around 2^{15}, but with significant variance.

The results show that exact computation is much more accurate than approximate computation for all mean edge weights. The difference becomes more pronounced for larger edge weights, with the average error being ~ 100 times less for exact computation with edge weight 2.0. Further, the variance across different runs (i.e. sampled true graphs/edge coefficients) is much smaller, indicating that the performance of exact computation is much more robust to different underlying causal systems.

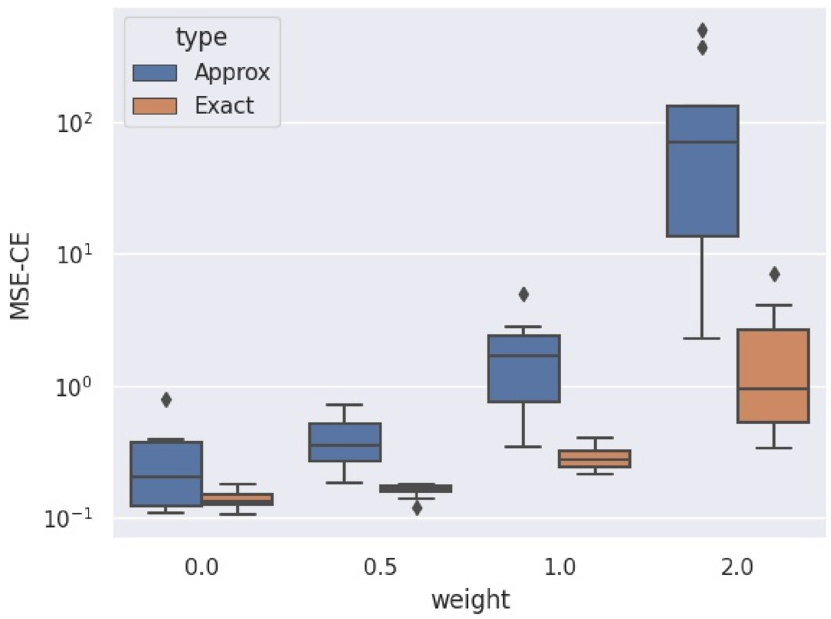


Figure 6.6: MSE-CE for approximate (10000 graph samples) and exact computation of pairwise causal effects, as a function of the mean edge weight magnitude. Data shown is over 10 runs per weight.

6.4.4 Coverage and Query Answering

We now compare the query answering capabilities of Trust to Dibs and Gadget, for $d = 16$ networks. We set up the task by selecting n edges randomly from the true graph, which we use to form the condition $\bigwedge_{i=1}^{d} c_i'$ in Section 6.2.3 (requiring that all n edges are present). A good representation of the posterior should consistently have posterior mass over this condition. For Dibs and Gadget, we obtain sample-based approximations q of the posterior, for which we take 30 and 10000 samples respectively, as indicated by the respective papers and reference implementations. For Trust-d and Trust-g we directly perform the inference queries on the learned OrderSPN.

We begin by considering the marginal probability $p(\bigwedge_{i=1}^{d} c_i')$. In Figure 6.7, we compute this over 30 different runs and 50 random edge selections for each run, for different values of n, and plot the proportion of times that the probability is non-zero. We see that, as n increases, both methods based on Trust consistently outperform their counterparts. This demonstrates how Trust can be used to augment an oracle method to significantly improve the reliability of posterior coverage. This is

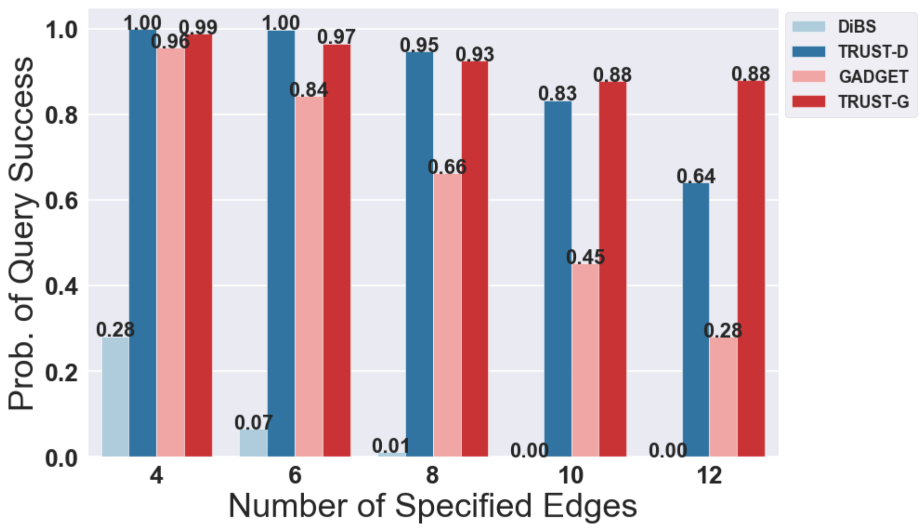


Figure 6.7: As we specify more edges in our query, the probability that sample-based posteriors (DIBS and GADGET) have support over the queried edges drops. TRUST-D and TRUST-G, in contrast, maintain much greater coverage.

particularly noteworthy for DIBS, whose coverage is otherwise limited by its quadratic time complexity in the number of samples.

From a practical perspective, this is especially important for conditional inference. In Table 6.2, we simulate a scenario where we obtain information on the true causal graph after learning. In particular, given $n = 4, 8, 16$ randomly specified edges from the true graph as a condition, we compute conditional probabilities for all unspecified (potential) edges. This can be viewed as "injecting" causal information, which, for instance, could permit distinguishing between DAGs in the same Markov equivalence class where observational data would not suffice. To evaluate, we compute the AUROC given the computed probabilities for each edge. In the case where the representation q has no probability over the condition, we simply take the overall AUROC for the unconditional distribution. Table 6.2 shows mean and standard deviation for AUROC over 30 runs for each method. As the number of specified edges increases, we see that the performance of GADGET degrades despite the extra information, since the sample-based representation suffers from prohibitively high variance when estimating conditional probability. On the other hand, the greater coverage of TRUST-G ensures that we can take advantage of the extra information, improving the quality of inferences.

No. Edges	Method	AUROC
4	Gadget	0.905 ± 0.073
	Trust-g	0.903 ± 0.057
8	Gadget	0.888 ± 0.089
	Trust-g	0.933 ± 0.048
16	Gadget	0.876 ± 0.081
	Trust-g	0.957 ± 0.077

Table 6.2: Quality of inference for conditional queries. Results show Trust-g is significantly better at inferring conditional distributions, especially as the condition becomes more restrictive.

6.5 Discussion

In this Chapter, we studied the problem of causal learning and reasoning when there is uncertainty over the causal graphical structure of the domain. In this setting, we need to consider both learning a distribution over structures from data, as well as reasoning on the resulting joint distribution over graphs and variables. Instead of maintaining a set of DAGs or orders, we introduced OrderSPNs, a new approximate probabilistic representation of distributions over orders and DAGs. We showed that OrderSPNs enable tractable and exact inference over the representation for a variety of important classes of causal reasoning queries, including, remarkably, inference of causal effects for linear Gaussian networks.

In comparison to current approaches to Bayesian structure learning, OrderSPNs can, in some sense, be considered to be the most expressive and general representations upon which we can reason tractably. While it is, of course, possible to approximately fit the posterior using expressive (but intractable) probabilistic models, we can only perform sampling of orders, or graphs, from the model, which provides a bottleneck for downstream reasoning. This is reflected in the empirical results, where learned OrderSPNs not only perform competitively on standard metrics, but can also be used to reason effectively in ways that maintaining a sample of graphs/orders cannot.

A limitation of the approach is the assumption that there is no unobserved confounding (i.e. Markovianity), which is typical in Bayesian structure learning due to the complexity of the problem (with the recent exception of [5]). Extending our Bayesian approach to settings with unobserved confounding has two main challenges. The first is how to specify a reasonable prior over the degree and nature of unobserved confounding; for example, over semi-Markovian causal diagrams with bidirected edges. A promising direction in this respect is to adapt scoring functions for causal models with confounding in score-based causal discovery [181, 11, 12]. The second challenge relates to tractably learning and reasoning about the distribution, given that the exogenous variables are unobserved. Adapting latent-variable methods such as expectation-maximization may be a possible solution.

Another challenge for future work is to consider tractable approximate representations for the leaf node distributions $p_{S_1,\{i\}}(G_i)$. In Section 6.2.2, we showed that under the assumption that the parents of each variable come from a candidate set (of size up to 16), it is possible to precompute all leaf node distributions in a tractable tabular representation in exponential time and space. However, the time and memory requirements are onerous in practice, and the candidate set assumption becomes increasingly untenable as we scale up the dimension. Instead, we could consider modelling the posterior over parents $p_{G_i}(G_i)$ using a tractable probabilistic circuit, as an approximation to the true posterior. The advantage is that conditioning on membership in S_1, and computing marginal/MAP/sampling queries on this distribution remain tractable, while we can freely configure the size of the circuit according to available computational resources. This could also open up the application of the method to models with intractable marginal likelihoods (e.g. non-linear, non-Gaussian noise).

Chapter 7

Conclusions

The implementation of causal reasoning into intelligent systems is a major ongoing research challenge, due to its utility as an abstraction of an underlying physical system that goes beyond purely probabilistic information; its corresponding ability to capture important concepts such as robustness, fairness, and distribution shift that cannot be expressed by purely probabilistic models; and its close connection with how humans reason about the world, making causal reasoning queries naturally interpretable. The conceptual frameworks that have been developed in the last few decades for understanding causality, notably that of the structural causal model and information hierarchy, have elucidated what we can hope to infer about the underlying causal system given the information available to the modeler. This thesis has been concerned with the parallel question of when the computational process of performing this inference, which is often more complex than probabilistic inference, is tractable. We have argued that, rather than building separate models every time we want to answer a causal reasoning query, it is more natural and often more desirable to consolidate all of the information available to the modeler in a unified model, that can then be used to answer any causal query of interest. The key challenge, however, is that structural causal models are not particularly useful for this purpose due to intractable reasoning, as well as a lack of flexibility. We have hypothesized that tractable probabilistic models are a natural fit due to their ability to consistently, efficiently, and exactly reason over probability distributions.

To this end, we identified three different scenarios corresponding to differing states of knowledge of the modeler; in each, we identified conditions (and algorithms) for being able to tractably compute various causal reasoning queries, as well as pushing the boundaries of expressivity/generality of the underlying TPM (circuit) while satisfying these conditions. These theoretical and methodological results provide strong evidence in support of the hypothesis that TPMs can be used (possibly in conjunction with a causal diagram) as unified, tractable causal models. We also hope, more broadly, that the perspective and results of this thesis will help to inspire a new paradigm for causal reasoning in which alternative models/representations of the underlying SCM reality are used for computational, or other, benefits. This is in alignment, for example, with recent work on causal abstraction [155, 2, 64], which aims to make a SCM more interpretable by clustering variables; and in a similar manner to how the field of tractable probabilistic modelling has emerged out of attempts to make Bayesian networks more practical to learn and reason over.

7.1 Discussion

Our primary subject of inquiry in this thesis has been the tractability of causal inference queries on probabilistic circuit models. Answering this question has required a clearer picture of what the PC represents, in the context of the causal information hierarchy. In Chapter 4, we considered the case where the causal model is fully specified and known to the modeler in one of the standard forms (causal Bayesian networks or SCMs). The computational task is then to reason efficiently over this given (graphical) model. Towards our goal of producing tractable models that can answer any causal query efficiently, compiled representations are appealing as they convert the intractable model into a tractable form. We showed that, besides interventional marginals, applying additional (topological) constraints to the compiled circuit structure allowed for the tractability of an even wider range of advanced causal queries. These methods are suitable for (fairly) low-dimensional problems, where each variable and the relationships between them are meaningful, and one needs to extract detailed causal information.

However, compilation is not a silver bullet; the size of the compiled circuit depends on the complexity of reasoning on the underlying CBN/SCM. Thus in Chapter 5 we assumed instead that the modeler had access to a causal diagram (possibly over sets of variables) and observational data. Crucially, rather than requiring that the PC follows (is compiled from) a granular graphical causal model, and then imposing conditions for tractability on top of that assumption, we started from minimal assumptions for causal inference (structured decomposability and smoothness) and systematically derived tractability conditions based on these. The resulting circuits, MDNets, can be freely scaled depending on the desired trade-off between expressivity and size (and thus cost of inference). In Chapter 6, in the setting where the casual diagram was additionally unknown, we applied a similar principle in the design of OrderSPNs, which can be scaled using the expansion factors (number of children of each sum node). These more general, learned circuits can be much more practical as a scalable, black-box causal model that takes in data and returns answers to causal reasoning queries, but are not as reliable or tractable as compiled circuits when we have all information about the underlying SCM.

It is now also apparent that as we decrease the knowledge available to the modeler, the tractability of the representations of that knowledge is lessened. In particular, interventional marginals are almost trivially tractable in compiled circuits, while they are only tractable in certain circumstances given a learned circuit and diagram separately, and not tractable at all (except in the particular linear Gaussian case) when we additionally model diagram uncertainty. This has important implications for knowledge representation of causal models; namely that there is a tradeoff between the flexibility of the representation, and its tractability. As a result, there is not necessarily one representation that is optimal in all cases, but instead we may want to choose different types of representations depending on our level of certainty about the underlying causal structure of the domain.

7.2 Future Work

We now highlight some overall directions for future work based on this thesis.

Firstly, and most directly, the results and discussion in this thesis raise new open questions about the tractability of causal reasoning. In Chapter 5, we showed tractability results for the backdoor, frontdoor, and napkin formulae for probabilistic circuits. This raises the question of whether *any* do-calculus formula or ID algorithm estimand can be computed in polynomial time (in the size of the circuit); and if not (as might be suggested by the restrictive conditions of the MD-calculus), how the class of tractable causal estimands can be characterized. Another important goal is to discover other types of tractable models (circuit-based or otherwise) that are well-suited to causal inference. This may include not only models that are tractable for exact causal inference queries but also models which can produce guaranteed approximations. In view of the results in Chapter 6 regarding causal effects in linear models, it is also worth investigating whether making parametric assumptions can also aid with making more causal queries tractable.

Secondly, an important direction is to exploit the results of this thesis to design practical causal inference algorithms using probabilistic circuits. We have already explored in Chapter 6 the application of probabilistic circuits to causal structure, and showed that it produces competitive results compared to the state-of-the-art. However, for causal inference, the results of Chapter 5 are primarily theoretical in nature, due to the quadratic/cubic complexity in the size of the original circuit (which can be prohibitive in practice). Overcoming this is a key challenge; as is integrating with techniques for improving the statistical performance of causal estimation, such as double machine learning [28].

Finally, an ambitious future direction is to connect the work in this thesis with the nascent field of causal representation learning [159]. Throughout this work, we have assumed that the causal variables (i.e. the variables constituting nodes in an SCM) are given in advance. However, in many practical scenarios this might not be the case; consider, for example, analyzing the causal structure of a video. Attempting to construct a causal diagram or SCM on the space of pixels is hardly feasible, but there may be an underlying causal structure between features of the video frames. Causal representation learning aims to automatically extract these causal variables and relationships from data. Clearly, one can employ representation learning methods as a pre-processing step before reasoning on the obtained variables and graph. However, the resulting distribution over causal variables will not usually be tractable. Perhaps the most promising approach is to integrate the tools and theory developed in this thesis as a "reasoning engine" on the lower dimensional space of causal variables, while offloading the representation learning component to neural networks. Such a synthesis could both aid learning of representations, as well as ensure that the final learned representation can be efficiently and consistently reasoned over.

Bibliography

[1] Tameem Adel, David Balduzzi, and Ali Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. In *Uncertainty in Artificial Intelligence - Proceedings of the 31st Conference, UAI 2015*, 2015.

[2] Tara V. Anand, Adèle H. Ribeiro, Jin Tian, and Elias Bareinboim. Causal Effect Identification in Cluster DAGs. In *AAAI 2023*, 2023.

[3] Yashas Annadani, Jonas Rothfuss, Alexandre Lacoste, Nino Scherrer, Anirudh Goyal, Yoshua Bengio, and Stefan Bauer. Variational Causal Networks: Approximate Bayesian Inference over Causal Structures. *arXiv preprint arXiv:2106.07635*, 2021.

[4] Alessandro Antonucci, Cassio P. De Campos, David Huber, and Marco Zaffalon. Approximate credal network updating by linear programming with applications to decision making. *International Journal of Approximate Reasoning*, 58:25–38, 3 2015.

[5] Matthew Ashman, Chao Ma, Agrin Hilmlkil, Joel Jennings, and Cheng Zhang. Causal Reasoning in the Presence of Latent Confounders via Neural ADMG Learning. In *Proceedings of the Eleventh International Conference on Learning Representations*, 2023.

[6] Chen Avin, Ilya Shpitser, and Judea Pearl. Identifiability of path-specific effects. In *IJCAI International Joint Conference on Artificial Intelligence*, 2005.

[7] Francis R. Bach and Michael I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems*, 2002.

[8] Alexander Balke and Judea Pearl. Probabilistic evaluation of counterfactual queries. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1, 1994.

[9] Elias Bareinboim, Juan D. Correa, Duligur Ibeling, and Thomas Icard. On Pearl's Hierarchy and the Foundations of Causal Inference. In *Probabilistic and Causal Inference*, pages 507–556. 2022.

[10] Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2015, pages 2770–2776, 2015.

[11] Daniel Bernstein, Basil Saeed, Chandler Squires, and Caroline Uhler. Ordering-based causal structure learning in the presence of latent variables. In *International Conference on Artificial Intelligence and Statistics*, pages 4098–4108. PMLR, 2020.

[12] Rohit Bhattacharya, Tushar Nagarajan, Daniel Malinsky, and Ilya Shpitser. Differentiable Causal Discovery Under Unmeasured Confounding. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2314–2322. PMLR, 2021.

[13] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244, 1997.

[14] Stephan Bongers, Patrick Forré, Jonas Peters, and Joris M. Mooij. Foundations of structural causal models with cycles and latent variables. *Annals of Statistics*, 49(5), 2021.

[15] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-Specific Independence in Bayesian Networks. In *Proceedings of the Twelfth*

International Conference on Uncertainty in Artificial Intelligence, UAI'96, page 115–123, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[16] Marius Bozga and Oded Maler. On the representation of probabilities over structured domains. In *Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11*, pages 261–273, 1999.

[17] Philippe Brouillard, Sébastien Lachapelle, Alexandre Lacoste, Simon Lacoste-Julien, and Alexandre Drouin. Differentiable Causal Discovery from Interventional Data. In H Larochelle, M Ranzato, R Hadsell, M F Balcan, and H Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21865–21877. Curran Associates, Inc., 2020.

[18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and others. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[19] Cory Butz, Jhonatan S. Oliveira, and Robert Peharz. Sum-Product Network Decompilation. In Manfred Jaeger and Thomas Dyhre Nielsen, editors, *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 53–64. PMLR, 2020.

[20] Rafael Cabanas and Alessandro Antonucci. Crepo: An open repository to benchmark credal network algorithms. In *International Symposium on Imprecise Probability: Theories and Applications*, pages 352–356. PMLR, 2021.

[21] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[22] Federico Castelletti and Guido Consonni. Bayesian inference of causal effects from observational data in Gaussian graphical models. *Biometrics*, 77(1):136–149, 2021.

[23] Hei Chan and Adnan Darwiche. On the Robustness of Most Probable Explanations. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'06, page 63–71, Arlington, Virginia, USA, 2006. AUAI Press.

[24] Mark Chavira and Adnan Darwiche. Compiling Bayesian Networks with Local Structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, page 1306–1312, 8 2005.

[25] Mark Chavira and Adnan Darwiche. Encoding CNFs to empower component analysis. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4121 LNCS, 2006.

[26] Mark Chavira and Adnan Darwiche. Compiling Bayesian networks using variable elimination. In *IJCAI International Joint Conference on Artificial Intelligence*, 2007.

[27] Yizuo Chen and Adnan Darwiche. On the definition and computation of causal treewidth. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence*, 2022.

[28] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal*, 21(1), 2018.

[29] David Chickering. Optimal Structure Identification With Greedy Search. *Journal of Machine Learning Research*, 3:507–554, 2002.

[30] David Maxwell Chickering. Learning Bayesian Networks is NP-Complete. 1996.

[31] Arthur Choi and Adnan Darwiche. On Relaxing Determinism in Arithmetic Circuits. In *International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, page 825–833, 2017.

[32] YooJung Choi, Tal Friedman, and Guy den Broeck. Solving Marginal MAP Exactly by Probabilistic Circuit Transformations. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 3 2022.

[33] YooJung Choi, Antonio Vergari, and Guy den Broeck. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models. Technical report, 10 2020.

[34] C. K. Chow and C. N. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[35] Diego Colombo, Marloes H. Maathuis, Markus Kalisch, and Thomas S. Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *Annals of Statistics*, 40(1), 2012.

[36] Gregory F Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

[37] Juan Correa and Elias Bareinboim. General transportability of soft interventions: Completeness results. *Advances in Neural Information Processing Systems*, 33:10902–10912, 2020.

[38] Juan D. Correa and Elias Bareinboim. A calculus for stochastic interventions: Causal effect identification and surrogate experiments. In *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, volume 34, pages 10093–10100. AAAI press, 4 2020.

[39] Fabio G. Cozman. Credal networks. *Artificial Intelligence*, 120(2):199–233, 2000.

[40] Chris Cundy, Aditya Grover, and Stefano Ermon. BCD Nets: Scalable Variational Approaches for Bayesian Causal Discovery. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[41] Meihua Dang, Antonio Vergari, and Guy den Broeck. Strudel: Learning Structured-Decomposable Probabilistic Circuits. In Manfred Jaeger and Thomas Dyhre Nielsen, editors, *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 137–148. PMLR, 2020.

[42] Adnan Darwiche. A Differential Approach to Inference in Bayesian Networks. *J. ACM*, 50(3):280–305, 2003.

[43] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*, volume 9780521884. 2009.

[44] Adnan Darwiche. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *IJCAI*, 2011.

[45] Adnan Darwiche. An advance on variable elimination with applications to tensor-based computation. In *Frontiers in Artificial Intelligence and Applications*, volume 325, 2020.

[46] Adnan Darwiche. Causal inference using tractable circuits. In *NeurIPS Workshop on Causal Inference and Machine Learning: Why Now? (WHY21)*, 2021.

[47] Adnan Darwiche. Tractable Boolean and Arithmetic Circuits. 2 2022.

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

[49] Cassio Polpo De Campos and Fabio Gagliardi Cozman. The inferential complexity of bayesian and credal networks. In *IJCAI International Joint Conference on Artificial Intelligence*, 2005.

[50] Alexis de Colnet and Stefan Mengel. A Compilation of Succinctness Results for Arithmetic Circuits. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021*, pages 205–215, 10 2021.

[51] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1), 1977.

[52] Guy den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the Tractability of SHAP Explanations. *J. Artif. Int. Res.*, 74, 8 2022.

[53] Nicola Di Mauro, Gennaro Gala, Marco Iannotta, and Teresa M A Basile. Random probabilistic circuits. In Cassio de Campos and Marloes H Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1682–1691. PMLR, 2021.

[54] Pedro Zuidberg Dos Martires, Anton Dries, and Luc de Raedt. Exact and approximate weighted model integration with probability density functions using knowledge compilation. In *33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, 2019.

[55] Ralf Eggeling, Jussi Viinikka, Aleksis Vuoksenmaa, and Mikko Koivisto. On Structure Priors for Learning Bayesian Networks. In *International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1687–1695, 2019.

[56] Thomas Eiter and Thomas Lukasiewicz. Complexity results for structure-based causality. *Artificial Intelligence*, 142(1), 2002.

[57] Gal Elidan and Stephen Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9, 2008.

[58] Bradley J. Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L. Kline. Machine learning for medical imaging. *Radiographics*, 37(2):505–515, 3 2017.

[59] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1-2), 1990.

[60] Ronald A. Fisher. Cancer and smoking. *Nature*, 182(4635), 1958.

[61] Tiago M. Fragoso, Wesley Bertoli, and Francisco Louzada. Bayesian Model Averaging: A Systematic Review and Conceptual Classification. *International Statistical Review*, 86(1), 2018.

[62] Nir Friedman and Daphne Koller. Being Bayesian About Network Structure. {A} Bayesian Approach to Structure Discovery in Bayesian Networks. *Machine Learning*, 50(1-2):95–125, 2003.

[63] Isabel R Fulcher, Ilya Shpitser, Stella Marealle, and Eric J Tchetgen Tchetgen. Robust inference on population indirect causal effects: the generalized front door criterion. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(1):199–214, 2020.

[64] Atticus Geiger, Chris Potts, and Thomas Icard. Causal Abstraction for Faithful Model Interpretation. *arXiv preprint arXiv:2301.04709*, 2023.

[65] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, volume 4, 2012.

[66] Robert Gens and Domingos Pedro. Learning the Structure of Sum-Product Networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 873–880, 2013.

[67] Paolo Giudici and Robert Castelo. Improving Markov chain Monte Carlo model search for data mining. *Machine Learning*, 50(1):127–158, 2003.

[68] Stanton A Glantz, John Slade, Lisa A Bero, Peter Hanauer, and Deborah E Barnes. *The cigarette papers.* Univ of California Press, 1998.

[69] Goran Gogic, Henry Kautz, Christos Papadimitriou, and Bart Selman. The Comparative Linguistics of Knowledge Representation. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95,

page 862–869, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[70] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

[71] Marco Grzegorczyk and Dirk Husmeier. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning*, 71(2-3):265, 2008.

[72] Lewis Hammond and Vaishak Belle. Learning tractable probabilistic models for moral responsibility and blame. *Data Mining and Knowledge Discovery*, 35(2):621–659, 3 2021.

[73] Yunqiu Han, Yizuo Chen, and Adnan Darwiche. On the Complexity of Counterfactual Reasoning. *arXiv preprint arXiv:2211.13447*, 2022.

[74] David Heckerman, Christopher Meek, and Gregory Cooper. A Bayesian Approach to Causal Discovery. In *Innovations in Machine Learning: Theory and Applications*, pages 1–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[75] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4), 1999.

[76] Patrik O. Hoyer, Dominik Janzing, Joris Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, 2009.

[77] Jinbo Huang, Mark Chavira, and Adnan Darwiche. Solving MAP Exactly by Searching on Compiled Arithmetic Circuits. In *AAAI*, 2006.

[78] Jinbo Huang and Adnan Darwiche. DPLL with a trace: From SAT to knowledge compilation. In *IJCAI International Joint Conference on Artificial Intelligence*, 2005.

[79] Jinbo Huang and Adnan Darwiche. The Language of Search. *J. Artif. Int. Res.*, 29(1):191–219, 6 2007.

[80] Yimin Huang and Marco Valtorta. Pearl's Calculus of Intervention is Complete. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'06, page 217–224, Arlington, Virginia, USA, 2006. AUAI Press.

[81] Duligur Ibeling and Thomas Icard. Probabilistic reasoning across the causal hierarchy. In *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 2020.

[82] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What Are Bayesian Neural Network Posteriors Really Like? In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4629–4640. PMLR, 2021.

[83] Manfred Jaeger. Probabilistic decision graphs—combining verification and AI techniques for probabilistic inference. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 12(supp01):19–42, 2004.

[84] Manfred Jaeger, Jens D. Nielsen, and Tomi Silander. Learning probabilistic decision graphs. *International Journal of Approximate Reasoning*, 42(1-2), 2006.

[85] Yonghan Jung, Jin Tian, and Elias Bareinboim. Estimating Causal Effects Using Weighting-Based Estimators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 2020.

[86] Guy Katz, Clark W Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An Efficient {SMT} Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification - Part {I}*, pages 97–117, 2017.

[87] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On Tractable Computation of Expected Predictions, 12 2019.

[88] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[89] Doga Kisa, Guy den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *International Conference on Principles of Knowledge Representation and Reasoning*, 7 2014.

[90] Yaroslav Kivva, Ehsan Mokhtarian, Jalal Etesami, and Negar Kiyavash. Revisiting the general identifiability problem. In *Uncertainty in Artificial Intelligence*, pages 1022–1030, 2022.

[91] Mikko Koivisto. Advances in Exact Bayesian Structure Discovery in Bayesian Networks. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'06, page 241–248, Arlington, Virginia, USA, 2006. AUAI Press.

[92] Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *The Journal of Machine Learning Research*, 5:549–573, 2004.

[93] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[94] Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Journal of Machine Learning Research*, volume 31, 2013.

[95] Jack Kuipers, Giusi Moffa, and David Heckerman. Addendum on the scoring of Gaussian directed acyclic graphical models. *The Annals of Statistics*, 42(4), 8 2014.

[96] Jack Kuipers, Polina Suter, and Giusi Moffa. Efficient sampling and structure learning of Bayesian networks. *Journal of Computational and Graphical Statistics*, 31(3):639–650, 2021.

[97] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2-3), 2012.

[98] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual Fairness. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[99] Johan H.P. Kwisthout, Hans L. Bodlaender, and L. C. Van Der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *Frontiers in Artificial Intelligence and Applications*, volume 215, 2010.

[100] Emanuele La Malfa, Min Wu, Luca Laurenti, Benjie Wang, Anthony Hartshorn, and Marta Kwiatkowska. Assessing Robustness of Text Classification through Maximal Safe Radius Computation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2949–2968, Online, 11 2020. Association for Computational Linguistics.

[101] D Larkin and Rina Dechter. Bayesian inference in the presence of determinism. *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, (January 2003), 2003.

[102] Sanghack Lee, Juan D Correa, and Elias Bareinboim. General identifiability with arbitrary surrogate experiments. In *Uncertainty in artificial intelligence*, pages 389–398, 2020.

[103] Yitao Liang, Jessa Bekker, and Guy den Broeck. Learning the Structure of Probabilistic Sentential Decision Diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 8 2017.

[104] Yitao Liang and Guy den Broeck. Learning logistic circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4277–4286, 2019.

[105] Yitao Liang and Guy van den Broeck. Towards Compact Interpretable Models: Shrinking of Learned Probabilistic Sentential Decision Diagrams. In *IJCAI Workshop on Explainable AI (XAI)*, 2017.

[106] D V Lindley. *Bayesian Statistics*. Society for Industrial and Applied Mathematics, 1972.

[107] Anji Liu, Honghua Zhang, and Guy den Broeck. Scaling Up Probabilistic Circuits by Latent Variable Distillation. *arXiv preprint arXiv:2210.04398*, 2022.

[108] Qiang Liu and Alexander Ihler. Variational algorithms for marginal map. *Journal of Machine Learning Research*, 14, 2013.

[109] Qiang Liu and Dilin Wang. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[110] Lars Lorch, Jonas Rothfuss, Bernhard Schölkopf, and Andreas Krause. DiBS: Differentiable Bayesian Structure Learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[111] Daniel Lowd and Pedro Domingos. Learning Arithmetic Circuits. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, page 383–392, Arlington, Virginia, USA, 2008. AUAI Press.

[112] Marloes H. Maathuis, Markus Kalisch, and Peter Bühlmann. Estimating high-dimensional intervention effects from observational data. *Annals of Statistics*, 37(6 A), 2009.

[113] David Madigan, Jeremy York, and Denis Allard. Bayesian graphical models for discrete data. *International Statistical Review/Revue Internationale de Statistique*, 63(2):215–232, 1995.

[114] Radu Marinescu, Akihiro Kishimoto, and Adi Botea. Parallel AND/OR search for marginal MAP. In *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 2020.

[115] Robert Mateescu, Rina Dechter, and Radu Marinescu. AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. *Journal of Artificial Intelligence Research*, 33, 2008.

[116] Denis Deratani Mauá, Fabio Gagliardi Cozman, Diarmaid Conaty, and Cassio Polpo De Campos. Credal sum-product networks. In *Proceedings of the 10th International Symposium on Imprecise Probability: Theories and Applications, ISIPTA 2017*, 2019.

[117] Marina Meilă and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1(1), 2001.

[118] Ramsés H. Mena and Stephen G. Walker. On the Bayesian Mixture Model and Identifiability. *Journal of Computational and Graphical Statistics*, 24(4), 2015.

[119] Thomas Mitchell. *Machine Learning (McGraw-Hill International Editions Computer Science Series): Tom M. Mitchell: 9780071154673*. McGraw-Hill Pub. Co. (ISE Editions), 1997.

[120] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *The Journal of Machine Learning Research*, 21(1):5183–5244, 2020.

[121] Milan Mossé, Duligur Ibeling, and Thomas Icard. Is Causal Reasoning Harder than Probabilistic Reasoning? *Review of Symbolic Logic*, 2022.

[122] Sajjad Mozaffari, Omar Y. Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep Learning-Based Vehicle Behavior Prediction for Autonomous Driving Applications: A Review, 1 2022.

[123] Kevin P. Murphy. Active learning of causal bayes net structure, 2001.

[124] Michael Oberst and David Sontag. Counterfactual off-policy evaluation with gumbel-max structural causal models. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, 2019.

[125] Umut Oztok, Arthur Choi, and Adnan Darwiche. Solving PPPP-Complete Problems Using Knowledge Compilation. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'16, page 94–103. AAAI Press, 2016.

[126] Ioannis Papantonis and Vaishak Belle. Interventions and Counterfactuals in Tractable Probabilistic Models: Limitations of Contemporary Transformations. *arXiv preprint*, abs/2001.1, 2020.

[127] James Park and Adnan Darwiche. Solving MAP Exactly using Systematic Search. In *Proceedings of the Nineteenth Conference Conference on Uncertainty in Artificial Intelligence (UAI 2003)*, 2003.

[128] James D. Park and Adnan Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, 2 2004.

[129] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[130] Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4), 1995.

[131] Judea Pearl. Probabilities Of Causation: Three Counterfactual Interpretations And Their Identification. *Synthese*, 121(1/2):93–149, 1999.

[132] Judea Pearl. *Causality*. Cambridge University Press, Cambridge, UK, second edition, 2009.

[133] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009.

[134] Robert Peharz. *Foundations of sum-product networks for probabilistic modeling*. PhD thesis, Medical University of Graz, 2015.

[135] Robert Peharz, Bernhard C Geiger, and Franz Pernkopf. Greedy Part-Wise Learning of Sum-Product Networks. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 612–627, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[136] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro M Domingos. On the Latent Variable Interpretation in Sum-Product Networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(10):2030–2044, 2017.

[137] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 7 2020.

[138] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On Theoretical Properties of Sum-Product Networks. In Guy Lebanon and S V N Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 744–752, San Diego, California, USA, 6 2015. PMLR.

[139] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning. In Ryan P Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344. PMLR, 2020.

[140] Johan Pensar, Topi Talvitie, Antti Hyttinen, and Mikko Koivisto. A Bayesian Approach for Estimating Causal Effects from Observational Data. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5395–5402, 2020.

[141] Jonas Peters, Dominik Janzing, and Bernhard Schlkopf. *Elements of Causal Inference: Foundations and Learning Algorithms.* The MIT Press, 2017.

[142] Knot Pipatsrisawat and Adnan Darwiche. New Compilation Languages Based on Structured Decomposability. In *AAAI*, pages 517–522. {AAAI} Press, 2008.

[143] Knot Pipatsrisawat and Adnan Darwiche. A new d-DNNF-based bound computation algorithm for functional E-MAJSAT. In *Twenty-First International Joint Conference on Artificial Intelligence*. Citeseer, 2009.

[144] Thammanit Pipatsrisawat and Adnan Darwiche. A Lower Bound on the Size of Decomposable Negation Normal Form. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

[145] Drago Plecko and Elias Bareinboim. Causal fairness analysis. *arXiv preprint arXiv:2207.11385*, 2022.

[146] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in Artificial Intelligence*, 2011.

[147] Biao Qin. Differential semantics of intervention in Bayesian networks. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 710–716, 7 2015.

[148] Tahrima Rahman, Shasha Jin, and Vibhav Gogate. Cutset Bayesian networks: A new representation for learning rao-blackwellised graphical models. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2019-Augus, 2019.

[149] Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 630–645, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[150] Alexander Reisach, Christof Seiler, and Sebastian Weichwald. Beware of the Simulated DAG! Causal Discovery Benchmarks May Be Easy to Game. In M Ranzato, A Beygelzimer, Y Dauphin, P S Liang, and J Wortman Vaughan,

editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27772–27784. Curran Associates, Inc., 2021.

[151] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[152] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *32nd International Conference on Machine Learning, ICML 2015*, volume 2, 2015.

[153] Thomas S Richardson, Robin J Evans, James M Robins, and Ilya Shpitser. Nested Markov properties for acyclic directed mixed graphs. *The Annals of Statistics*, 51(1):334–361, 2023.

[154] Amirmohammad Rooshenas and Daniel Lowd. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In *International Conference on International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, 2014.

[155] Paul K. Rubenstein, Sebastian Weichwald, Stephan Bongers, Joris M. Mooij, Dominik Janzing, Moritz Grosse-Wentrup, and Bernhard Schölkopf. Causal consistency of structural equation models. In *Uncertainty in Artificial Intelligence - Proceedings of the 33rd Conference, UAI 2017*, 2017.

[156] Raquel Sanchez-Cauce, Iago Paris, and Francisco Javier Diez. Sum-Product Networks: A Survey, 7 2022.

[157] Mauro Scanagatta, Giorgio Corani, Cassio P. De Campos, and Marco Zaffalon. Learning Treewidth-Bounded Bayesian networks with thousands of variables. In *Advances in Neural Information Processing Systems*, 2016.

[158] Bernhard Schölkopf. Causality for Machine Learning. In *Probabilistic and Causal Inference*, pages 765–804. 11 2022.

[159] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

[160] Marco Scutari. Bayesian Network Repository. \url{https://www.bnlearn.com/bnrepository/}, 2022.

[161] Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140, 2022.

[162] Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable Operations for Arithmetic Circuits of Probabilistic Models. In D Lee, M Sugiyama, U Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, 2016.

[163] Yujia Shen, Arthur Choi, and Adnan Darwiche. Conditional PSDDs: Modeling and Learning With Modular Knowledge, 2018.

[164] Yujia Shen, Anchal Goyanka, Adnan Darwiche, and Arthur Choi. Structured bayesian networks: From inference to learning with routes. In *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, 2019.

[165] Andy Shih, Guy den Broeck, Paul Beame, and Antoine Amarilli. Smoothing Structured Decomposable Circuits. In H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019.

[166] Andy Shih and Stefano Ermon. Probabilistic Circuits for Variational Inference in Discrete Graphical Models. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[167] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvärinen, and Antti Kerminen. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7, 2006.

[168] Ilya Shpitser and Judea Pearl. Identification of Joint Interventional Distributions in Recursive Semi-Markovian Causal Models. AAAI'06, page 1219–1226. AAAI Press, 2006.

[169] Ilya Shpitser and Judea Pearl. Complete identification methods for the causal hierarchy. *Journal of Machine Learning Research*, 9:1941–1979, 2008.

[170] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[171] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *9th International Conference on Learning Representations, {ICLR} 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net, 2021.

[172] Peter Spirtes. Introduction to causal inference. *Journal of Machine Learning Research*, 11, 2010.

[173] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search.* MIT press, 2000.

[174] Peter Spirtes and Kun Zhang. Causal discovery and inference: concepts and recent methodological advances. In *Applied informatics*, volume 3, pages 1–28, 2016.

[175] M Stephens. Dealing with multimodal posteriors and non-identifiablity in mixture models. *Journal of the Royal Statistical Society, Series B*, xx(xx), 2000.

[176] Adarsh Subbaswamy, Peter Schulam, and Suchi Saria. Preventing Failures Due to Dataset Shift: Learning Predictive Models That Transport. In *Proceedings*

of the 22nd International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, pages 3118–3127, 2019.

[177] Jin Tian and Judea Pearl. On the testable implications of causal models with hidden variables. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, 2002.

[178] Panagiotis Tigas, Yashas Annadani, Andrew Jesson, Bernhard Schölkopf, Yarin Gal, and Stefan Bauer. Interventions, Where and How? Experimental Design for Causal Models at Scale. In S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, and A Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24130–24143. Curran Associates, Inc., 2022.

[179] Santtu Tikka, Antti Hyttinen, and Juha Karvanen. Identifying causal effects via context-specific independence relations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[180] Christian Toth, Lars Lorch, Christian Knoll, Andreas Krause, Franz Pernkopf, Robert Peharz, and Julius von Kügelgen. Active Bayesian Causal Inference. In S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, and A Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 16261–16275. Curran Associates, Inc., 2022.

[181] Konstantinos Tsirlis, Vincenzo Lagani, Sofia Triantafillou, and Ioannis Tsamardinos. On scoring Maximal Ancestral Graphs with the Max–Min Hill Climbing algorithm. *International Journal of Approximate Reasoning*, 102:74–85, 2018.

[182] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[183] Benito van der Zander, Markus Bläser, and Maciej Liśkiewicz. The Hardness of Reasoning about Probabilities and Causality. *arXiv preprint arXiv:2305.09508*, 2023.

[184] Artem Velikzhanin, Benjie Wang, and Marta Kwiatkowska. Bayesian Network Models of Causal Interventions in Healthcare Decision Making: Literature Review and Software Evaluation. *arXiv preprint arXiv:2211.15258*, 2022.

[185] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy den Broeck. A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In M Ranzato, A Beygelzimer, Y Dauphin, P S Liang, and J Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 13189–13201, 2021.

[186] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and Understanding Sum-Product Networks. *Machine Learning*, 108(4):551–573, 4 2019.

[187] Thomas Verma and Judea Pearl. Equivalence and Synthesis of Causal Models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, pages 255–270, USA, 1990. Elsevier Science Inc.

[188] Jussi Viinikka, Antti Hyttinen, Johan Pensar, and Mikko Koivisto. Towards scalable Bayesian learning of causal DAGs. In *Advances in Neural Information Processing Systems*, volume 2020-Decem, pages 6584–6594, 2020.

[189] Benjie Wang and Marta Kwiatkowska. Symbolic Causal Inference via Operations on Probabilistic Circuits. In *NeurIPS 2022 Workshop on Neuro Causal and Symbolic AI (nCSI)*, 2022.

[190] Benjie Wang and Marta Kwiatkowska. Compositional Probabilistic and Causal Inference using Tractable Circuit Models. In *International Conference on Artificial Intelligence and Statistics*, pages 9488–9498. PMLR, 2023.

[191] Benjie Wang, Clare Lyle, and Marta Kwiatkowska. Provable Guarantees on the Robustness of Decision Rules to Causal Interventions. In *IJCAI International Joint Conference on Artificial Intelligence*, 2021.

[192] Benjie Wang, Stefan Webb, and Tom Rainforth. Statistically robust neural network classification. In *Uncertainty in Artificial Intelligence*, pages 1735–1745. PMLR, 2021.

[193] Benjie Wang, Matthew R Wicker, and Marta Kwiatkowska. Tractable Uncertainty for Structure Learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23131–23150. PMLR, 2022.

[194] Sergio Wechsler, Rafael Izbicki, and Luís Gustavo Esteves. A Bayesian Look at Nonidentifiability: A Simple Example. *American Statistician*, 67(2), 2013.

[195] Matthew Wicker, Luca Laurenti, Andrea Patane, Zhuotong Chen, Zheng Zhang, and Marta Kwiatkowska. Bayesian Inference with Certifiable Adversarial Robustness. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2431–2439. PMLR, 2021.

[196] Hjalmar Wijk, Benjie Wang, and Marta Kwiatkowska. Robustness Guarantees for Credal Bayesian Networks via Constraint Relaxation over Probabilistic Circuits. In *International Joint Conference on Artificial Intelligence*, 2022.

[197] Sewall Wright. The Method of Path Coefficients. *The Annals of Mathematical Statistics*, 5(3):161–215, 1934.

[198] Kevin Xia, Kai Zhan Lee, Yoshua Bengio, and Elias Bareinboim. The Causal-Neural Connection: Expressiveness, Learnability, and Inference. In *Advances in Neural Information Processing Systems*, volume 13, 2021.

[199] Yue Yu, Jie Chen, Tian Gao, and Mo Yu. {DAG}-{GNN}: {DAG} Structure Learning with Graph Neural Networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on*

Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, pages 7154–7163. PMLR, 2019.

[200] Marco Zaffalon, Alessandro Antonucci, and Rafael Cabañas. Structural Causal Models Are (Solvable by) Credal Networks. 8 2020.

[201] Matej Zecevic, Devendra Singh Dhami, Athresh Karanam, Sriraam Natarajan, and Kristian Kersting. Interventional Sum-Product Networks: Causal Inference with Tractable Probabilistic Models. In A Beygelzimer, Y Dauphin, P Liang, and J Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[202] Matej Zecevic, Devendra Singh Dhami, and Kristian Kersting. A Taxonomy for Inference in Causal Model Families. *CoRR*, abs/2110.1, 2021.

[203] Honghua Zhang, Steven Holtzen, and Guy Broeck. On the relationship between probabilistic circuits and determinantal point processes. In *Conference on Uncertainty in Artificial Intelligence*, pages 1188–1197, 2020.

[204] Honghua Zhang, Brendan Juba, and Guy den Broeck. Probabilistic generating circuits. In *International Conference on Machine Learning*, pages 12447–12457, 2021.

[205] Junzhe Zhang, Jin Tian, and Elias Bareinboim. Partial counterfactual identification from observational and experimental data. In *International Conference on Machine Learning*, pages 26548–26558. PMLR, 2022.

[206] Kun Zhang and Lai-Wan Chan. Extensions of ICA for Causality Discovery in the Hong Kong Stock Market. In *Proceedings of the 13th International Conference on Neural Information Processing - Volume Part III*, ICONIP'06, pages 400–409, Berlin, Heidelberg, 2006. Springer-Verlag.

[207] Kun Zhang and Aapo Hyvärinen. On the Identifiability of the Post-Nonlinear Causal Model. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 647–655, Arlington, Virginia, USA, 2009. AUAI Press.

[208] Xiyue Zhang, Benjie Wang, and Marta Kwiatkowska. On Preimage Approximation for Neural Networks. *arXiv preprint arXiv:2305.03686*, 2023.

[209] Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed Variational Inference for Sum-Product Networks. In Maria Florina Balcan and Kilian Q Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1310–1318, New York, New York, USA, 2016. PMLR.

[210] Han Zhao, Mazen Melibari, and Pascal Poupart. On the Relationship between Sum-Product Networks and Bayesian Networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 116–124, Lille, France, 2015. PMLR.

[211] Han Zhao, Pascal Poupart, and Geoff Gordon. A unified approach for learning the parameters of Sum-Product Networks. In *Advances in Neural Information Processing Systems*, 2016.

[212] Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. DAGs with NO TEARS: Continuous Optimization for Structure Learning. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[213] Xun Zheng, Chen Dan, Bryon Aragam, Pradeep Ravikumar, and Eric Xing. Learning Sparse Nonparametric DAGs. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3414–3425. PMLR, 2020.

Appendix A

Advanced Causal Reasoning via Compilation

A.1 Proofs

Lemma 4.1 (Known Parents)**.** *Let $\mathcal{C}$ be a compiled PC that symbolically computes the CBN $\mathcal{N}$, and satisfies topological ordering constraints $<_{top}$. Then, for any sum node T deciding on a variable V_i, every complete subcircuit $\mathcal{C}_{sub}$ of $\mathcal{C}$ which contains T must contain the same indicators for the parents $\boldsymbol{PA}_i$.*

Proof. Suppose for contradiction that this was not the case; then there are two subcircuits $\mathcal{C}_{sub,1}, \mathcal{C}_{sub,2}$ containing conflicting indicators for the parents $\boldsymbol{PA}_i$ of V_i. We will split each subcircuit into a *prefix* and a *suffix*. Let $\mathcal{C}_T$ denote the circuit rooted at T. The suffix $\mathcal{C}_{sub,S}$ of a subcircuit $\mathcal{C}_{sub}$ is defined to be all nodes, edges, weights, and leaf functions which are contained within $\mathcal{C}_T$ and $\mathcal{C}_{sub}$, while the prefix $\mathcal{C}_{sub,P}$ is all nodes, edges, weights and leaf functions contained in $\mathcal{C}_{sub}$ but not contained within $\mathcal{C}_T$, i.e. external to $\mathcal{C}_T$.

We thus have $\mathcal{C}_{sub,1} = (\mathcal{C}_{sub,1,P}, \mathcal{C}_{sub,1,S})$ and $\mathcal{C}_{sub,2} = (\mathcal{C}_{sub,2,P}, \mathcal{C}_{sub,2,S})$. By the topological ordering constraints, $\mathcal{C}_T$ cannot contain any sum nodes deciding on any parent variable in $\boldsymbol{PA}_i$, and thus there cannot be any indicator corresponding to any parent variable in $\boldsymbol{PA}_i$ in $\mathcal{C}_T$ (and thus in any suffix subcircuit). Now, we define $\mathcal{C}'_{sub,2} := (\mathcal{C}_{sub,2,P}, \mathcal{C}_{sub,1,S})$. Since no indicators for parent variables are contained in $\mathcal{C}_{sub,1,S}$ or $\mathcal{C}_{sub,2,S}$, $\mathcal{C}'_{sub,2}$ contains indicators corresponding to the same values of the parent variables as $\mathcal{C}_{sub,2}$. In particular, this is still different from $\mathcal{C}_{sub,1}$.

As the circuit symbolically computes a Bayesian network, each subcircuit corresponds to some instantiation of the variables. Now, comparing $\mathcal{C}_{sub,1}, \mathcal{C}'_{sub,2}$, we see that they have the same suffix subcircuit, meaning that they specify the same value of V_i (say v_i), but different values $\boldsymbol{pa}_i, \boldsymbol{pa}'_i$ of $\boldsymbol{PA}_i$. This means that the subcircuits contain parameters $\theta_{v_i|\boldsymbol{pa}_i}, \theta_{v_i|\boldsymbol{pa}'_i}$ respectively. Since these parameters depend on the value of V_i, they must appear in $\mathcal{C}_T$. But this is a contradiction as both suffix subcircuits are identical.

□

Proposition 4.1. *Suppose T decides on variable V_i. Then, for any parent $PA_i \in \boldsymbol{PA}_i$ of V_i, $\boldsymbol{P}(T, PA_i)$ is a singleton set.*

Proof. Recall the definition:

$$\boldsymbol{P}(N, V_i) := \{v_i : \exists L \in \mathrm{desc}(N) : g_N = \theta_{v_j|\boldsymbol{pa}_j} \text{ and } v_i \in \boldsymbol{pa}_j\} \tag{A.1}$$

In particular, $\boldsymbol{P}(T, PA_i)$ contains the set of all values of PA_i such that there is a descendant leaf node of T corresponding to a parameter that specifies that value of PA_i. Such a parameter exists: namely $\theta_{V_i|\boldsymbol{pa}_i}$, which is contained within $\mathcal{C}_T$. Thus $\boldsymbol{P}(T, PA_i)$ is non-empty. Further, by Lemma 4.1, it cannot contain more than one element.

□

Proposition 4.2 (Correctness of Transpiled PC)**.** *The output of Algorithm 4.1 symbolically computes $\mathcal{CBN}$.*

Proof. Given a compiled PC $\mathcal{C} = (G, \boldsymbol{\omega} = 1, \boldsymbol{g}(\boldsymbol{\lambda}, \boldsymbol{\Theta}))$ symbolically computing $\mathcal{CBN}$, the algorithm outputs a transpiled circuit $\mathcal{C}' = (G, \boldsymbol{\omega}(\boldsymbol{\Theta}), \boldsymbol{g}(\boldsymbol{\lambda}))$. We will show that $f_{\mathcal{C}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) = f_{\mathcal{C}}(\boldsymbol{\lambda}, \boldsymbol{\Theta})$, which implies that $\mathcal{C}'$ symbolically computes $\mathcal{CBN}$.

The only changes the algorithm makes are to remove leaves corresponding to parameter variables, and encode those parameters in the sum node weights. In particular, the leaf nodes corresponding to indicator nodes are not changed. Thus we can put the complete subcircuits of $\mathcal{C}, \mathcal{C}'$ in one-to-one correspondence such that $(\mathcal{C}_{sub}, \mathcal{C}'_{sub})$ both correspond to the same value $\boldsymbol{v}$ of the domain variables $\boldsymbol{V}$. For each variable $V_i \in \boldsymbol{V}$, the subcircuit $\mathcal{C}_{sub}$ contains exactly one sum node T_i deciding on V_i, and one parameter of the form $\theta_{v_i|\boldsymbol{pa}_i}$. The algorithm moves this parameter to be the weight (in the subcircuit) of T_i, so that the overall term is unchanged in $\mathcal{C}'_{sub}$. Applying this to all variables, we have that the terms of every corresponding pair of subcircuits $(\mathcal{C}_{sub}, \mathcal{C}'_{sub})$ is the same and so $f_{\mathcal{C}}(\boldsymbol{\lambda}, \boldsymbol{\Theta}) = f_{\mathcal{C}}(\boldsymbol{\lambda}, \boldsymbol{\Theta})$. □

Proposition 4.3 (Correctness of Algorithm 4.3)**.** *The output $\mathcal{C}'$ of Algorithm 4.3 has polynomial $f_{\mathcal{M}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}}$.*

Proof. Firstly, we note that the computation of p_{cond} in the algorithm corresponds to a marginal inference query for the event $\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}$ on the original circuit $\mathcal{C}$, such that $p_{cond}(R) = p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})$, where R is the root sum unit.

Consider any subcircuit $\mathcal{C}_{sub}$ of $\mathcal{C}$, corresponding to some particular values $\boldsymbol{u}, \boldsymbol{v}$ of the endogenous and exogenous variables. Further, let $\mathcal{C}'_{sub}$ be the corresponding subcircuit in $\mathcal{C}'$ (also corresponding to $\boldsymbol{u}, \boldsymbol{v}$. The terms of these subcircuits differ only in the parts changed by the algorithm: that is, the indicators $\mathbb{1}_{v_i^{(1)} = F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}$, and the reweighting of the weight associated with the root sum node. Since each subcircuit has exactly one sum node deciding on each endogenous variable $V_i \in \boldsymbol{V}$, we have that the term of $\mathcal{C}'_{sub}$ is equal to the term of $\mathcal{C}_{sub}$ weighted by a factor $\frac{\prod_{i=1}^{d} \mathbb{1}_{v_i^{(1)} = F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})}$.

Applying this result for all subcircuits, we have that:

$$
\begin{aligned}
f_{\mathcal{C}'} &= \sum_{\mathcal{C}'_{sub} \in \mathcal{C}'} M_{\mathcal{C}'_{sub}} && \text{(A.2)} \\
&= \sum_{\mathcal{C}_{sub} \in \mathcal{C}} \frac{\prod_{i=1}^{d} \mathbb{1}_{v_i^{(1)} = F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} M_{\mathcal{C}_{sub}} && \text{(A.3)} \\
&= \sum_{\boldsymbol{u} \in \mathrm{val}(\boldsymbol{U}), \boldsymbol{v} \in \mathrm{val}(\boldsymbol{V})} \frac{\prod_{i=1}^{d} \mathbb{1}_{v_i^{(1)} = F_i^{(1)}(\boldsymbol{u}_i, \boldsymbol{pa}_i^{(1)})}}{p_{\mathcal{M}}(\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}})} \prod_{U \in \boldsymbol{U}} \mathbb{1}_{U=u} \mathrm{Pr}_U(u) \prod_{i=1}^{d} \mathbb{1}_{V_i = v_i} \theta_{v_i|\boldsymbol{u}_i, \boldsymbol{pa}_i} && \text{(A.4)} \\
&= f_{\mathcal{M}|\boldsymbol{v}^{(1)}_{\boldsymbol{F}^{(1)}}} && \text{(A.5)}
\end{aligned}
$$

□

Appendix B

Tractability of Causal Inference

B.1 Proofs

Proposition 5.1 (Tractable Symbolic Conditioning). *Let $\mathcal{C}$ be a decomposable, smooth and $\boldsymbol{Q}$-deterministic circuit over variables $\boldsymbol{V}$. Then computing $\texttt{COND}(\mathcal{C};\boldsymbol{Q})$ as a decomposable, smooth and $\boldsymbol{Q}$-deterministic circuit is tractable in $O(|\mathcal{C}|)$ time and space.*

Proof. Firstly, it is clear that Algorithm 5.1 runs in $O(|\mathcal{C}|)$ time, using a single forward pass through the circuit starting at the leaves (and keeping track of the value of $c[N]$, which computes the normalizing constant $p_N(\emptyset)$ at each node N). Further, the output circuit $\mathcal{C}'$ is decomposable and smooth as the operations do not change the scope of any of the nodes, and is further $\boldsymbol{Q}$-deterministic as the support of the leaf nodes do not change.

It remains to show that $\mathcal{C}'$ faithfully represents the conditional distribution according to Definition 5.3, i.e.:

$$p_{\mathcal{C}'}(\boldsymbol{V}) = \texttt{COND}(p_{\mathcal{C}'};\boldsymbol{Q}) \begin{cases} \frac{p_{\mathcal{C}}(\boldsymbol{V})}{p_{\mathcal{C}}(\boldsymbol{Q})} & \text{if } p_{\mathcal{C}}(\boldsymbol{Q}) > 0 \\ 0 & \text{if } p_{\mathcal{C}}(\boldsymbol{Q}) = 0 \end{cases} \tag{B.1}$$

We prove this by induction over for all nodes N (children before parents); namely, by showing that:

$$p'_N(\boldsymbol{V})^1 = \texttt{COND}(p'_N;\boldsymbol{Q}) = \begin{cases} \frac{p_N(\boldsymbol{V})}{p_N(\boldsymbol{Q})} & \text{if } p_N(\boldsymbol{Q}) > 0 \\ 0 & \text{if } p_N(\boldsymbol{Q}) = 0 \end{cases} \tag{B.2}$$

[1]Recall that for node distributions, we write e.g. $p'_N(\boldsymbol{V})$ to mean $p'_N(\boldsymbol{V} \cap \phi(N))$.

where p' represents the distribution in $\mathcal{C}'$, and p the distribution in $\mathcal{C}$. The above result then follows from this result for the root node R.

Leaf Nodes For the base case, we consider leaf nodes L. Here, by Line 5 the result follows immediately.

Sum Nodes Now, we consider sum nodes T. Firstly, let us consider the case that $p_T(\boldsymbol{Q}) = 0$; then, we must have that $p_{N_i}(\boldsymbol{Q}) = 0$ whenever $\omega_{T,i} > 0$, and so by the inductive hypothesis, $p'_{N_i}(\boldsymbol{V}) = 0$. Further, note that the changes to the weights in Lines 11, 13 never makes a weight $\omega'_{T,i}$ positive if the original $\omega_{T,i} = 0$. This implies that $p'_{T,i}(\boldsymbol{V}) = 0$, as required.

Now, suppose instead that $p_T(\boldsymbol{Q}) > 0$. If $\phi(T) \cap \boldsymbol{Q} \neq \emptyset$, then by Line 11 we set $\omega'_{T,i} = \mathbb{1}_{\omega_{T,i}>0}$ for all i. Further, by $\boldsymbol{Q}$-determinism of $\mathcal{C}$, only one child N_j has $p_{N_j}(\boldsymbol{Q}) > 0$. This child must have $\omega_{T,j} > 0$ in order that $p_T(\boldsymbol{Q}) > 0$, and so $\omega'_{T,j} = 1$. Thus:

$$\frac{p_T(\boldsymbol{V})}{p_T(\boldsymbol{Q})} = \frac{\omega_{T,j} p_{N_j}(\boldsymbol{V})}{\omega_{T,j} p_{N_j}(\boldsymbol{Q})} = p'_{N_j}(\boldsymbol{V}) = p'_T(\boldsymbol{V}) \tag{B.3}$$

Here, the second equality follows by the inductive hypothesis, while the third equality follows from the fact that $\omega'_{T,j} = 1$.

If instead $\phi(T) \cap \boldsymbol{Q} = \emptyset$, then by Line 13 we set $\omega'_{T,i} = \frac{\omega_{T,i} p_{N_i}(\emptyset)}{\sum_{N_j \in \mathrm{ch}(T)} \omega_{T,j} p_{N_j}(\emptyset)}$ for all i. We thus have:

$$\frac{p_T(\boldsymbol{V})}{p_T(\boldsymbol{Q})} = \frac{\sum_i \omega_{T,i} p_{N_i}(\boldsymbol{V})}{\sum_j \omega_{T,j} p_{N_j}(\boldsymbol{Q})} = \frac{\sum_i \omega_{T,i} p_{N_i}(\boldsymbol{V})}{\sum_j \omega_{T,j} p_{N_j}(\emptyset)} = \sum_i \omega'_{T,i} \frac{p_{N_i}(\boldsymbol{V})}{p_{N_i}(\emptyset)} \tag{B.4}$$

$$= \sum_i \omega'_{T,i} \frac{p_{N_i}(\boldsymbol{V})}{p_{N_i}(\boldsymbol{Q})} = \sum_i \omega'_{T,i} p'_{N_i}(\boldsymbol{V}) = p'_{T_i}(\boldsymbol{V}) \tag{B.5}$$

The second equality follows because $\phi(T) = \phi(T_j)$ (smoothness), and so $\phi(T_j) \cap \boldsymbol{Q} = \emptyset$. The third equality follows by the definition of the new weight, the fourth again by $\phi(T_j) \cap \boldsymbol{Q} = \emptyset$, and the fifth by the inductive hypothesis.

Thus we have shown in all cases that the inductive hypothesis holds for T.

Product Node Finally, we consider product nodes P. If $p_P(\boldsymbol{Q}) = 0$, then one of its children (say N_1 has $p_{N_i}(\boldsymbol{Q}) = 0$. By the inductive hypothesis, $p'_{N_i}(\boldsymbol{V}) = 0$ and thus $p'_P(\boldsymbol{V}) = 0$.

If $p_P(\boldsymbol{Q}) > 0$, then we have:

$$\frac{p_P(\boldsymbol{V})}{p_P(\boldsymbol{Q})} = \frac{\prod_{i=1,2} p_{N_i}(\boldsymbol{V})}{\prod_{i=1,2} p_{N_i}(\boldsymbol{Q})} = \prod_{i=1,2} \frac{p_{N_i}(\boldsymbol{V})}{p_{N_i}(\boldsymbol{Q})} = \prod_{i=1,2} p'_{N_i}(\boldsymbol{V}) = p'_P(\boldsymbol{V}) \tag{B.6}$$

Here, the first inequality follows because of decomposability, the third inequality by the inductive hypothesis, and the final equality by Line 7.

$\square$

Theorem 5.1 (Hardness of Backdoor Query). *The backdoor query for decomposable and smooth PCs is #P-hard, even if the PC is structured decomposable and deterministic.*

Proof. We prove this in the case of binary variables for brevity of presentation, though the proof can be easily extended to non-binary discrete variables. Our proof is based on a reduction from the problem of computing the expectation of a logistic regression model, which was defined and shown to be #P-hard in [52] and which we refer to as the `EXPLR` problem. In particular, for any `EXPLR` problem over variables $\boldsymbol{Z}$, with input size $n_{\boldsymbol{Z}} = |\boldsymbol{Z}|$, we construct a circuit in time and with size linear in $\boldsymbol{Z}$ and where computing the backdoor query $p_{\boldsymbol{x}}(\boldsymbol{y}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}) p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{Z})$ is equivalent to solving the `EXPLR` problem.

The `EXPLR` problem is defined as computing the following quantity (where $w_i \in \mathbb{R}$):

$$\texttt{EXPLR}(\boldsymbol{w}) = \sum_{\boldsymbol{Z}} \frac{1}{1 + e^{-(w_0 + \sum_i w_i Z_i)}} \tag{B.7}$$

We will construct a circuit over variables $\boldsymbol{V} = \{\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}\}$, where the sets $\boldsymbol{X} = \{X\}$ and $\boldsymbol{Y} = \{Y\}$ each consist of a single variable, and consider the backdoor query for instantiations x, y of X, Y. We begin by defining a number of auxiliary circuits/nodes for $\boldsymbol{X}, \boldsymbol{Y}$ and $\boldsymbol{Z}$ individually, all structured decomposable, smooth and deterministic, which will be part of the construction of the main circuit.

First, for $\boldsymbol{Y}$ we define the leaf nodes $\mathbb{1}_y, \mathbb{1}_{\neg y}$ to encode the functions $p_{\mathbb{1}_y}(Y) := \mathbb{1}_{Y=y}, p_{\mathbb{1}_{\neg y}}(Y) := \mathbb{1}_{Y=\neg y}$ respectively. For $\boldsymbol{X}$, we define $\mathbb{1}_x, \mathbb{1}_{\neg x}$ to encode $p_{\mathbb{1}_x}(X) :=$

$\mathbb{1}_{X=x}, p_{\mathbb{1}_{\neg x}}(X) := \mathbb{1}_{X=\neg x}$ (respectively) in a similar manner. Finally, for $\boldsymbol{Z}$, we define two circuits, $\mathbb{1}_{\boldsymbol{Z}}$ and $\mathcal{C}_{\boldsymbol{Z}}$, as follows. Let $\boldsymbol{Z} := \{Z_1, ..., Z_{n_{\boldsymbol{Z}}}\}$ be an arbitrary ordering of the variables in $\boldsymbol{Z}$, and let $\boldsymbol{Z}_{\geq i}$ denote $\{X_i, ..., X_{n_{\boldsymbol{Z}}}\}$ for any $1 \leq i \leq n_{\boldsymbol{Z}}$. Then we define the circuit $\mathbb{1}_{\boldsymbol{Z}}$ recursively as follows, where $\mathbb{1}_{\boldsymbol{Z}} := \mathbb{1}_{\boldsymbol{Z}_{\geq 1}}$ (where $\times$ indicates a product node with its arguments as children):

$$\mathbb{1}_{\boldsymbol{Z}_{\geq i}} := \begin{cases} \mathbb{1}_{Z_i} \times \mathbb{1}_{\boldsymbol{Z}_{\geq i+1}} & 1 \leq i < n_{\boldsymbol{Z}} \\ \mathbb{1}_{Z_i} & i = n_{\boldsymbol{Z}} \end{cases} \tag{B.8}$$

This circuit consists of a series of product units, and leaf units $\mathbb{1}_{Z_i}$ for each $Z_i \in Z$ which we define to encode the function $p_{\mathbb{1}_{Z_i}}(Z_i) \equiv 1$ (for all values of Z_i). Thus, the circuit as a whole encodes $p_{\mathbb{1}_{\boldsymbol{Z}}}(\boldsymbol{Z}) \equiv 1$ for all values of $\boldsymbol{Z}$. In terms of structural and support properties, the circuit is trivially deterministic and smooth as it does not contain any sum nodes, and is clearly also structured decomposable. Finally, it can also be seen that the size $|\mathbb{1}_{\boldsymbol{Z}}|$ (number of edges) of $\mathbb{1}_{\boldsymbol{Z}}$ is $O(n_{\boldsymbol{Z}})$.

We now design a circuit $\mathcal{C}_{\boldsymbol{Z}}$ to encode the function $e^{-(w_0 + \sum_i w_i Z_i)}$ as follows, where $\mathcal{C}_{\boldsymbol{Z}} := \mathcal{C}_{\boldsymbol{Z}_{\geq 1}}$:

$$\mathcal{C}_{\boldsymbol{Z}_{\geq i}} := \begin{cases} \mathcal{C}_{Z_i} \times \mathcal{C}_{\boldsymbol{Z}_{\geq i+1}} & 1 \leq i < n_{\boldsymbol{Z}} \\ \mathcal{C}_{Z_i} & i = n_{\boldsymbol{Z}} \end{cases} \tag{B.9}$$

where we define leaf nodes $\mathcal{C}_{Z_i}$ to encode $p_{\mathcal{C}_{Z_i}}(Z_i) := e^{-w_i Z_i}$ for $1 \leq i < n_{\boldsymbol{Z}}$ and $p_{\mathcal{C}_{Z_i}}(Z_i) := e^{-(w_0 + w_i Z_i)}$ for $i = n_{\boldsymbol{Z}}$. By recursion it can be seen that this circuit does indeed encode $p_{\mathcal{C}_{\boldsymbol{Z}}}(\boldsymbol{Z}) = e^{-(w_0 + \sum_i w_i Z_i)}$. This circuit is deterministic and smooth, and also decomposes in the same way as $\mathbb{1}_{\boldsymbol{Z}}$, i.e. they are structured decomposable with the same vtree. It can also be seen that the size $|\mathcal{C}_{\boldsymbol{Z}}|$ of $\mathcal{C}_{\boldsymbol{Z}}$ is $O(n_{\boldsymbol{Z}})$.

Now, consider the following probabilistic circuit over $\boldsymbol{V} = \boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z}$ (where $\times, +$ represent product, sum nodes respectively):

$$\mathcal{C} := \mathbb{1}_y \times (\mathbb{1}_x \times \mathbb{1}_{\boldsymbol{Z}} + \mathbb{1}_{\neg x} \times \mathbb{1}_{\boldsymbol{Z}}) + \mathbb{1}_{\neg y} \times (\mathbb{1}_x \times \mathcal{C}_{\boldsymbol{Z}} + \mathbb{1}_{\neg x} \times \mathbb{1}_{\boldsymbol{Z}}) \tag{B.10}$$

$\mathcal{C}$ is structured decomposable as all of the product units with the same scope in the equation above decompose in the same way, and we have seen that $\mathbb{1}_{\boldsymbol{Z}}$ and $\mathcal{C}_{\boldsymbol{Z}}$ are structured decomposable with respect to the same vtree. It is also smooth and deterministic as the individual circuits $\mathbb{1}_{\boldsymbol{Z}}$ and $\mathcal{C}_{\boldsymbol{Z}}$ are smooth and deterministic, and the sum nodes in the equation satisfy determinism by the fact that $(\mathbb{1}_y, \mathbb{1}_{\neg y})$ and

$(\mathbb{1}_x, \mathbb{1}_{\neg x})$ have disjoint support. Finally, as the sizes of $\mathbb{1}_{\boldsymbol{Z}}$ and $\mathcal{C}_{\boldsymbol{Z}}$ are $O(n_{\boldsymbol{Z}})$, $|\mathcal{C}(\boldsymbol{V}))|$ is also $O(n_{\boldsymbol{Z}})$.

Now, we show that the backdoor query on $\mathcal{C}$ is equivalent to solving the corresponding `EXPLR` problem. First, we derive expressions for all of the individual components of the backdoor formula on the circuit $\mathcal{C}$, by evaluating according to Equation B.10:

$$\begin{aligned}
p_{\mathcal{C}}(x, y, \boldsymbol{Z}) &= p_{\mathbb{1}_y}(y) \times (p_{\mathbb{1}_x}(x) \times p_{\mathbb{1}_{\boldsymbol{Z}}}(\boldsymbol{Z}) + p_{\mathbb{1}_{\neg x}}(x) \times p_{\mathbb{1}_{\boldsymbol{Z}}}(\boldsymbol{Z})) \\
&\quad + p_{\mathbb{1}_{\neg y}}(y) \times (p_{\mathbb{1}_x}(x) \times p_{\mathcal{C}_{\boldsymbol{Z}}}(\boldsymbol{Z}) + p_{\mathbb{1}_{\neg x}}(x) \times p_{\mathbb{1}_{\boldsymbol{Z}}}(\boldsymbol{Z})) \\
&= 1 \times (1 \times 1 + 0 \times 1) + 0 \times (1 \times 1 + 0 \times 1) \\
&= 1 \\
p_{\mathcal{C}}(x, \boldsymbol{Z}) &= p_{\mathcal{C}}(x, y, \boldsymbol{Z}) + p_{\mathcal{C}}(x, \neg y, \boldsymbol{Z}) \\
&= 1 + \mathcal{C}_{\boldsymbol{Z}}(\boldsymbol{Z}) \\
p_{\mathcal{C}}(\boldsymbol{Z}) &= p_{\mathcal{C}}(x, \boldsymbol{Z}) + p_{\mathcal{C}}(\neg x, y, \boldsymbol{Z}) + p_{\mathcal{C}}(\neg x, \neg y, \boldsymbol{Z}) \\
&= p_{\mathcal{C}}(x, \boldsymbol{Z}) + 1 + 1 \\
&= p_{\mathcal{C}}(x, \boldsymbol{Z}) + 2
\end{aligned}$$

The backdoor query for $\mathcal{C}$ can then be expressed as

$$\begin{aligned}
p_{\boldsymbol{x}}(\boldsymbol{y}) &= \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}) p_{\mathcal{C}}(y|x, \boldsymbol{Z}) = \sum_{\boldsymbol{Z}} (p_{\mathcal{C}}(x, \boldsymbol{Z}) + 2) \frac{p_{\mathcal{C}}(x, y, \boldsymbol{Z})}{p_{\mathcal{C}}(x, \boldsymbol{Z})} = \sum_{\boldsymbol{Z}} \left[1 + \frac{2}{1 + p_{\mathcal{C}_{\boldsymbol{Z}}}(\boldsymbol{Z})} \right] \\
&= 2^{n_{\boldsymbol{Z}}} + 2 \sum_{\boldsymbol{Z}} \frac{1}{1 + e^{-(w_0 + \sum_i w_i Z_i)}}
\end{aligned}$$

Thus, if we can compute the backdoor query for $\mathcal{C}$, then we can compute the given `EXPLR` problem, completing the reduction. □

Proposition 5.2 (Scope of Structured Decomposable and Smooth Circuits). *Given a structured decomposable and smooth circuit $\mathcal{C}$ respecting vtree v, with at least one product node, every node N in the circuit has a corresponding vtree node m such that $\phi(N) = \phi(m)$.*

Proof. By structured decomposability, the scope of each product node matches the scope of a vtree node, and the scope of its children match the scope of the vtree node's children. By smoothness, the scope of a sum node is equal to the scope of each of its children. Thus, if there is at least one product node, the scope of the root sum node is given by the the scope of any immediate descendant product node (that is, any product node such that the path between the root and product node does not contain another product node).

We can then recursively apply the above rules for the scope of the children of sum and product nodes starting from the root, which ensures that the scope of all nodes matches the scope of some vtree node. $\square$

Proposition 5.3 (Conflicting $\boldsymbol{Q}$-Determinisms for Sum Nodes)**.** *Let T be a non-trivial*[2] *sum node T, and let $\boldsymbol{Q}, \boldsymbol{Q}'$ be sets of variables such that neither is a subset of the other. Then, if T is both $\boldsymbol{Q}$-deterministic and $\boldsymbol{Q}'$-deterministic, but not $(\boldsymbol{Q} \cap \boldsymbol{Q}')$-deterministic, it cannot have full support, i.e. supp$(T) \subset$ val$(\phi(T))$.*

Proof. Since the sum node T is non-trivial, it has at least two children. Let N_1, N_2 be two distinct children of T. Let I_1, I_2 be the sets of values $\boldsymbol{q}$ of $\boldsymbol{Q}$ such that $p_{N_1}(\boldsymbol{q}) > 0, p_{N_2}(\boldsymbol{q}) > 0$ respectively. Define I'_1, I'_2 similarly for $\boldsymbol{Q}'$. Note that, by $\boldsymbol{Q}$-determinism and $\boldsymbol{Q}'$-determinism, I_1, I_2 are disjoint, and similarly I'_1, I'_2 are disjoint.

Now, we claim that there exists values $\boldsymbol{q} \in I_1$ and $\boldsymbol{q}' \in I'_2$ such that they agree over the intersection $\boldsymbol{Q} \cap \boldsymbol{Q}'$. If not, then p_{N_1} and p_{N_2} are non-zero for disjoint subsets of values of $(\boldsymbol{Q} \cap \boldsymbol{Q}')$, which implies $(\boldsymbol{Q} \cap \boldsymbol{Q}')$-determinism, which is a contradiction of the assumption of the Proposition. Now consider the value $\boldsymbol{q} \cup \boldsymbol{q}'$ of $\boldsymbol{Q} \cup \boldsymbol{Q}'$. $p_{N_1}(\boldsymbol{q} \cup \boldsymbol{q}') = 0$ since $p_{N_1}(\boldsymbol{q}') = 0$ (by the disjointness of I'_1, I'_2), and similarly $p_{N_2}(\boldsymbol{q} \cup \boldsymbol{q}') = 0$ since $p_{N_2}(\boldsymbol{q}) = 0$. For any other child N_3 of T, we have that I_1 and I_3 are disjoint by $\boldsymbol{Q}$-determinism, so $p_{N_3}(\boldsymbol{q}) = 0$ and we get $p_{N_3}(\boldsymbol{q} \cup \boldsymbol{q}') = 0$. Putting it all together, $p_T(\boldsymbol{q} \cup \boldsymbol{q}') = 0$ and thus T does not have full support.

$\square$

Proposition 5.4 (Superset $\boldsymbol{Q}$-Determinisms for Sum Nodes)**.** *Suppose that a sum node T is $\boldsymbol{Q}$-deterministic. Then it is also $\boldsymbol{Q}'$-deterministic for any $\boldsymbol{Q} \subseteq \boldsymbol{Q}' \subseteq \boldsymbol{V}$.*

[2]A sum node is non-trivial if it has more than one child.

Proof. By definition, a sum node T is $\boldsymbol{Q}$-deterministic if for any instantiation $\boldsymbol{q}$ of $\boldsymbol{Q}$, at most one of its children N_i evaluate to a nonzero output under $\boldsymbol{q}$. If $\boldsymbol{Q}' \supseteq \boldsymbol{Q}$, then any instantiation $\boldsymbol{q}'$ of $\boldsymbol{Q}'$ will imply a specific instantiation of $\boldsymbol{q}$, and so at most one of the children of T evaluate to a nonzero output under $\boldsymbol{q}'$. More formally, $p_{N_i}(\boldsymbol{q}) = \sum_{\boldsymbol{Q}' \setminus \boldsymbol{Q}} p_{N_i}(\boldsymbol{q}, \boldsymbol{Q}' \setminus \boldsymbol{Q}) = 0 \implies p_{N_i}(\boldsymbol{q}') = 0$. □

Proposition 5.5 (Validity of Implied $\boldsymbol{Q}$-Determinisms)**.** *For any PC $\mathcal{C}$ respecting md-vtree w, we have that $\mathcal{Q}(w) \subseteq \mathcal{Q}(\mathcal{C})$.*

Proof. Since $\mathcal{C}$ respects w, every sum unit $T \in \mathcal{C}$ has scope $\phi(T) = \phi(m)$ for some $m \in M$, and T is $\psi(m)$-deterministic. Further, since w implies $\boldsymbol{Q}$-determinism, we have that $\phi(m) \cap \boldsymbol{Q} = \emptyset$, or else $\boldsymbol{Q} \supseteq \psi(m)$. Combining these statements, we see that for all sum units $T \in \mathcal{C}$, either $\phi(T) \cap \boldsymbol{Q} = \emptyset$, or else T is $\psi(m)$-deterministic and thus (by Proposition 5.4) $\boldsymbol{Q}$-deterministic. This shows that $\mathcal{C}$ is $\boldsymbol{Q}$-deterministic. □

The following theorem justifies the intuition that having smaller labels $\psi(m)$ corresponds to a stronger restriction on the circuit, such that less circuits respect the md-vtree, but more marginal determinisms are implied:

Theorem 5.2 (Generality-Tractability Tradeoff)**.** *Let $w = (v, \psi)$ and $w' = (v, \psi')$ be two md-vtrees, such that $\psi'(m) \supseteq \psi(m)$ for all $m \in M$. Then we have that $\mathcal{Q}(w) \supseteq \mathcal{Q}(w')$, and $\boldsymbol{\mathcal{C}}_w \subseteq \boldsymbol{\mathcal{C}}_{w'}$.*

Proof. For the first part, suppose $\boldsymbol{Q} \in \mathcal{Q}(w')$. Then for all $m \in M$, it holds that $\phi(m) \cap \boldsymbol{Q} = \emptyset$, or else $\boldsymbol{Q} \supseteq \psi'(m)$. Since $\psi'(m) \supseteq \psi(m)$, it holds that $\phi(m) \cap \boldsymbol{Q} = \emptyset$, or else $\boldsymbol{Q} \supseteq \psi(m)$ also, so $\boldsymbol{Q} \in \mathcal{Q}(w)$. This shows that $\mathcal{Q}(w) \supseteq \mathcal{Q}(w')$.

For the second part, suppose that $\mathcal{C} \in \boldsymbol{\mathcal{C}}_w$. Then, for any sum unit $T \in \mathcal{C}$, there is an $m \in M$ that T is marginally deterministic w.r.t. $\psi(m)$. As $\psi'(m) \supseteq \psi(m)$, this means that T is also marginally deterministic w.r.t. $\psi'(m)$. Thus $\mathcal{C}$ respects w' also, i.e. $\mathcal{C} \in \boldsymbol{\mathcal{C}}_{w'}$. □

Theorem 5.3 (Admissible Labelling Functions are Regular)**.** *Given a vtree v, let ψ be any non-regular labelling function. Then there exists a regular labelling function ψ' such that (i) $\psi'(m) \supseteq \psi(m)$ for all $m \in M$, (ii) $\exists m^* \in M$ such that $\psi'(m) \supset \psi(m)$, and $\mathcal{Q}(w') = \mathcal{Q}(w)$, where $w := (v, \psi), w' := (v, \psi')$.*

We will prove this Theorem explicitly by constructing a regular labelling function with these properties. To do this, we prove two Lemmas which define operations which do not change $\mathcal{Q}(w)$, while keeping the same or increasing the label set $\psi(m)$ for each m; the result of iterative application of the two operations being a regular labelling function.

Definition B.1 (Expand Child Labels)**.** *Given a md-vtree $w = (v, \psi)$, and any vtree nodes m_{pa^*}, m_{ch^*} such that m_{ch^*} is a child of m_{pa^*}, the operation $\texttt{ECL}(w, m_{pa^*}, m_{ch^*})$ returns a new md-vtree $w' = (v, \psi')$, defined as follows:*

$$\psi'(m) = \begin{cases} \psi(m_{ch^*}) \cup (\psi(m_{pa^*}) \cap \phi(m_{ch^*})) & \text{if } m = m_{ch} \\ \psi(m) & \text{otherwise} \end{cases} \tag{B.11}$$

Lemma B.1. *The output $w' = \texttt{ECL}(w, m_{pa^*}, m_{ch^*})$ satisfies $\mathcal{Q}(w') = \mathcal{Q}(w)$, and for all $m \in M$, $\psi'(m) \supseteq \psi(m)$.*

Proof. The only difference between w and w' is the label of m_{ch^*}. Suppose that $\boldsymbol{Q} \in \mathcal{Q}(w)$, then we have that either $\phi(m_{\text{ch}^*}) \cap \boldsymbol{Q} = \emptyset$, or else $\boldsymbol{Q} \supseteq \psi(m_{\text{ch}^*})$. In the former case, since the vtrees and thus scopes are the same between w, w, it follows that $\boldsymbol{Q} \in \mathcal{Q}(w')$ also. In the latter case, since the scope of the parent $\phi(m_{\text{pa}^*}) \supseteq \phi(m_{\text{ch}^*})$, $\boldsymbol{Q}$ overlaps with the parent scope as well, implying that that $\boldsymbol{Q} \supseteq \psi(m_{\text{pa}^*})$. Thus we have that $\boldsymbol{Q} \supseteq \psi(m_{\text{ch}^*}) \cup \psi(m_{\text{pa}^*}) \supseteq \psi(m_{\text{ch}^*}) \cup (\psi(m_{\text{pa}^*}) \cap \phi(m_{\text{ch}^*})) = \psi'(m_{\text{ch}^*})$. Thus, $\boldsymbol{Q} \in \mathcal{Q}(w')$ also. That is, $\mathcal{Q}(w) \subseteq \mathcal{Q}(w')$.

To complete the result, note that $\psi(m) \subseteq \psi'(m)$ for all vtree nodes m. Thus by Theorem 5.2, it follows that $\mathcal{Q}(w) \supseteq \mathcal{Q}(w')$ and $\boldsymbol{\mathcal{C}}_w \subseteq \boldsymbol{\mathcal{C}}_{w'}$. Combining with the paragraph above we have shown that $\mathcal{Q}(w) = \mathcal{Q}(w')$.

□

Intuitively, this operation "pushes down" elements of $\psi(m_{\text{pa}^*})$ to its children. If we apply this operation to all pairs of parent/child vtree nodes $(m_{\text{pa}}, m_{\text{ch}})$, then it can be seen that the new labels will have the property that *all elements of the parent label that are contained in the scope of a child, will be in the label of that child.* More formally, $\psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}})$. This is the starting point for the next operation:

Definition B.2 (Expand Parent Labels)**.** *Let $w = (v, \psi)$ be a md-vtree such that $\psi(m_{pa}) \cap \phi(m_{ch}) = \psi(m_{pa}) \cap \psi(m_{ch})$ holds for all pairs of parents m_{pa} and children m_{ch}. Then, given any non-leaf vtree node m_{pa^*}, the operation $\texttt{EPL}(w, m_{pa^*})$ returns a new md-vtree $w' = (v, \psi')$, defined as follows:*

$$\psi'(m) = \begin{cases} \bigcup_{m_{ch^*} \in M_{active}} \psi(m_{ch^*}) & \text{if } m = m_{pa} \\ \psi(m) & \text{otherwise} \end{cases} \tag{B.12}$$

where we define $M_{active} = \{m_{ch^*} | m_{ch^*} \in M_{ch^*}, \psi(m_{ch^*}) \cap \psi(m_{pa^*}) \neq \emptyset\}$ to be the set of all children whose labellings have non-empty intersection with the labelling of the parent.

Lemma B.2. *The output $w' = \texttt{EPL}(w, m_{pa^*})$ satisfies $\mathcal{Q}(w') = \mathcal{Q}(w)$, and for all $m \in M$, $\psi'(m) \supseteq \psi(m)$.*

Further, the property that $\psi'(m_{pa}) \cap \phi(m_{ch}) = \psi'(m_{pa}) \cap \psi'(m_{ch})$ holds for all pairs of parents m_{pa} and children m_{ch} in w' (i.e. is maintained in w').

Proof. Firstly, we show that $\psi(m_{\text{pa}^*}) \subseteq \psi'(m_{\text{pa}^*})$. This follows by taking a union over children of both sides of the assumption $\psi(m_{\text{pa}^*}) \cap \phi(m_{\text{ch}^*}) = \psi(m_{\text{pa}^*}) \cap \psi(m_{\text{ch}^*})$, where the LHS becomes $\bigcup_{m_{\text{ch}^*} \in M_{\text{ch}^*}} (\psi(m_{\text{pa}^*}) \cap \phi(m_{\text{ch}^*})) = \psi(m_{\text{pa}^*}) \cap \bigcup_{m_{\text{ch}^*} \in M_{\text{ch}^*}} \phi(m_{\text{ch}^*}) = \psi(m_{\text{pa}^*}) \cap \phi(m_{\text{pa}^*}) = \psi(m_{\text{pa}^*})$, and the RHS becomes $\bigcup_{m_{\text{ch}^*} \in M_{\text{ch}^*}} (\psi(m_{\text{pa}^*}) \cap \psi(m_{\text{ch}^*})) = \bigcup_{m_{\text{ch}^*} \in M_{\text{active}}} (\psi(m_{\text{pa}^*}) \cap \psi(m_{\text{ch}^*})) = \psi(m_{\text{pa}^*}) \cap \bigcup_{m_{\text{ch}^*} \in M_{\text{active}}} \psi(m_{\text{ch}^*}) \subseteq \psi'(m_{\text{pa}^*})$. Thus $\psi(m) \subseteq \psi'(m)$ for all vtree nodes m, and by Theorem 5.2, it follows that $\mathcal{Q}(w) \supseteq \mathcal{Q}(w')$ and $\boldsymbol{\mathcal{C}}_w \subseteq \boldsymbol{\mathcal{C}}_{w'}$.

Now suppose $\boldsymbol{Q} \in \mathcal{Q}(w)$. We consider two cases. Firstly, if $\phi(m_{\text{pa}^*}) \cap \boldsymbol{Q} = \emptyset$, then since vtrees and scopes are the same between w, w', we have $\boldsymbol{Q} \in \mathcal{Q}(w')$. Otherwise, we have $\phi(m_{\text{pa}^*}) \cap \boldsymbol{Q} \neq \emptyset$ and $\boldsymbol{Q} \supseteq \psi(m_{\text{pa}^*})$. Now, for those children in M_{active}, we have that $\psi(m_{\text{ch}^*}) \cap \psi(m_{\text{pa}^*}) \neq \emptyset$ and so since $\boldsymbol{Q} \supseteq \psi(m_{\text{pa}^*})$ and $\phi(m_{\text{ch}^*}) \supseteq \psi(m_{\text{ch}^*})$, we have $\boldsymbol{Q} \cap \phi(m_{\text{ch}^*}) \neq \emptyset$. The marginal determinism property on these children then implies that $\boldsymbol{Q} \supseteq \psi(m_{\text{ch}^*})$ for all $m_{\text{ch}^*} \in M_{\text{active}}$; and so $\boldsymbol{Q} \supseteq \bigcup_{m_{\text{ch}^*} \in M_{\text{active}}} \psi(m_{\text{ch}^*}) = \psi'(m_{\text{pa}^*})$. This shows that $\boldsymbol{Q} \in \mathcal{Q}(w')$ also. This gives $\mathcal{Q}(w) \subseteq \mathcal{Q}(w')$, and combined with the previous result, $\mathcal{Q}(w) = \mathcal{Q}(w')$.

Finally, we show that the property that $\psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}})$ holds for all pairs of parents m_{pa} and children m_{ch} in w'. The only label which has changed is that of m_{pa}, so we need only consider the pairs $(m_{\text{pa}}, m_{\text{ch}})$ with either (a) $m_{\text{pa}} = m_{\text{pa}^*}$ and m_{ch} is a child of m_{pa^*} or (b) $m_{\text{ch}} = m_{\text{pa}^*}$ and m_{pa} is the parent of m_{pa^*}.

- In case (a), by definition we have that $\psi'(m_{\text{pa}}) = \psi'(m_{\text{pa}^*}) = \bigcup_{m_{\text{ch}^*} \in M_{\text{active}}} \psi(m_{\text{ch}^*})$, and $\psi'(m_{\text{ch}}) = \psi'(m_{\text{ch}})$. If m_{ch} is an active child of m^*_{pa}, then we have that the LHS of the property $\psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \bigcup_{m_{\text{ch}^*} \in M_{\text{active}}} \psi(m_{\text{ch}^*}) \cap \phi(m_{\text{ch}}) = \psi(m_{\text{ch}})$, and the RHS of the property $\psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}}) = \bigcup_{m_{\text{ch}^*} \in M_{\text{active}}} \psi(m_{\text{ch}^*}) \cap \psi'(m_{\text{ch}}) = \psi(m_{\text{ch}})$. If m_{ch} is not an active child of m^*_{pa}, then both sides of the property correspond to the empty set.

- In case (b), by definition we have $\psi'(m_{\text{pa}}) = \psi(m_{\text{pa}})$, and $\psi'(m_{\text{ch}}) = \psi'(m_{\text{pa}^*})$. We have shown above that $\psi(m_{\text{pa}^*}) \subseteq \psi'(m_{\text{pa}^*})$, so $\psi(m_{\text{ch}}) \subseteq \psi'(m_{\text{ch}})$. By the precondition for applying the `EPL` operation, we have that $\psi(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \psi(m_{\text{pa}}) \cap \psi(m_{\text{ch}})$. Substituting, we get $\psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \psi'(m_{\text{pa}}) \cap \psi(m_{\text{ch}}) \subseteq \psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}})$. The other direction $\psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) \supseteq \psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}})$ is immediate as the label of a node is contained in its scope.

Thus, we have shown that $\psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}})$ holds for all pairs of parents m_{pa} and children m_{ch} in w', concluding the proof. $\square$

Intuitively, this operation "pulls up" elements $\psi(m_{\text{ch}})$ of the *active* children to the parent. After applying this operation to all nodes, we obtain a regular md-vtree, which has the same set $\mathcal{Q}$ of marginal determinisms, and is at least as expressive. More formally:

Proof. (of Theorem) Starting from w, apply the `ECL` operation to each pair of parent and child nodes, in a topological order starting from the root. For each $m_{\text{pa}}, m_{\text{ch}}$ pair, we have that $\psi'(m_{\text{pa}}) = \psi(m_{\text{pa}})$ and $\psi'(m_{\text{ch}}) = \psi(m_{\text{ch}}) \cup (\psi(m_{\text{pa}}) \cap \phi(m_{\text{ch}}))$ by definition of the operation. Then $\psi'(m_{\text{pa}}) \cap \psi'(m_{\text{ch}}) = \psi(m_{\text{pa}}) \cap (\psi(m_{\text{ch}}) \cup (\psi(m_{\text{pa}}) \cap \phi(m_{\text{ch}}))) = (\psi(m_{\text{pa}}) \cap \psi(m_{\text{ch}})) \cup (\psi(m_{\text{pa}}) \cap \phi(m_{\text{ch}})) = \psi(m_{\text{pa}}) \cap \phi(m_{\text{ch}}) = \psi'(m_{\text{pa}}) \cap \phi(m_{\text{ch}})$, which is the required property for applying the `EPL` operation. As we proceed in a topological order, and the operation only modifies the label of the child, it follows that the property $\psi'(m_{\mathrm{pa}}) \cap \psi'(m_{\mathrm{ch}}) = \psi'(m_{\mathrm{pa}}) \cap \phi(m_{\mathrm{ch}})$ holds for all parent/children pairs at the end.

This allows us to apply the EPL operation. We apply this operation to every non-leaf node, in a reverse topological order from the leaves to the root. The precondition for applying the operation holds at all points due to the result of Lemma B.2. This operation only modifies the label of the parent, and so after we have modified all the labels, we have the property that $\psi'(m_{\mathrm{pa}}) = \bigcup_{m_{\mathrm{ch}} \in M_{\mathrm{active}}} \psi(m_{\mathrm{ch}})$ for every non-leaf node m_{pa}. That is, it satisfies the conditions to be a regular md-vtree, i.e. $\psi'(m_{\mathrm{pa}}) = \emptyset, \psi'(m_1), \psi'(m_2)$, or $\psi'(m_1) \cup \psi'(m_2)$, where m_1, m_2 are the children of m_{pa}.

We have already shown that applications of ECL and EPL do not change the implied marginal determinisms (i.e. point (iii) of the Theorem), and keep the same or increase the size of the label (i.e. point (i) of the Theorem). The strictness of the label change (point (ii) of the Theorem) follows from the fact that the original labelling function was not regular, while the output labelling function is regular; thus, there must be a difference in the label of at least one vtree node m. □

Proposition 5.6 (Correctness and Admissibility of $\boldsymbol{S}$-constrained Label)**.** *For any vtree v and set of marginal determinisms $\boldsymbol{S}$, the md-vtree $w := (v, \psi_{\boldsymbol{S}})$ satisfies $\mathcal{Q}(w) \supseteq \boldsymbol{S}$. Further, $\psi_{\boldsymbol{S}}$ is admissible.*

Proof. Recall the definition of the $\boldsymbol{S}$-constrained label:

$$\psi_{\boldsymbol{S}}(m) = \begin{cases} U & \text{if } \boldsymbol{Q} \cap \phi(m) = \emptyset \ \ \forall \boldsymbol{Q} \in \boldsymbol{S} \\ \phi(m) \cap (\bigcap_{\boldsymbol{Q} \in \boldsymbol{S}: \boldsymbol{Q} \cap \phi(m) \neq \emptyset} \boldsymbol{Q}) & \text{otherwise} \end{cases} \tag{B.13}$$

For correctness, we need to show that, for each $m \in M$ and $\boldsymbol{Q} \in \boldsymbol{S}$, if $\phi(m) \cap \boldsymbol{Q} \neq \emptyset$, then $\boldsymbol{Q} \supseteq \psi_{\boldsymbol{S}}(m)$. This is immediate by the definition of the otherwise clause.

For admissibility, suppose for contradiction that there was a labelling function ψ' such that $\psi'(m) \supseteq \psi_{\boldsymbol{S}}(m)$ for all nodes m, and $\psi'(m^*) \supset \psi_{\boldsymbol{S}}(m^*)$ for some node m^*, and $\mathcal{Q}(\psi') \supseteq \mathcal{Q}(\psi_{\boldsymbol{S}})$. Since the universal set has no strict superset, this must mean that $\psi_{\boldsymbol{S}}(m^*) = \phi(m^*) \cap (\bigcap_{\boldsymbol{Q} \in \boldsymbol{S}: \boldsymbol{Q} \cap \phi(m^*) \neq \emptyset} \boldsymbol{Q})$.

Let $\boldsymbol{Q}^*$ be some $\boldsymbol{Q} \in \boldsymbol{S}$ such that $\boldsymbol{Q} \cap \phi(m^*) \neq \emptyset$. Then $\psi'(m^*)$ must contain an variable $V \in \phi(m^*)$ such that $V \notin \psi'(m^*)$. But this means that ψ' does not imply $\boldsymbol{Q}$-determinism. Thus $\psi'(m^*) \not\supset \psi_{\boldsymbol{S}}(m^*)$ and we have a contradiction.

□

Theorem 5.5 (MD-calculus)**.** *The conditions in Table 5.2 hold.*

Proof. See section B.2.2. □

Proposition 5.8 (Tractable Frontdoor Adjustment)**.** *Let $\mathcal{C}$ be a structured decomposable and smooth circuit. Then, in the frontdoor case:*

- *Computing the interventional distribution $p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^3)$ time, if $\mathcal{C}$ is additionally $\boldsymbol{X}$-deterministic and $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic;*
- *Computing the instantiated interventional distribution $p_{\mathcal{C},\boldsymbol{x}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^3)$ time, if $\mathcal{C}$ is additionally $(\boldsymbol{X} \cup \boldsymbol{Z})$-deterministic.*

Proof. The frontdoor formula is given by:

$$p_{\mathcal{C},\boldsymbol{X}}(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z}|\boldsymbol{X}) \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y}|\boldsymbol{X}', \boldsymbol{Z}) \tag{B.14}$$

The interventional distribution can be expressed through the composition:

$$\mathcal{C}_1 = \texttt{MARG}(\mathcal{C}; \boldsymbol{V} \setminus (\boldsymbol{X}, \boldsymbol{Z})); p_{\mathcal{C}_1}(\boldsymbol{X}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Z}) \tag{B.15}$$

$$\mathcal{C}_2 = \texttt{COND}(\mathcal{C}_1; \boldsymbol{X}); p_{\mathcal{C}_2}(\boldsymbol{X}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z}|\boldsymbol{X}) \tag{B.16}$$

$$\mathcal{C}_3 = \texttt{BACKDOOR}(\mathcal{C}; \boldsymbol{Z}, \boldsymbol{Y}, \boldsymbol{X}); p_{\mathcal{C}_3}(\boldsymbol{Y}, \boldsymbol{Z}) = \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y}|\boldsymbol{X}', \boldsymbol{Z}) \tag{B.17}$$

$$\mathcal{C}_4 = \texttt{PROD}(\mathcal{C}_2, \mathcal{C}_3); p_{\mathcal{C}_4}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Z}) \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y}|\boldsymbol{X}', \boldsymbol{Z}) \tag{B.18}$$

$$\mathcal{C}_5 = \texttt{MARG}(\mathcal{C}_4; \boldsymbol{Z}); p_{\mathcal{C}_5}(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{X}, \boldsymbol{Z}) \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y}|\boldsymbol{X}', \boldsymbol{Z}) \tag{B.19}$$

where we have written `BACKDOOR` to represent the composition of operations in the backdoor formula (note that $\boldsymbol{X}$ takes the position of $\boldsymbol{Z}$ in the backdoor formula here). As before, we can analyze the tractability of the pipeline by propagating backwards from the deterministic operations. In this case, the backdoor and conditioning operations are deterministic. For the backdoor operation, we saw that $(\boldsymbol{Z} \cup \boldsymbol{X})$-determinism was sufficient, while for conditioning, we need $\boldsymbol{X}$-determinism. Working back through the marginalization operation $\texttt{MARG}(\mathcal{C}; \boldsymbol{V} \setminus (\boldsymbol{X}, \boldsymbol{Z}))$ does not change the marginal determinism (by the MD-calculus), so the final requirement (sufficient condition) is for $\mathcal{C}$ to be $(\boldsymbol{Z} \cup \boldsymbol{X})$-deterministic and $\boldsymbol{X}$-deterministic. The complexity follows from the fact that the backdoor formula produces a circuit that is $O(|\mathcal{C}|^2)$ in size, and so the time complexity of the product is $O(|\mathcal{C}| * |\mathcal{C}^2|) = O(|\mathcal{C}|^3)$.

For the instantiated case, we have:

$$\mathcal{C}_1 = \texttt{INST}(\mathcal{C}; \boldsymbol{x}); p_{\mathcal{C}_2}(\boldsymbol{V} \setminus \boldsymbol{X}) = p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{V} \setminus \boldsymbol{X}) \tag{B.20}$$

$$\mathcal{C}_2 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{V} \setminus (\boldsymbol{X}, \boldsymbol{Z})); p_{\mathcal{C}_2}(\boldsymbol{X}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Z}) \tag{B.21}$$

$$p_{\mathcal{C}_3}(\boldsymbol{X}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z} | \boldsymbol{x}) \tag{B.22}$$

$$\mathcal{C}_4 = \texttt{BACKDOOR}(\mathcal{C}; \boldsymbol{Z}, \boldsymbol{Y}, \boldsymbol{X}); p_{\mathcal{C}_4}(\boldsymbol{Y}, \boldsymbol{Z}) = \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{X}', \boldsymbol{Z}) \tag{B.23}$$

$$\mathcal{C}_5 = \texttt{PROD}(\mathcal{C}_3, \mathcal{C}_4); p_{\mathcal{C}_5}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{Z} | \boldsymbol{x}) \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{X}', \boldsymbol{Z}) \tag{B.24}$$

$$\mathcal{C}_6 = \texttt{MARG}(\mathcal{C}_5; \boldsymbol{Z}); p_{\mathcal{C}_6}(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{\boldsymbol{Z}} p_{\mathcal{C}}(\boldsymbol{Z} | \boldsymbol{x}) \sum_{\boldsymbol{X}'} p_{\mathcal{C}}(\boldsymbol{X}') p_{\mathcal{C}}(\boldsymbol{Y} | \boldsymbol{X}', \boldsymbol{Z}) \tag{B.25}$$

Note that to obtain the third line, rather than conditioning symbolically on $\boldsymbol{X}$, we can just divide through by the scalar $p_{\mathcal{C}}(\boldsymbol{x})$; this does not impose any marginal determinism requirements. Thus, we only require $\mathcal{C}$ to be $(\boldsymbol{Z} \cup \boldsymbol{X})$-deterministic (due to the backdoor) in this case. □

Proposition 5.9 (Tractable Napkin)**.** *Let $\mathcal{C}$ be a structured decomposable and smooth circuit. Then, in the napkin case:*

- *Computing the instantiated interventional distribution $p_{\mathcal{C},\boldsymbol{x}}(\boldsymbol{Y})$ as a structured decomposable and smooth circuit is tractable in $O(|\mathcal{C}|^2)$ time, if $\mathcal{C}$ is additionally $(\boldsymbol{W} \cup \boldsymbol{X}' \cup \boldsymbol{Z})$-deterministic for some $\boldsymbol{X}' \subseteq \boldsymbol{X}$.*

Proof. The instantiated napkin formula is:

$$p_{\mathcal{C},\boldsymbol{x}}(\boldsymbol{Y}) = \frac{\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{x} | \boldsymbol{W}, \boldsymbol{Z}) p_{\mathcal{C}}(\boldsymbol{W})} \tag{B.26}$$

Recall that, if the interventional distribution is identifiable through the napkin formula, then this formula is valid (i.e. represents the same, correct interventional distribution

$p_{\boldsymbol{X}}(\boldsymbol{Y})$ for any value of $\boldsymbol{Z}$; thus, we can instantiate with an arbitrary value $\boldsymbol{z}$.

$$\mathcal{C}_1 = \texttt{INST}(\mathcal{C}; \boldsymbol{x}); p_{\mathcal{C}_1}(\boldsymbol{V} \setminus \boldsymbol{X}) = p_{\mathcal{C}}(\boldsymbol{V} \setminus \boldsymbol{X}, \boldsymbol{x}) \tag{B.27}$$

$$\mathcal{C}_2 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{V} \setminus (\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})); p_{\mathcal{C}_2}(\boldsymbol{W}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{W}, \boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{Z}) \tag{B.28}$$

$$\mathcal{C}_3 = \texttt{COND}(\mathcal{C}_2; \boldsymbol{W}, \boldsymbol{Z}); p_{\mathcal{C}_3}(\boldsymbol{W}, \boldsymbol{Y}, \boldsymbol{Z}) = p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{Z}) \tag{B.29}$$

$$\mathcal{C}_4 = \texttt{INST}(\mathcal{C}_3; \boldsymbol{z}); p_{\mathcal{C}_4}(\boldsymbol{W}, \boldsymbol{Y}) = p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{z}) \tag{B.30}$$

$$\mathcal{C}_5 = \texttt{MARG}(\mathcal{C}; \boldsymbol{V} \setminus \boldsymbol{W}); p_{\mathcal{C}_5}(\boldsymbol{W}) = p_{\mathcal{C}}(\boldsymbol{W}) \tag{B.31}$$

$$\mathcal{C}_6 = \texttt{PROD}(\mathcal{C}_4, \mathcal{C}_5); p_{\mathcal{C}_6}(\boldsymbol{W}, \boldsymbol{Y}) = p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{z}) p_{\mathcal{C}}(\boldsymbol{W}) \tag{B.32}$$

$$\mathcal{C}_7 = \texttt{MARG}(\mathcal{C}_6, \boldsymbol{W}); p_{\mathcal{C}_7}(\boldsymbol{Y}) = \sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{z}) p_{\mathcal{C}}(\boldsymbol{W}) \tag{B.33}$$

$$p_{\mathcal{C}_8}(\boldsymbol{Y}) = \frac{\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{Y} | \boldsymbol{W}, \boldsymbol{z}) p_{\mathcal{C}}(\boldsymbol{W})}{\sum_{\boldsymbol{W}} p_{\mathcal{C}}(\boldsymbol{x} | \boldsymbol{W}, \boldsymbol{z}) p_{\mathcal{C}}(\boldsymbol{W})} \tag{B.34}$$

On the last line, the denominator is just a scalar so we can divide through without any marginal determinism requirements. The only deterministic operation is the conditioning operation, which requires $\mathcal{C}_2$ to be $(\boldsymbol{W} \cup \boldsymbol{Z})$-deterministic. Propagating backwards, we get:

1. **Requirement**: $\mathcal{C}_2 = \texttt{MARG}(\mathcal{C}_1; \boldsymbol{V} \setminus (\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}))$ is $(\boldsymbol{W} \cup \boldsymbol{Z})$-deterministic.

2. $\mathcal{C}_1 = \texttt{INST}(\mathcal{C}; \boldsymbol{x})$ is $(\boldsymbol{W} \cup \boldsymbol{Z})$-deterministic.

3. **Sufficient Condition**: $\mathcal{C}$ is $(\boldsymbol{W} \cup \boldsymbol{X}' \cup \boldsymbol{Z})$-deterministic for some $\boldsymbol{X}' \subseteq \boldsymbol{X}$.

□

B.2 Operations and MD-Calculus

In this section, we provide further details on the results in Chapter 5.4 regarding inference on md-vtrees. First, for the *forward* problem, we describe algorithms for soundly propagating the md-vtree forward under each basic circuit operation, as mentioned in Section 5.4.2. Then, for the *backward* problem, we analyze these algorithms to prove the MD-calculus for propagating marginal determinisms backwards through operations.

Notation In the rest of this section, we will write m_l, m_r (left, right) for the children of m instead of m_1, m_2; this is to avoid confusion with the notation $m^{(1)}, m^{(2)}$ which we use to denote vtree nodes in two different vtrees. We will use the shorthand $\phi_{\boldsymbol{W}}(m) := \phi(m) \cap \boldsymbol{W}$ to denote the scope of m *restricted to* $\boldsymbol{W}$.

B.2.1 Algorithms and the Forward Problem

For each of the basic operations, there exist efficient (polynomial time) algorithms for computing them on probabilistic circuits satisfying the requirement column in Table 5.1 [33, 185]. In this section, we will also describe, for each basic operation, an algorithm for computing the operation on md-vtrees w, that is a sound abstraction of the corresponding algorithm on circuits. By *sound*, we mean that, given an input md-vtree w and the output of the md-vtree algorithm w', it is guaranteed for any input PC respecting w, the output of the corresponding PC algorithm will respect w'.

The construction of these md-vtrees algorithms is based upon the corresponding PC algorithm. Thus, we present the algorithms as applying to both the md-vtree and PC. For convenience, we assume that the PC satisfies the following condition, which we call *exactly respecting* a md-vtree:

Definition B.3 (PC exactly respecting md-vtree). *A PC $\mathcal{C}$ exactly respects a md-vtree w if (1) it respects w and (2) the children of any sum node T corresponding to a non-leaf vtree node m are all product nodes P, where P has two children which are sum nodes, each corresponding to a child of m.*

PC architectures are typically designed with these alternating sum and product nodes, where the product nodes are binary; for example, both MDNets and PSDDs satisfy this property. Further, any PC which respects a md-vtree can be transformed into an equivalent PC which exactly respects the md-vtree, as follows. For every sum node T which has a sum node child T', we can directly attach the children of the T' to T (with the appropriate combination of weights). Then, for every product node P which has a product node child P', we can replace P' with a new single-child sum node T', which has P' as its child. The resulting circuit still encodes the same function, and has the same marginal determinisms as the original circuit.

Given that a PC exactly respects a md-vtree, for each non-leaf vtree node m, we can represent the corresponding PC layer simply as a vector of sum nodes $\boldsymbol{T}_m$ with length $K_m := |\boldsymbol{T}_m|$, and a weight/parameter matrix ω_m with shape (K_m, K_{m_1}, K_{m_2}), with the semantics that $p_{T_{m,i}}(\boldsymbol{V}) = \sum_{jk} \omega_{m,ijk} p_{T_{m_1,j}}(\boldsymbol{V}) p_{T_{m_2,k}}(\boldsymbol{V})$ (where m_1, m_2 are the children of m). Note that for any pair of sum nodes $T_{m_1,j}, T_{m_2,k}$ for which there isn't a product in the PC connected to $T_{m,i}$, we can simply set the weight $\omega_{m,ijk}$ to zero[3]. For leaf vtree nodes, the corresponding layer can consist of both sum and leaf nodes (with the sum nodes being mixtures over the leaf nodes, e.g. $0.7\mathbb{1}_{X=0} + 0.3\mathbb{1}_{X=1}$). In this case, we represent the sum and leaf nodes as a vector $\boldsymbol{T}_m$ with length $K_m := |\boldsymbol{T}_m|$, and ω_m, and a weight matrix ω_m, with $\omega_{m,ij} > 0$ iff $T_{m,j}$ is a leaf node that is a child of sum node $T_{m,i}$.

This characterization of a PC as a pair $\tau(m) := (\boldsymbol{T}_m, \omega_m)$ for each vtree node, which we call the *parameter function*, allows us to efficiently describe algorithms for the basic operations. Thus, in the algorithms below we will represent $\mathcal{C}$ exactly respecting some md-vtree using the triple $\mathcal{C} = (v, \psi, \tau)$, where v is the vtree, ψ is the labelling function, and τ the parameter function.

`MARG`$(\cdot; \boldsymbol{W})$ The marginalization algorithm is depicted in Algorithm B.1. For the marginalization operation, we can take advantage of the fact that marginalization commutes with both product and sum nodes in a decomposable and smooth PC (which is the basis of tractable marginal inference).

$$\sum_{\boldsymbol{W}} p_P(\phi(P)) = \sum_{\boldsymbol{W}} p_{N_1}(\phi(N_1)) p_{N_2}(\phi(N_2)) = (\sum_{\boldsymbol{W}} p_{N_1}(\phi(N_1)))(\sum_{\boldsymbol{W}} p_{N_2}(\phi(N_2))) \tag{B.35}$$

$$\sum_{\boldsymbol{W}} p_T(\phi(T)) = \sum_{\boldsymbol{W}} \sum_{N_i \in \mathrm{ch}(T)} \omega_i p_{N_i}(\phi(N_i)) = \sum_{N_i \in \mathrm{ch}(T)} \omega_i (\sum_{\boldsymbol{W}} p_{N_i}(\phi(N_i))) \tag{B.36}$$

where the last equality on the first line holds because $\phi(N_1) \cap \phi(N_2) = \emptyset$ by decomposability. This means that, in order to marginalize a circuit, we simply need to marginalize the leaf nodes. In Algorithm B.1, we show the (recursive) procedure of

[3]While this is sufficient to represent any PC exactly respecting an md-vtree, it may be inefficient to represent $\omega_{m,ijk}$ as a dense matrix if the connections in the PC are sparse, i.e. $\omega_{m,ijk} = 0$ for many i, j, k. In the evaluation of a sum vector as a function of its child sum vectors, we only require the sum $\sum_{jk} \omega_{m,ijk} p_{T_{m_1,j}} p_{T_{m_2,k}}$ to be computed, so this can be implemented in a sparse manner if that is more appropriate. Similar reasoning applies to the product algorithm.

marginalizing a circuit represented as (v, ψ, τ). In lines 3-5 we marginalize out $\boldsymbol{W}$ from the leaf nodes in the PC, and in lines 6-12, we handle non-leaf vtree nodes simply by copying the existing circuit. To update the md-vtree, in line 13, we update the scope of the vtree node, removing the marginalized variables, and in lines 14-17, we assign a label to the new vtree node m'.

The new label is justified as follows. Suppose we have a sum node $T \in T_m$, with children $N_1, ..., N_n$; by definition, T is marginally deterministic with respect to $\psi(m)$. After marginalization, the function encoded by each child N_i' satisfies $p_{N_i'}(\boldsymbol{q}) = \sum_{\boldsymbol{W}} p_{N_i}(\boldsymbol{q})$ for any value $\boldsymbol{q}$ of $\psi(m)$ by definition. Now:

- If $\psi(m) \cap \boldsymbol{W} = \emptyset$, then this is just proportional to $p_{N_i}(\boldsymbol{q})$ and so the marginalized support will remain the same for each child, and T' will maintain $\psi(m)$-determinism.
- On the other hand, if $\psi(m) \cap \boldsymbol{W} \neq \emptyset$, then we do not have any such guarantee; in fact, we cannot be sure that T' will be $\boldsymbol{Q}$-deterministic for any $\boldsymbol{Q}$, so we assign the universal set.

`INST`$(\cdot; \boldsymbol{w})$ For the instantiation operation, we have Algorithm B.2. At first glance, this seems to be very similar to the marginalization operation; it changes the scope in the same way, and the changes to the circuit can be implemented through the leaf nodes. However, the crucial difference is in the label function.

The new label of $\psi'(m) = \psi(m) \setminus \boldsymbol{W}$ is justified as follows. Suppose that we have a sum node $T \in T_m$, with children $N_1, ..., N_n$, with T marginally deterministic with respect to $\psi(m)$. After instantiation (of $\boldsymbol{W}$ with the value $\boldsymbol{w}$), the function encoded by each child N_i' satisfies $p_{N_i'}(\boldsymbol{q} \setminus \boldsymbol{W}) = p_{N_i}(\boldsymbol{w}, \boldsymbol{q} \setminus \boldsymbol{W})$, for any value $\boldsymbol{q}$ of $\psi(m)$ by definition[4].

Now, we claim that N_i' is $(\psi(m) \setminus \boldsymbol{W})$-deterministic, i.e. N_i', N_j' have distinct marginalized support $\mathrm{supp}_{\psi(m)\setminus\boldsymbol{W}}(N_i')$, $\mathrm{supp}_{\psi(m)\setminus\boldsymbol{W}}(N_j')$ for $i \neq j$. Suppose for contradiction there exists a value $\boldsymbol{q}^* \setminus \boldsymbol{W}$ of $(\psi(m) \setminus \boldsymbol{W})$ such that $p_{N_i'}(\boldsymbol{q}^* \setminus \boldsymbol{W}) > 0$ and

[4]Note that we write $\boldsymbol{q} \setminus \boldsymbol{W}$ to represent the value of $\psi(m) \setminus \boldsymbol{W}$ given by $\boldsymbol{q}$ restricted to this variable set.

Algorithm B.1: MARG($\mathcal{C}, \boldsymbol{W}$)

```
Input: Input circuit $\mathcal{C} = (v = (M, E, \phi), \psi, \tau)$; set of variables to be
       marginalized $\boldsymbol{W}$
Result: Output circuit $\mathcal{C}' = (v', \psi', \tau')$
1  $m \leftarrow$ root($v$);
2  $m' \leftarrow$ newnode();
3  if $m$ is leaf then  // Update vtree structure and parameter function
    (leaf)
4      $v' \leftarrow$ createvtree($m'$);           // create vtree with single node
5      $\tau'(m') \leftarrow$ (MARG($L; \boldsymbol{W}$) for $L \in \boldsymbol{T}_m, \omega_m$);      // marginalize leaf PC
       nodes
6  else  // Update vtree structure and parameter function (non-leaf)
7      $m_l, m_r \leftarrow$ children($m$);
8      $v'_l, \psi'_l, \tau'_l \leftarrow$ MARG($(v_{m_l}, \psi, \tau), \boldsymbol{W}$);
9      $v'_r, \psi'_r, \tau'_r \leftarrow$ MARG($(v_{m_r}, \psi, \tau), \boldsymbol{W}$);
10     $v', \psi', \tau' \leftarrow v'_l \cup v'_r, \psi'_l \cup \psi'_r, \tau'_l \cup \tau'_r$;   // combine the vtrees/labelling
       fn/param fn
11     $v' \leftarrow$ addnode($v'; m'$); $v' \leftarrow$ addchildren($v'; m'$, root($v'_l$), root($v'_r$));
12     $\tau'(m') \leftarrow \tau(m)$;
13 $\phi'(m') \leftarrow \phi(m) \setminus \boldsymbol{W}$;                    // Update scope function
14 if $\psi(m) \cap \boldsymbol{W} = \emptyset$ then                  // Update labelling function
15     $\psi'(m') \leftarrow \psi(m)$;
16 else
17     $\psi'(m') \leftarrow U$;
18 Return $(v', \psi', \tau')$
```

$p_{N'_j}(\boldsymbol{q}^* \setminus \boldsymbol{W}) > 0$. Then we have

$$p_{N'_i}(\boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \text{ and } p_{N'_j}(\boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \tag{B.37}$$

$$p_{N_i}(\boldsymbol{w}, \boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \text{ and } p_{N_j}(\boldsymbol{w}, \boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \tag{B.38}$$

$$\sum_{\boldsymbol{W} \setminus \boldsymbol{Q}} p_{N_i}(\boldsymbol{W} \setminus \boldsymbol{Q}, \boldsymbol{w} \cap \boldsymbol{Q}, \boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \text{ and } \sum_{\boldsymbol{W} \setminus \boldsymbol{Q}} p_{N_j}(\boldsymbol{W} \setminus \boldsymbol{Q}, \boldsymbol{w} \cap \boldsymbol{Q}, \boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \tag{B.39}$$

$$p_{N_i}(\boldsymbol{w} \cap \boldsymbol{Q}, \boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \text{ and } p_{N_j}(\boldsymbol{w} \cap \boldsymbol{Q}, \boldsymbol{q}^* \setminus \boldsymbol{W}) > 0 \tag{B.40}$$

The second line follows by definition of the instantiation algorithm, the third line is a sum of non-negative terms including a positive term from the previous line (when $\boldsymbol{W} \setminus \boldsymbol{Q} = \boldsymbol{w} \setminus \boldsymbol{Q}$), and the fourth line rewrites the sum. Now we have a value

Algorithm B.2: INST($\mathcal{C}, \boldsymbol{w}$)

```
Input: Input circuit $\mathcal{C} = (v = (M, E, \phi), \psi, \tau)$; instantiation $\boldsymbol{w}$ of some subset
       of variables $\boldsymbol{W}$
Result: Output circuit $\mathcal{C}' = (v', \psi', \tau')$
1  $m \leftarrow$ root($v$);
2  $m' \leftarrow$ newnode();
3  if $m$ is leaf then  // Update vtree structure and parameter function
     (leaf)
4      $v' \leftarrow$ createvtree($m'$);            // create vtree with single node
5      $\tau'(m') \leftarrow$ (INST($L; \boldsymbol{w}$) for $L \in \boldsymbol{T}_m, \omega_m$);       // instantiate leaf PC
       nodes
6  else  // Update vtree structure and parameter function (non-leaf)
7      $m_l, m_r \leftarrow$ children($m$);
8      $v_l', \psi_l', \tau_l' \leftarrow$ INST($(v_{m_l}, \psi, \tau), \boldsymbol{w}$);
9      $v_r', \psi_r', \tau_r' \leftarrow$ INST($(v_{m_r}, \psi, \tau), \boldsymbol{w}$);
10     $v', \psi', \tau' \leftarrow v_l' \cup v_r', \psi_l' \cup \psi_r', \tau_l' \cup \tau_r'$;   // combine the vtrees/labelling
       fn/param fn
11     $v' \leftarrow$ addnode($v'; m'$); $v' \leftarrow$ addchildren($v'; m'$, root($v_l'$), root($v_r'$));
12     $\tau'(m') \leftarrow \tau(m)$;
13 $\phi'(m') \leftarrow \phi(m) \setminus \boldsymbol{W}$;                            // Update vtree scope function
14 $\psi'(m') \leftarrow \psi(m) \setminus \boldsymbol{W}$;                             // Update labelling function
15 Return $(v', \psi', \tau')$
```

$\boldsymbol{q} := (\boldsymbol{w} \cap \boldsymbol{Q}, \boldsymbol{q}^* \setminus \boldsymbol{W})$ of $\psi(m)$, such that $p_{N_i}(\boldsymbol{q}) > 0$ and $p_{N_j}(\boldsymbol{q}) > 0$, which is a contradiction as T is $\psi(m)$-deterministic.

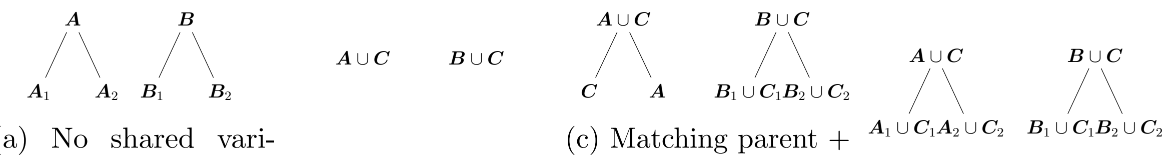
PROD($\cdot, \cdot$) Now, we consider the product of two circuits exactly respecting *compatible* vtrees.

Definition B.4 (Vtree Compatibility). *Let $v^{(1)} = (M^{(1)}, E^{(1)}, \phi^{(1)})$ and $v^{(2)} = (M^{(2)}, E^{(2)}, \phi^{(2)})$ be two vtrees, with root nodes $m^{(1)}, m^{(2)}$ respectively. Define $\boldsymbol{C} := \phi^{(1)}(m^{(1)}) \cup \phi^{(2)}(m^{(2)})$ to be the common variables. Then we say that $v^{(1)}, v^{(2)}$ are compatible if any of the following hold:*

1. *There are no common variables, $\boldsymbol{C} = \emptyset$;*

2. *Both of $m^{(1)}, m^{(2)}$ are leaf vtree nodes;*

3. *One of the root nodes has the same restricted scope on $\boldsymbol{C}$ as one of the children of the other root node,* ***and*** *the vtrees rooted at these nodes are compatible. For*


$\boldsymbol{A}$
$\boldsymbol{A}_1$
$\boldsymbol{A}_2$
$\boldsymbol{B}$
$\boldsymbol{B}_1$
$\boldsymbol{B}_2$
$\boldsymbol{A} \cup \boldsymbol{C}$
$\boldsymbol{B} \cup \boldsymbol{C}$
$\boldsymbol{A} \cup \boldsymbol{C}$
$\boldsymbol{C}$
$\boldsymbol{A}$
$\boldsymbol{B} \cup \boldsymbol{C}$
$\boldsymbol{B}_1 \cup \boldsymbol{C}_1$
$\boldsymbol{B}_2 \cup \boldsymbol{C}_2$
$\boldsymbol{A} \cup \boldsymbol{C}$
$\boldsymbol{A}_1 \cup \boldsymbol{C}_1$
$\boldsymbol{A}_2 \cup \boldsymbol{C}_2$
$\boldsymbol{B} \cup \boldsymbol{C}$
$\boldsymbol{B}_1 \cup \boldsymbol{C}_1$
$\boldsymbol{B}_2 \cup \boldsymbol{C}_2$

(a) No shared variables (b) Leaf vtrees (c) Matching parent + child (d) Matching children

Figure B.1: Examples of (possibly) compatible vtrees, where $\boldsymbol{A} \cap \boldsymbol{B} = \emptyset$, and $\boldsymbol{C}$ are the shared variables

example, $\phi^{(1)}_{\boldsymbol{C}}(m^{(2)}) = \phi^{(1)}_{\boldsymbol{C}}(m^{(1)}_r)$, and $v^{(2)}_{m^{(2)}}$ and $v^{(1)}_{m^{(1)}_r}$ are compatible.

4. *The children of the root nodes have matching restricted scopes, **and** are compatible. For example, $\phi^{(1)}_{\boldsymbol{C}}(m^{(1)}_l) = \phi^{(2)}_{\boldsymbol{C}}(m^{(2)}_l)$ and $v^{(1)}_{m^{(1)}_l}, v^{(2)}_{m^{(2)}_l}$ are compatible, and $\phi^{(1)}_{\boldsymbol{C}}(m^{(1)}_r) = \phi^{(2)}_{\boldsymbol{C}}(m^{(2)}_r)$ and $v^{(1)}_{m^{(1)}_r}, v^{(2)}_{m^{(2)}_r}$ are compatible.*

This recursive definition allows for products of circuits not necessarily respecting the same vtree, but merely vtrees which "essentially have the same structure" over the shared variables. Intuitively, there are four cases that allow us to maintain (structured) decomposability in the output circuit, illustrated in Figure B.1. The first two are base cases where the product is directly tractable: namely, when the root vtree nodes have disjoint scopes, or when they are both leaves. Note, in particular, that the product of a leaf region and a non-leaf region that have overlapping variables is considered intractable here (unless condition 3. holds). The last two are cases where we can recursively call the product algorithm on the children of the root vtree nodes. Each of the four cases above correspond to a slightly different algorithm for computing the product of the corresponding PC sum nodes, which we will explain next.[5]

The product algorithm, depicted in Algorithm B.3, (recursively) constructs a circuit $\mathcal{C} = (v', \psi', \tau')$ that is the product of the input circuits $\mathcal{C}^{(1)} = (v^{(1)}, \psi^{(1)}, \tau^{(1)})$ and $\mathcal{C}^{(2)} = (v^{(2)}, \psi^{(2)}, \tau^{(2)})$ respectively. In particular, at each recursive step, it computes the root node m' of the new vtree, its label $\psi'(m')$, and the parameter function value $\tau'(m') = (\boldsymbol{T}_{m'}, \omega_{m'})$ of that node[6]. We consider each of the four compatibility cases

[5]The notion of compatibility between vtrees is somewhat similar to the notion of compatibility between circuits [185], but acts at the level of groups of circuit nodes with the same scope (i.e. a vtree node) rather than individual circuit nodes.

[6]Consider the root nodes of the input vtrees $m^{(1)}, m^{(2)}$, and their parameter function values $\tau^{(1)}(m^{(1)}) = (\boldsymbol{T}_{m^{(1)}}, \omega_{m^{(1)}}), \tau^{(2)}(m^{(2)}) = (\boldsymbol{T}_{m^{(2)}}, \omega_{m^{(2)}})$. In every case, $\boldsymbol{T}_{m'}$ will contain one node $T_{m',i^{(1)},i^{(2)}}$ corresponding to every *pair of nodes* $T_{m^{(1)},i^{(1)}} \in \boldsymbol{T}_{m^{(1)}}$, $T_{m^{(2)},i^{(2)}} \in \boldsymbol{T}_{m^{(2)}}$; we thus notate it

separately:

1. Firstly, if there are no common variables, i.e. $\boldsymbol{C} = \emptyset$, then we can simply introduce product nodes for each pair of sum nodes, while maintaining decomposability, as in Figure B.2a.

    - *Vtree node:* We create a vtree node m' with $(m'_l = m^{(1)}, m'_r = m^{(2)})$ as children.

    - *Parameter function:* The parameter function value for the new node $\tau'(m') = (\boldsymbol{T}_{m'}, \omega_{m'})$ is given as follows. For each pair of sum nodes $T_{m^{(1)},j} \in \boldsymbol{T}_{m^{(1)}}, T_{m^{(2)},k} \in \boldsymbol{T}_{m^{(2)}}$, we create a sum node $T_{m',i^{(1)}i^{(2)}}$, representing the product of $T_{m^{(1)},j}, T_{m^{(2)},k}$. To achieve this, the weights $\omega_{m',i^{(1)}i^{(2)}jk}$ are defined to be 1 if $i^{(1)} = j$ and $i^{(2)} = k$, and 0 otherwise. Note that j, k only have a single index as the children m'_l, m'_r correspond to the input circuits, which only have a single dimension.

    - *Md-vtree labelling:* The label is set to be $\psi'(m') := \emptyset$; this is since all sum nodes only effectively have a single child, so they are trivially $\boldsymbol{Q}$-deterministic for any $\boldsymbol{Q}$. An example can be seen in Figure B.2a.

2. Secondly, if both $m^{(1)}, m^{(2)}$ are leaves, then the PC nodes corresponding to these vtree nodes are also either leaves, or simple mixtures (sum nodes) of leaves. To compute the product of two sum nodes, we expand all combinations of the children of the sum nodes, as shown in Figure B.2c.

    - *Vtree node:* We create a leaf vtree node m'.

    - *Parameter function:* The parameter function value for the new node $\tau'(m') = (\boldsymbol{T}_{m'}, \omega_{m'})$ is given as follows. For each pair of nodes $N_{m^{(1)},j} \in \boldsymbol{T}_{m^{(1)}}, N_{m^{(2)},k} \in \boldsymbol{T}_{m^{(2)}}$, we create a sum/leaf node $N_{m',i^{(1)}i^{(2)}}$, representing the product of $N_{m^{(1)},j}, N_{m^{(2)},k}$. The weights are defined as $\omega_{m',i^{(1)},i^{(2)},j^{(1)},j^{(2)}} := \omega_{m',i^{(1)},j^{(1)}}\omega_{m',i^{(2)},j^{(2)}}$.

with two dimensions. Correspondingly, in general, $\omega_{m'}$ will contain one weight $\omega_{m',i^{(1)}i^{(2)}j^{(1)}j^{(2)}k^{(1)}k^{(2)}}$ for every *combination of nodes* $T_{m',i^{(1)},i^{(2)}} \in \boldsymbol{T}_{m'}, T_{m'_l,j^{(1)},j^{(2)}} \in \boldsymbol{T}_{m'_l}, T_{m'_r,k^{(1)},k^{(2)}} \in \boldsymbol{T}_{m'_r}$, where m'_l and m'_r are the left and right children of m' in the *new* md-vtree.

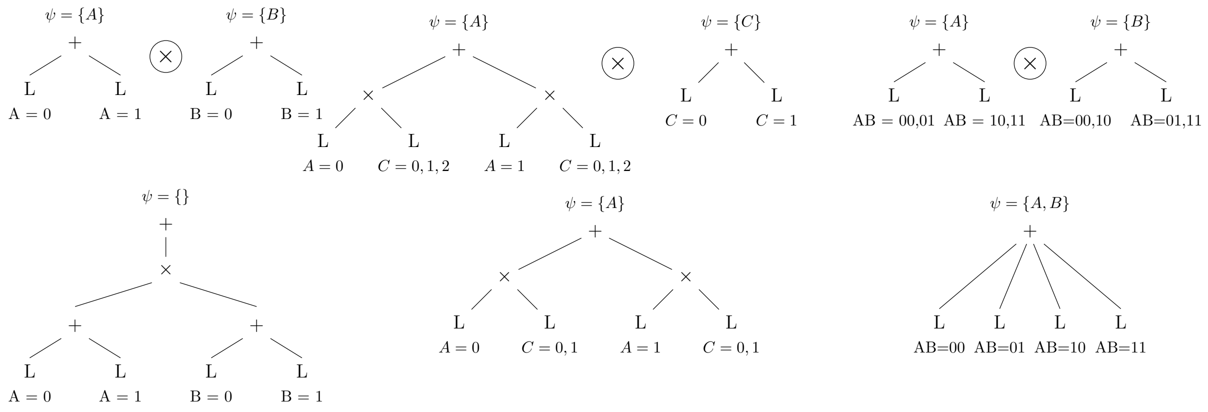
- *Md-vtree labelling:* The label is set to be $\psi(m') := \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$. This is best seen with an example; in Figure B.2c, we see an example of the product of two sum nodes with leaf node children, where one node is A-deterministic and the other is B-deterministic. The resulting sum node in the output circuit has children corresponding to (the product of) each combination of the children of the original two sum nodes; as a result, each child of the output sum node corresponds to a different value of (A, B), and so the output sum node is $\{A, B\}$-deterministic.

3. Thirdly, if one of the nodes has the same restricted scope as a child of the other node, e.g. $\phi^{(1)}_{\boldsymbol{C}}(m^{(2)}) = \phi^{(1)}_{\boldsymbol{C}}(m^{(1)}_r)$, we "defer" the product as shown in Figure B.2b.

   - *Vtree node:* We create a vtree node m' with $(m'_l = m^{(1)}_l, m'_r = \texttt{PROD}(v^{(1)}_{m^{(1)}_r}, v^{(2)}_{m^{(2)}}))$ as children.
   - *Parameter function:* The parameter function value for the new node $\tau'(m') = (\boldsymbol{T}_{m'}, \omega_{m'})$ is given as follows. For every pair of sum nodes $T_{m^{(1)},i^{(1)}} \in \boldsymbol{T}_{m^{(1)}}$, $T_{m^{(2)},i^{(2)}} \in \boldsymbol{T}_{m^{(2)}}$, we create a sum node $T_{m',i^{(1)}i^{(2)}}$. The weights are defined as $\omega_{m',i^{(1)}i^{(2)}j^{(1)}k^{(1)}k^{(2)}} := \omega_{m^{(1)},i^{(1)}j^{(1)}k^{(1)}} \mathbb{1}_{i^{(2)}=k^{(2)}}$. Note that the index j only has a single index as the left child $m'_l = m^{(1)}_l$, and so the sum nodes $\boldsymbol{T}_{m'_l}$ are copies of the sum nodes from $\boldsymbol{T}_{m^{(1)}_l}$.
   - *Md-vtree labelling:* The label is set to be $\psi'(m') := \psi^{(1)}(m^{(1)})$, as the marginalized support of the children of the output sum nodes is a subset of the marginalized support of the corresponding sum node from $m^{(1)}$, as can be seen in Figure B.2b.

4. Finally, in any other case, the children of $m^{(1)}, m^{(2)}$ have matching restricted scopes, e.g. $\phi^{(1)}_{\boldsymbol{C}_{12}}(m^{(1)}_l) = \phi^{(2)}_{\boldsymbol{C}_{12}}(m^{(2)}_l)$ and $\phi^{(1)}_{\boldsymbol{C}}(m^{(1)}_r) = \phi^{(2)}_{\boldsymbol{C}}(m^{(2)}_r)$, we expand all combinations:

   - *Vtree node:* We create a vtree node m' with $(m'_l = \texttt{PROD}(m^{(1)}_l, m^{(2)}_l)$, $m'_r = \texttt{PROD}(m^{(1)}_r, m^{(2)}_r))$ as children.



(a) Direct Product (b) Deferred Product (c) Product by Expansion

Figure B.2: Examples of product of the two sum nodes on the top half, with the result shown in the bottom half. The root sum node is labelled with the corresponding vtree node label, while the leaves are labelled with their support.

- *Parameter function:* The parameter function value for the new node $\tau'(m') = (\boldsymbol{T}_{m'}, \omega_{m'})$ is given as follows. For every pair of sum nodes $T_{m^{(1)},i^{(1)}} \in \boldsymbol{T}_{m^{(1)}}$, $T_{m^{(2)},i^{(2)}} \in \boldsymbol{T}_{m^{(2)}}$, we create a sum node $T_{m',i^{(1)}i^{(2)}}$. The weights are defined as $\omega_{m',i^{(1)}i^{(2)}j^{(1)}j^{(2)}k^{(1)}k^{(2)}} := \omega_{m^{(1)},i^{(1)}j^{(1)}k^{(1)}}\omega_{m^{(1)},i^{(2)}j^{(2)}k^{(2)}}$.
- *Md-vtree labelling:* The label is set to be $\psi(m') := \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$, for similar reasons to the product of two leaf vtree nodes above.

With this, we have shown how to each recursive step of the product algorithm. Now, taking a step back, we consider the entire run of the recursive algorithm. Starting from md-vtrees $v^{(1)}, v^{(2)}$ over variables $\boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}$, with common variables $\boldsymbol{C}_{\text{global}} := \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$, in each recursive call, we reduce the common variables, until either we reach two vtree nodes that do not have common variables, or we reach leaf vtree nodes. One property of the algorithm, which will be important for the proof of the MD-calculus rule below, is that at each recursive step of the product algorithm, the two input vtree nodes have the same restricted scope over $\boldsymbol{C}_{\text{global}}$.

Proposition B.1. *At each recursive step of Algorithm B.3, we have that* $\phi^{(1)}_{\boldsymbol{C}_{global}}(m^{(1)}) = \phi^{(2)}_{\boldsymbol{C}_{global}}(m^{(2)})$.

Proof. Proof is by inspection; in each recursive case (3, 4), we have that $\boldsymbol{C} = \phi^{(1)}_{\boldsymbol{C}_{\text{global}}}(m^{(1)}) = \phi^{(2)}_{\boldsymbol{C}_{\text{global}}}(m^{(2)})$, and match up the common variables among the recursive call(s). □

`POW`$(\cdot; \alpha)$ For the power operation, we have Algorithm B.4. This algorithm simply inverts all the weights/parameters of the circuit, as well as replacing the leaves with their reciprocals. Provided that the input circuit is deterministic, the output circuit faithfully represents the reciprocal of the input circuit [185]. As the transformation is simply numerical (i.e. not affecting the support of any node), the labels of all nodes remain the same.

`MAX`$(\cdot; \alpha)$ This operation returns a scalar.

`LOG`$(\cdot)$ This operation returns a circuit which respects the same vtree, but does not have (marginal) determinism [185].

B.2.2 MD-calculus and the Backward Problem

The algorithms for each of the basic operations in the above section allow us to derive a md-vtree for the output circuit, given the md-vtree that the input circuit respects. The MD-calculus in Table 5.2 (repeated for convenience in Table B.1) turns these results into a series of straightforward rules that can easily be applied to derive sufficient (but possibly not necessary) conditions for tractability of compositions of operations.

Theorem 5.5 (MD-calculus)**.** *The conditions in Table 5.2 hold.*

Proof. To state the result more formally, we claim that if the input circuit(s) respect md-vtrees(s) implying the input condition, then the result of the operation applied to the input circuit(s) will respect a md-vtree implying the output condition.

`MARG`$(\cdot; \boldsymbol{W})$ For the marginalization operation, the output md-vtree is over variables $\boldsymbol{V} \setminus \boldsymbol{W}$. Thus, let $\boldsymbol{Q}$ be any subset of $\boldsymbol{V} \setminus \boldsymbol{W}$.

- *Input Condition:* The input condition requires that the input md-vtree w implies $\boldsymbol{Q}$-determinism; that is, for every vtree node m, either $\phi_{\boldsymbol{Q}}(m) = \emptyset$, or else $\boldsymbol{Q} \supseteq \psi(m)$.

- *Algorithm:* In Algorithm B.1, every vtree node m' in the output md-vtree corresponds to a vtree node m' in the input md-vtree, such that $\phi'(m') = \phi(m) \setminus \boldsymbol{W}$, and $\psi'(m') = \psi(m)$ if $\psi(m) \cap \boldsymbol{W} = \emptyset$, or $\psi'(m') = U$ otherwise.

- *Proof for Output Condition:* For each vtree node m', if $\phi'_{\boldsymbol{Q}}(m') \neq \emptyset$, then, we have that:

$$\begin{aligned} & \phi_{\boldsymbol{Q}}(m) \setminus \boldsymbol{W} \neq \emptyset && \text{(by effect of algorithm)} \\ \Longrightarrow\ & \phi_{\boldsymbol{Q}}(m) \neq \emptyset && \text{(weaker statement)} \\ \Longrightarrow\ & \psi(m) \subseteq \boldsymbol{Q} && \text{(by input condition)} \\ \Longrightarrow\ & \psi'(m') \subseteq \boldsymbol{Q} \end{aligned}$$

  The last line follows since $\boldsymbol{Q} \cap \boldsymbol{W} = \emptyset$, so $\psi(m) \cap \boldsymbol{W} = \emptyset$, and so we are in the algorithm case where the label is "copied". Thus, we have shown that the output md-vtree implies $\boldsymbol{Q}$-determinism, as required.

`INST`$(\cdot; \boldsymbol{w})$ For the instantiation operation, the output md-vtree is over variables $\boldsymbol{V} \setminus \boldsymbol{W}$. Thus, let $\boldsymbol{Q}$ be any subset of $\boldsymbol{V} \setminus \boldsymbol{W}$.

- *Input Condition:* The input condition requires that the input md-vtree w implies $(\boldsymbol{Q} \cup \boldsymbol{W}')$-determinism for some $\boldsymbol{W}' \subseteq \boldsymbol{W}$; that is, for every vtree node m, either $\phi_{\boldsymbol{Q} \cup \boldsymbol{W}'}(m) = \emptyset$, or else $\boldsymbol{Q} \cup \boldsymbol{W}' \supseteq \psi(m)$.

- *Algorithm:* In Algorithm B.2, every vtree node m' in the output md-vtree corresponds to a vtree node m' in the input md-vtree, such that $\phi'(m') = \phi(m) \setminus \boldsymbol{W}$, and $\psi'(m') = \psi(m) \setminus \boldsymbol{W}$.

- *Proof for Output Condition:* For each vtree node m', if $\phi'_{\boldsymbol{Q}}(m') \neq \emptyset$, then, we have that:

$$\begin{aligned} & \phi_{\boldsymbol{Q}}(m) \setminus \boldsymbol{W} \neq \emptyset && \text{(by effect of algorithm)} \\ \Longrightarrow\ & \phi_{\boldsymbol{Q}}(m) \neq \emptyset && \text{(weaker statement)} \\ \Longrightarrow\ & \phi_{\boldsymbol{Q}\cup\boldsymbol{W}'}(m) \neq \emptyset && \text{(weaker statement)} \\ \Longrightarrow\ & \psi(m) \subseteq \boldsymbol{Q} \cup \boldsymbol{W}' && \text{(by input condition)} \\ \Longrightarrow\ & \psi'(m') \subseteq \boldsymbol{Q} \end{aligned}$$

Here, the last line follows since the new label $\psi'(m') = \psi(m) \setminus \boldsymbol{W}$ removes all elements of $\boldsymbol{W}$, and thus $\boldsymbol{W}'$, from $\psi(m)$. Thus, we have shown that the output md-vtree implies $\boldsymbol{Q}$-determinism, as required.

`PROD`$(\cdot,\cdot)$ For the product operation, the output md-vtree is over variables $\boldsymbol{V}^{(1)} \cup \boldsymbol{V}^{(2)}$. Thus, let $\boldsymbol{Q}$ be any subset of $\boldsymbol{V}^{(1)} \cup \boldsymbol{V}^{(2)}$.

- *Input Condition:* The input condition requires that the first input md-vtree $w^{(1)}$ implies $\boldsymbol{Q}^{(1)}$-determinism, and the second input md-vtree $w^{(2)}$ implies $\boldsymbol{Q}^{(2)}$-determinism, where *one* of the following holds:

  (a) $\boldsymbol{Q} \subseteq \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$ and $\boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)} = \boldsymbol{Q}$;

  (b) $\boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} \supseteq \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$ and $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)}$

- *Algorithm:* In Algorithm B.3, every vtree node m' in the output md-vtree corresponds to a pair $m^{(1)}, m^{(2)}$ in the input md-vtrees respectively, such that $\phi'(m') = \phi^{(1)}(m^{(1)}) \cup \phi^{(2)}(m^{(2)})$. There are four cases of the algorithm to consider, in which the label is:

  1. $\psi'(m') = \emptyset$.
  2. $\psi'(m') = \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$
  3. $\psi'(m') = \psi^{(1)}(m^{(1)})$
  4. $\psi'(m') = \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$

- *Proof for Output Condition:* We need to show that for each case 1-3 of the algorithm, and for either input condition (a), (b), that the condition for implied $\boldsymbol{Q}$-determinism holds on m'; that is, if $\phi'_{\boldsymbol{Q}}(m') \neq \emptyset$, then $\psi'(m') \subseteq \boldsymbol{Q}$. Assuming that $\phi'_{\boldsymbol{Q}}(m') \neq \emptyset$, we have that

$$\begin{aligned}
&\phi'_{\boldsymbol{Q}}(m') \neq \emptyset \\
\implies & \phi'(m') \cap \boldsymbol{Q} \neq \emptyset && \text{(by definition of restricted scope)} \\
\implies & (\phi^{(1)}(m^{(1)}) \cup \phi^{(2)}(m^{(2)})) \cap \boldsymbol{Q} \neq \emptyset && \text{(by effect of algorithm)} \\
\implies & \phi^{(1)}_{\boldsymbol{Q}}(m^{(1)}) \cup \phi^{(2)}_{\boldsymbol{Q}}(m^{(2)}) \neq \emptyset && \text{(rewriting)}
\end{aligned}$$

However, this does not in general imply that $\phi^{(1)}_{\boldsymbol{Q}^{(1)}}(m^{(1)}) \neq \emptyset$ or $\phi^{(2)}_{\boldsymbol{Q}^{(2)}}(m^{(2)}) \neq \emptyset$. Thus, we look at the special cases defined by (a) and (b), and the algorithm variations 1, 2, 3, 4.

(a1, a2, a3, a4) In case (a), we have $\boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)} = \boldsymbol{Q} \subseteq \boldsymbol{C}_{\text{global}}$.

$$\begin{aligned}
&\phi^{(1)}_{\boldsymbol{C}_{\text{global}}}(m^{(1)}) = \phi^{(2)}_{\boldsymbol{C}_{\text{global}}}(m^{(2)}) && \text{(by Proposition B.1)} \\
\implies & \phi^{(1)}_{\boldsymbol{C}_{\text{global}}}(m^{(1)}) \cap \boldsymbol{Q} = \phi^{(2)}_{\boldsymbol{C}_{\text{global}}}(m^{(2)}) \cap \boldsymbol{Q} \\
\implies & \phi^{(1)}_{\boldsymbol{C}_{\text{global}} \cap \boldsymbol{Q}}(m^{(1)}) = \phi^{(2)}_{\boldsymbol{C}_{\text{global}} \cap \boldsymbol{Q}}(m^{(2)}) \\
\implies & \phi^{(1)}_{\boldsymbol{Q}}(m^{(1)}) = \phi^{(2)}_{\boldsymbol{Q}}(m^{(2)}) && \text{(as } \boldsymbol{Q} \subseteq \boldsymbol{C}_{\text{global}}) \\
\implies & \phi^{(1)}_{\boldsymbol{Q}}(m^{(1)}) \neq \emptyset, \phi^{(2)}_{\boldsymbol{Q}}(m^{(2)}) \neq \emptyset && \text{(as } \phi^{(1)}_{\boldsymbol{Q}}(m^{(1)}) \cup \phi^{(2)}_{\boldsymbol{Q}}(m^{(2)}) \neq \emptyset) \\
\implies & \phi^{(1)}_{\boldsymbol{Q}^{(1)}}(m^{(1)}) \neq \emptyset, \phi^{(2)}_{\boldsymbol{Q}^{(2)}}(m^{(2)}) \neq \emptyset && \text{(as } \boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)} = \boldsymbol{Q})
\end{aligned}$$

Thus, we have that $\boldsymbol{Q} \supseteq \psi^{(1)}(m^{(1)})$ and $\boldsymbol{Q} \supseteq \psi^{(2)}(m^{(2)})$, and so $\boldsymbol{Q} \supseteq \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$. Finally, in each of the cases 1-4, we have $\boldsymbol{Q} \supseteq \psi'(m')$, so the output md-vtree implies $\boldsymbol{Q}$-determinism as required.

(b1) In case (b), we need to consider the cases of the algorithm separately. In case 1, $\psi'(m') \subseteq \boldsymbol{Q}$ holds trivially as $\psi'(m') = \emptyset$, so we are done.

(b2, b3, b4) We have that $\boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} \supseteq \boldsymbol{C}_{\text{global}}$ and $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)}$. The key observation is that, as we are not in case 1 of the Algorithm, $\boldsymbol{C} =$

$\phi^{(1)}(m^{(1)}) \cap \phi^{(2)}(m^{(2)})$ must be non-empty.

$$\begin{aligned}
&\boldsymbol{C} \neq \emptyset \\
\implies &\phi^{(1)}_{\boldsymbol{C}}(m^{(1)}) \neq \emptyset, \phi^{(2)}_{\boldsymbol{C}}(m^{(2)}) \neq \emptyset && \text{(by definition of } \boldsymbol{C}\text{)} \\
\implies &\phi^{(1)}_{\boldsymbol{Q}^{(1)}}(m^{(1)}) \neq \emptyset, \phi^{(2)}_{\boldsymbol{Q}^{(1)}}(m^{(2)}) \neq \emptyset && \text{(as } \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} \supseteq \boldsymbol{C}_{\text{global}} \supseteq \boldsymbol{C}\text{)}
\end{aligned}$$

Thus, we have that $\boldsymbol{Q}^{(1)} \supseteq \psi^{(1)}(m^{(1)})$ and $\boldsymbol{Q}^{(2)} \supseteq \psi^{(2)}(m^{(2)})$, and so $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)} \supseteq \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$. In each of the cases 2-4, we have $\boldsymbol{Q} \supseteq \psi'(m')$, so the output md-vtree implies $\boldsymbol{Q}$-determinism as required.

We previously showed in Proposition 5.1 that the conditioning algorithm on circuits (Algorithm 5.1) satisfies the property that a $\boldsymbol{Q}$-deterministic input circuit will result in a $\boldsymbol{Q}$-deterministic output circuit.

`POW` Since the power algorithm does not change the marginal determinisms of any sum node (retains the same labelling function), it follows that a $\boldsymbol{Q}$-deterministic input circuit will result in a $\boldsymbol{Q}$-deterministic output circuit.

`MAX` This operation returns a scalar.

`LOG` This operation does not have any marginal determinism conditions.

□

Algorithm B.3: `PROD`$(\mathcal{C}^{(1)}, \mathcal{C}^{(2)})$

```
   Input: Input circuits $\mathcal{C}^{(1)} = (v^{(1)} = (M^{(1)}, E^{(1)}, \phi^{(1)}), \psi^{(1)}, \tau^{(1)}), \mathcal{C}^{(2)} = (v^{(2)} = (M^{(2)}, E^{(2)}, \phi^{(2)}), \psi^{(2)}, \tau^{(2)})$
   Result: Output circuit $\mathcal{C}' = (v', \psi', \tau')$
1  $m^{(1)}, m^{(2)} \leftarrow$ root($v^{(1)}$), root($v^{(2)}$);
2  $(m_l^{(1)}, m_r^{(1)}), (m_l^{(2)}, m_r^{(2)}) \leftarrow$ children($m^{(1)}$), children($m^{(2)}$);      // null if $m^{(1)}$/$m^{(2)}$ are leaf
3  $m' \leftarrow$ newvtreenode();
4  $\boldsymbol{C} \leftarrow \phi^{(1)}(m^{(1)}) \cap \phi^{(2)}(m^{(2)})$;
5  if $\boldsymbol{C} = \emptyset$ then
6      $v_l', \psi_l', \tau_l' \leftarrow v^{(1)}, \psi^{(1)}, \tau^{(1)}$;
7      $v_r', \psi_r', \tau_r' \leftarrow v^{(2)}, \psi^{(2)}, \tau^{(2)}$;
8      $v', \psi', \tau' \leftarrow v_l' \cup v_r', \psi_l' \cup \psi_r', \tau_l' \cup \tau_r'$;   // combine the vtrees/labelling fn/param fn
9      $\psi'(m') \leftarrow \emptyset$;                                   // Update label function
10     $\boldsymbol{T}_{m'} \leftarrow \big(T_{m',i^{(1)}i^{(2)}}$ for $i^{(1)} = 1, ..., |\boldsymbol{T}_{m^{(1)}}|, i^{(2)} = 1, ...|\boldsymbol{T}_{m^{(2)}}|\big)$;
11     $\omega_{m',i^{(1)}i^{(2)}jk} \leftarrow \mathbb{1}_{i^{(1)}=j, i^{(2)}=k}$;
12 else if $m^{(1)}$ and $m^{(2)}$ are leaves then
13     $v', \psi', \tau' \leftarrow$ createemptyvtree();
14     $\psi'(m') = \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$;
15     $\boldsymbol{T}_{m'} \leftarrow \big(T_{m',i^{(1)}i^{(2)}}$ for $i^{(1)} = 1, ..., |\boldsymbol{T}_{m^{(1)}}|, i^{(2)} = 1, ...|\boldsymbol{T}_{m^{(2)}}|\big)$;
16     $\omega_{m',i^{(1)}i^{(2)}j^{(1)}j^{(2)}k^{(1)}k^{(2)}} \leftarrow \omega_{m^{(1)},i^{(1)}j^{(1)}k^{(1)}} \omega_{m^{(1)},i^{(2)}j^{(2)}k^{(2)}}$;
17 else if $\phi_{\boldsymbol{C}}^{(1)}(m^{(2)}) = \phi_{\boldsymbol{C}}^{(1)}(m_r^{(1)})$ then
18     $v_l', \psi_l', \tau_l' \leftarrow v_{m_l^{(1)}}^{(1)}, \psi^{(1)}, \tau^{(1)}$;
19     $v_r', \psi_r', \tau_r' \leftarrow$ PROD($m_r^{(1)}, m^{(2)}$);
20     $v', \psi', \tau' \leftarrow v_l' \cup v_r', \psi_l' \cup \psi_r', \tau_l' \cup \tau_r'$;   // combine the vtrees/labelling fn/param fn
21     $\psi'(m') = \psi^{(1)}(m^{(1)})$;                                   // Update label function
22     $\boldsymbol{T}_{m'} \leftarrow \big(T_{m',i^{(1)}i^{(2)}}$ for $i^{(1)} = 1, ..., |\boldsymbol{T}_{m^{(1)}}|, i^{(2)} = 1, ...|\boldsymbol{T}_{m^{(2)}}|\big)$;
23     $\omega_{m',i^{(1)}i^{(2)}j^{(1)}k^{(1)}k^{(2)}} \leftarrow \theta_{m^{(1)},i^{(1)}j^{(1)}k^{(1)}} \mathbb{1}_{i^{(2)}=k^{(2)}}$;
24 else if $\phi_{\boldsymbol{C}}^{(1)}(m_l^{(1)}) = \phi_{\boldsymbol{C}}^{(2)}(m_l^{(2)})$ and $\phi_{\boldsymbol{C}}^{(1)}(m_r^{(2)}) = \phi_{\boldsymbol{C}}^{(1)}(m_r^{(2)})$ then
25     $v_l', \psi_l', \tau_l' \leftarrow$ PROD($m_l^{(1)}, m_l^{(2)}$);
26     $v_r', \psi_r', \tau_r' \leftarrow$ PROD($m_r^{(1)}, m_r^{(2)}$);
27     $v', \psi', \tau' \leftarrow v_l' \cup v_r', \psi_l' \cup \psi_r', \tau_l' \cup \tau_r'$;   // combine the vtrees/labelling fn/param fn
28     $\psi'(m') = \psi^{(1)}(m^{(1)}) \cup \psi^{(2)}(m^{(2)})$;
29     $\boldsymbol{T}_{m'} \leftarrow \big(T_{m',i^{(1)}i^{(2)}}$ for $i^{(1)} = 1, ..., |\boldsymbol{T}_{m^{(1)}}|, i^{(2)} = 1, ...|\boldsymbol{T}_{m^{(2)}}|\big)$;
30     $\omega_{m',i^{(1)}i^{(2)}j^{(1)}j^{(2)}k^{(1)}k^{(2)}} \leftarrow \omega_{m^{(1)},i^{(1)}j^{(1)}k^{(1)}} \omega_{m^{(1)},i^{(2)}j^{(2)}k^{(2)}}$;
31 else
32     Return fail (not compatible)
33 $\phi'(m') \leftarrow \phi^{(1)}(m^{(1)}) \cup \phi^{(2)}(m^{(2)})$;                 // Update scope function
34 $\tau'(m') \leftarrow (\boldsymbol{T}_{m'}, \omega_{m'})$                          // Update parameter function
35 $v' \leftarrow$ addnode($v'; m'$);
36 $v' \leftarrow$ addchildren($v'; m'$, root($v_l'$), root($v_r'$));
37 Return $(v', \psi', \tau')$
```

Algorithm B.4: `POW`$(\mathcal{C}, \alpha)$

```
Input: Input circuit $\mathcal{C} = (v = (M, E, \phi), \psi, \tau)$; power $\alpha$
Result: Output circuit $\mathcal{C}' = (v', \psi', \tau')$
1  $m \leftarrow$ root$(v)$;
2  $m' \leftarrow$ newnode();
3  if $m$ is leaf then  // Update vtree structure and parameter function (leaf)
4      $v' \leftarrow$ createvtree$(m')$;            // create vtree with single node
5      $\tau'(m') \leftarrow$ (POW$(L;\alpha)$ for $L \in \boldsymbol{T}_m, \omega_m)$;      // apply power to leaf PC nodes
6  else  // Update vtree structure and parameter function (non-leaf)
7      $m_l, m_r \leftarrow$ children$(m)$;
8      $v'_l, \psi'_l, \tau'_l \leftarrow$ POW$((v_{m_l}, \psi, \tau), \alpha)$;
9      $v'_r, \psi'_r, \tau'_r \leftarrow$ POW$((v_{m_r}, \psi, \tau), \alpha)$;
10     $v', \psi', \tau' \leftarrow v'_l \cup v'_r, \psi'_l \cup \psi'_r, \tau'_l \cup \tau'_r$;   // combine the vtrees/labelling fn/param fn
11     $v' \leftarrow$ addnode$(v'; m')$; $v' \leftarrow$ addchildren$(v'; m', $root$(v'_l), $root$(v'_r))$;
12     $\tau'(m') \leftarrow \tau(m)$;
13 $\phi'(m') \leftarrow \phi(m)$;                          // Update scope function
14 $\psi'(m') \leftarrow \psi(m)$;                        // Update labelling function
15 Return $(v', \psi', \tau')$
```

Operation	Input Condition	Output
`MARG`$(\mathcal{C}; \boldsymbol{W})$	$\boldsymbol{Q}$-det	$\boldsymbol{Q}$-det
`INST`$(\mathcal{C}; \boldsymbol{w})$	$\exists \boldsymbol{W}' \subseteq \boldsymbol{W} : (\boldsymbol{Q} \cup \boldsymbol{W}')$-det	$\boldsymbol{Q}$-det
`PROD`$(\mathcal{C}^{(1)}, \mathcal{C}^{(2)})$	$\exists \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} : \boldsymbol{Q}^{(1)}$-det, $\boldsymbol{Q}^{(2)}$-det, and: • Either (a) $\boldsymbol{Q} \subseteq \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$ and $\boldsymbol{Q}^{(1)} = \boldsymbol{Q}^{(2)} = \boldsymbol{Q}$; • Or (b) $\boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)} \supseteq \boldsymbol{V}^{(1)} \cap \boldsymbol{V}^{(2)}$ and $\boldsymbol{Q} = \boldsymbol{Q}^{(1)} \cup \boldsymbol{Q}^{(2)}$	$\boldsymbol{Q}$-det
`COND`$(\mathcal{C}; \boldsymbol{W})$	$\boldsymbol{Q}$-det	$\boldsymbol{Q}$-det
`POW`$(\mathcal{C}; \alpha)$	$\boldsymbol{Q}$-det	$\boldsymbol{Q}$-det
`MAX`$(\mathcal{C})$	N/A	N/A
`LOG`$(\mathcal{C})$	-	-

Table B.1: MD-calculus: sufficient input-output conditions for each basic operation

Appendix C

Tractable Causal Reasoning with Structural Uncertainty

C.1 Proofs

Proposition 6.1 (Hierarchical Conditional Independences). *Let $p(\sigma, G) \propto p_G(G)\mathbb{1}_{G\models\sigma}$ be an order-modular distribution. Suppose that S_1, S_2 are any disjoint subsets of the variables $\{1, ..., d\}$, and let (S_{21}, S_{22}) be a partition of S_2. Then the following CI holds:*

$$\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}|\sigma_{S_2} = (\sigma_{S_{21}}, \sigma_{S_{22}})) \propto \tilde{p}_{S_1,S_{21}}(\sigma_{S_{21}}, G_{S_{21}})\tilde{p}_{S_1\cup S_{21},S_{22}}(\sigma_{S_{22}}, G_{S_{22}})$$

Proof. By definition, we have that $\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}) = \prod_{i\in S_2} p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1\cup\sigma^{<i}_{S_2}}$. Conditioning on the event $\sigma_{S_2} = (\sigma_{S_{21}}, \sigma_{S_{22}})$, we have that:

$$\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}|\sigma_{S_2} = (\sigma_{S_{21}}, \sigma_{S_{22}})) \propto \prod_{i\in S_2} p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1\cup\sigma^{<i}_{(S_{21},S_{22})}} \tag{C.1}$$

$$= \prod_{i\in S_{21}} p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1\cup\sigma^{<i}_{S_{21}}} \prod_{i\in S_{22}} p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1\cup S_{21}\cup\sigma^{<i}_{S_{22}}} \tag{C.2}$$

$$= \tilde{p}_{S_1,S_{21}}(\sigma_{S_{21}}, G_{S_{21}})\tilde{p}_{S_1\cup S_{21},S_{22}}(\sigma_{S_{22}}, G_{S_{22}}) \tag{C.3}$$

as required. The second line follows due to conditioning on the event that S_{21} comes before S_{22} in the ordering, and the final line is again by definition of $\tilde{p}$. □

Proposition 6.2 (Consistency of Graph and Order). *Let $\mathcal{C}$ be an OrderSPN. Then, for all pairs (σ, G) in the support of $\mathcal{C}$ (i.e. $p_{\mathcal{C}}(\sigma, G) > 0$), it holds that $G \models \sigma$.*

Proof. Recall that a complete subcircuit $\mathcal{C}_{sub} = (\mathcal{G}_{sub}, \boldsymbol{\omega}_{sub}, \boldsymbol{g}_{sub})$ (Definition 4.2) is obtained by traversing the circuit top-down and i) selecting one child of every sum-node;

ii) selecting all children of every product-node; $\mathcal{C}_{sub}$ is itself an OrderSPN expressing a distribution over (σ, G). The key point is that the order is determined in any complete subcircuit. At the leaf nodes, the orders $\sigma_{\{i\}}$ over singletons are trivially deterministic. At the product nodes in the subcircuit, the order is determined by the order specified by the first (left) and second (right) child. That is, for a product node P in the subcircuit associated with $\rho(P) = (S_1, S_{21}, S_{22})$, if the left child specifies an order $\sigma_{S_{21}}$ and the right child an order $\sigma_{S_{22}}$, then the order for P is determined as $(\sigma_{S_{21}}, \sigma_{S_{22}})$. Finally, the sum nodes in the subcircuit only have one child, so the order is determined from its child. Let the uniquely determined order for subcircuit C be denoted σ^C.

Now, consider any path from the root node to a leaf node in the subcircuit. Label the sum nodes (and leaf node) reached T_i for $i = 1, ..., m$ (for some m), associated with $\rho(T_i)(S_{1,i}, S_{2,i})$ respectively. We will now show, by induction, that for each sum node T_i, it is the case that all variables in $S_{1,i}$ come before $S_{2,i}$ in the ordering σ^C.

- The root R is associated with $\rho(R) = (S_{1,1}, S_{2,1}) = (\emptyset, \{1, ...d\})$, so the condition is trivially satisfied.

- Now, given node T_i with $i < m$, by definition T_i has a product node child P_i such that T_{i+1} is either the first or second child of P_i. Let P_i be associated with $\rho(P_i) = (S_{1,i}, S_{21,i}, S_{22,i})$. Then, (i) if T_{i+1} is the first child of P_i, then $(S_{1,i+1}, S_{2,i+1}) = (S_{1,i}, S_{21,i})$, while (ii) if T_{i+1} is the second child of P_i, then $(S_{1,i+1}, S_{2,i+1}) = (S_{1,i} \cup S_{21,i}, S_{22,i})$. Now, σ_C has the property that all nodes in $S_{21,i}$ come before those in $S_{22,i}$. Given the inductive hypothesis that $S_{1,i}$ comes before $S_{2,i}$ in the ordering, in both cases (i) and (ii) we have that all nodes in $S_{1,i+1}$ come before nodes in $S_{2,i+1}$ in the ordering.

This means that, at any leaf node L associated with some $\rho(L) = (S_1, \{i\})$, it will be the case that S_1 comes before i in σ^C. Since the leaf distribution only has support over graphs with $G_i \subseteq S_1$, it follows that all graphs G in the support satisfy $G \models \sigma^C$.

The overall distribution of the OrderSPN is given by a (weighted) sum over all complete subcircuits, so the result follows.

$\square$

Proposition 6.3 (Properties of OrderSPNs). *Any OrderSPN is smooth and decomposable, and regular OrderSPNs are additionally deterministic.*

Proof. Given any sum node T in the OrderSPN, completeness follows since the i^{th} product node has scope $(\sigma_{S_{21,i}\cup S_{22,i}}, G_{S_{21,i}\cup S_{22,i}}) = (\sigma_{S_2}, G_{S_2})$, as $S_{21,i}, S_{22,i}$ partitions S_2 by definition. Decomposability follows immediately from the scopes of the product nodes P and their children, where the variables $(\sigma_{S_{21}\cup S_{22}}, G_{S_{21}\cup S_{22}})$ are split into sum (or leaf) nodes with scope $(\sigma_{S_{21}}, G_{S_{21}})$ and $(\sigma_{S_{22}}, G_{S_{22}})$, where S_{21}, S_{22} are disjoint. Determinism holds for regular OrderSPNs since every sum node has children which split the order into different partitions, so that the children have distinct support over orders (in fact, the choice of child at each sum node can be viewed as determining the order). □

Proposition 6.4 (Compactness of OrderSPNs). *Given a regular OrderSPN $\mathcal{C}$ over $d = 2^l$ variables, with l sum (and product) layers and expansion factors $(K_0, ...K_{l-1})$ as above, then we have that:*

- *The size (number of edges) of $\mathcal{C}$ is given by: $\sum_{i=1}^{l}(2^i + 2^{i-1})\prod_{j<i} K_j$*
- *The size (number of orders) of the support of $\mathcal{C}$ is given by: $\prod_{i=0}^{l-1} K_i^{2^i}$*

Proof. Let $\boldsymbol{T}_i$, $\boldsymbol{P}_i$ be the i^{th} sum and product layers of the OrderSPN, with $|\boldsymbol{T}_i|, |\boldsymbol{P}_i|$ nodes respectively, for $i = 0, ..., l-1$. We will also write $\boldsymbol{T}_l$ to denote the leaf layer following all of the other layers. Then, by definition, the nodes in the l^{th} sum layer each have K_j children. Thus, $|\boldsymbol{P}_i| = K_i|\boldsymbol{T}_i|$. Each product node has two children, so we have the relationship $|\boldsymbol{T}_{l+1}| = 2|\boldsymbol{P}_l|$. Since the first sum layer $\boldsymbol{T}_0$ consists of just a single root node, $|\boldsymbol{T}_0| = 1$, and it can be easily checked by iteration that $|\boldsymbol{T}_i| = 2^i \prod_{j<i} K_j$ and $|\boldsymbol{P}_i| = 2^i \prod_{j<i+1} K_j$. Thus the total number of nodes is given by:

$$\begin{aligned}\sum_{i=0}^{l}|\boldsymbol{T}_i| + \sum_{i=0}^{l-1}|\boldsymbol{P}_i| &= \sum_{i=0}^{l} 2^i \prod_{j<i} K_j + \sum_{i=0}^{l-1} 2^i \prod_{j<i+1} K_j \\ &= 1 + \sum_{i=1}^{l}(2^i + 2^{i-1}) \prod_{j<i} K_j\end{aligned}$$

Note that the structure of an OrderSPN takes the form of a tree, i.e., each node has a unique parent (except the root). Thus, the number of edges in the OrderSPN is equal to the number of nodes, excluding the root, i.e. $\sum_{i=1}^{l}(2^i + 2^{i-1})\prod_{j<i} K_j$.

Now, each node represents a distribution over orders and graphs restricted to some subset of variables. Let $n(\boldsymbol{T}_i)$ denote the number of distinct orders in the support of the first node in $\boldsymbol{T}_i$ (similar for $n(\boldsymbol{P}_i)$). Notice that, since $d = 2^l$, all nodes in the layer $\boldsymbol{T}_i$ have support over the same number of orders. Thus, we need only consider how the number of orders covered changes as we move through the layers. Firstly note that, for the leaf layer $\boldsymbol{T}_l$, all nodes express distributions over $(\sigma_{\{i\}}, G_i)$ for some variable i. There is only one possible permutation over a singleton set, so $n(\boldsymbol{T}_l) = \sigma_{\{i\}}$. Then, for any sum-node in $\boldsymbol{T}_i$, by determinism, each child of $\boldsymbol{T}_i$ has disjoint support over orders, so it follows that $n(\boldsymbol{T}_i) = K_i \times n(\boldsymbol{T}_i)$. For any product-node in $\boldsymbol{P}_i$, we have that the two children of $\boldsymbol{P}_i$ express distributions over orders/permutations $\sigma_{S_{21}}, \sigma_{S_{22}}$, where S_{21}, S_{22} are disjoint sets. Since each of the children have support over $n(\boldsymbol{T}_{i+1})$ orders, the product node expressing a distribution over $\sigma_{S_{21}\cup S_{22}}$ has $n(\boldsymbol{P}_i) = n(\boldsymbol{T}_{i+1})^2$. It is worth comparing this to the corresponding relation $|\boldsymbol{T}_{l+1}| = 2|\boldsymbol{P}_l|$ above; the conditional independence asserted by the OrderSPN results in the compactness of the representation. We can now see (by induction) that $n(\boldsymbol{P}_i) = \prod_{j=i+1}^{l-1} K_j^{2^{j-i+1}}$ and $n(\boldsymbol{T}_i) = \prod_{j=i}^{l-1} K_j^{2^{j-i}}$, and so the root node has support size:

$$n(\boldsymbol{T}_0) = \prod_{j=0}^{l-1} K_j^{2^j}$$

□

Proposition 6.6 (Tractable ELBO)**.** *The ELBO and its gradients for any regular OrderSPN $\mathcal{C}$ and order-modular distribution p can be computed in linear time in the size of the circuit.*

Proof. This proof is based upon the corresponding result (Thm 1) in [166], but we provide it explicitly for OrderSPNs for completeness. Recall that an order-modular distribution takes the form $\tilde{p}(\sigma, G) = \prod_i p_{G_i}(G_i)\mathbb{1}_{G_i \subseteq \sigma^{<i}}$. Further, recall the definition of partial distributions (Definition 6.1):

$$\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2}) \triangleq \prod_{i\in S_2} p_{G_i}(G_i)\mathbb{1}_{G_i\subseteq S_1\cup\sigma_{S_2}^{<i}} \tag{C.4}$$

We can interpret each node of the OrderSPN as approximating a partial distribution; as such, we define the ELBO for a node as follows:

- **Sum:** For sum node T associated with $\rho(T) = (S_1, S_2)$, approximating $\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})$, we define:

$$ELBO(p_T) = \mathbb{E}_{p_T}[\log \tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})] + H(p_T(\sigma_{S_2}, G_{S_2})) \tag{C.5}$$

- **Product:** For product node P associated with $\rho(P) = (S_1, S_{21}, S_{22})$, approximating $\tilde{p}_{S_1,S_{21}\cup S_{22}}(\sigma_{S_{21}\cup S_{22}}, G_{S_{21}\cup S_{22}})$, we define:

$$ELBO(p_P) = \mathbb{E}_{p_P}[\log \tilde{p}_{S_1,S_{21}\cup S_{22}}(\sigma_{S_{21}\cup S_{22}}, G_{S_{21}\cup S_{22}})] + H(p_P(\sigma_{S_{21}\cup S_{22}}, G_{S_{21}\cup S_{22}})) \tag{C.6}$$

- **Leaf:** For leaf node L associated with $\rho(L) = (S_1, \{i\})$, approximating $\tilde{p}_{S_1,\{i\}}(\sigma_{\{i\}}, G_i)$, we define:

$$ELBO(p_L) = \mathbb{E}_{p_L}[\log \tilde{p}_{S_1,\{i\}}(\sigma_{\{i\}}, G_i)] + H(p_L(\sigma_{\{i\}}, G_i)) \tag{C.7}$$

We will now show how to decompose the computation of the ELBO of a node in terms of the ELBO of its children. Firstly, given a sum node T associated with $\rho(T) = (S_1, S_2)$, with children $N_1, ..., N_K$ and corresponding weights $\omega_{T,1}, ..., \omega_{T,K}$, we can write the expectation and entropy in Equation C.5 as:

$$\mathbb{E}_{p_T}[\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})] = \sum_{i=1}^{K} \omega_{T,i} \mathbb{E}_{p_{N_i}}[\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})] \tag{C.8}$$

$$\begin{aligned} H(p_T(\sigma_{S_2}, G_{S_2})) &= -\mathbb{E}_{p_T}[\log p_T(\sigma_{S_2}, G_{S_2})] & \text{(C.9)} \\ &= -\sum_{i=1}^{K} \omega_{T,i} \mathbb{E}_{p_{N_i}}[\log \sum_{j=1}^{K} \omega_{T,j} p_{N_j}(\sigma_{S_2}, G_{S_2})] & \text{(C.10)} \\ &= -\sum_{i=1}^{K} \omega_{T,i} \mathbb{E}_{p_{N_i}}[\log \omega_{T,i} p_{N_i}(\sigma_{S_2}, G_{S_2})] & \text{(C.11)} \\ &= -\sum_{i=1}^{K} \omega_{T,i} \log \omega_{T,i} - \sum_{i=1}^{K} \omega_{T,i} \mathbb{E}_{p_{N_i}}[\log p_{N_i}(\sigma_{S_2}, G_{S_2})] & \text{(C.12)} \\ &= -\sum_{i=1}^{K} \omega_{T,i} \log \omega_{T,i} + \sum_{i=1}^{K} \omega_{T,i} H(p_{N_i}(\sigma_{S_2}, G_{S_2})) & \text{(C.13)} \end{aligned}$$

In other words, the expectation decomposes as a weighted sum over expectations with respect to the child distributions, and the entropy decomposes as a sum of the entropy of the sum-node weights, and a weighted sum over entropies with respect to the child distributions. Note that the third equality in the derivation of the entropy decomposition holds only due to the fact that OrderSPNs are deterministic; this means that the children N_i have disjoint supports, and thus we can drop all terms with $j \neq i$ in the summation inside the expectation. Together, we have that:

$$ELBO(p_T) = \sum_{i=1}^{K} \omega_{T,i} \mathbb{E}_{p_{N_i}}[\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})] - \sum_{i=1}^{K} \omega_{T,i} \log \omega_{T,i} + \sum_{i=1}^{K} \omega_{T,i} H(p_{N_i}(\sigma_{S_2}, G_{S_2})) \tag{C.14}$$

$$= - \sum_{i=1}^{K} \omega_{T,i} \log \omega_{T,i} + \sum_{i=1}^{K} \omega_{T,i} (\mathbb{E}_{p_{N_i}}[\tilde{p}_{S_1,S_2}(\sigma_{S_2}, G_{S_2})] + H(p_{N_i}(\sigma_{S_2}, G_{S_2}))) \tag{C.15}$$

$$= - \sum_{i=1}^{K} \omega_{T,i} \log \omega_{T,i} + \sum_{i=1}^{K} \omega_{T,i} ELBO(p_{N_i}) \tag{C.16}$$

i.e. the ELBO of a sum-node can be expressed in terms of the ELBO of its children.

Secondly, given a product node P associated with $\rho(P) = (S_1, S_{21}, S_{22})$, with children N_1, N_2 associated with $\rho(N_1) = (S_1, S_{21})$ and $\rho(N_2) = (S_1 \cup S_{21}, S_{22})$, we can write the expectation and entropy in Equation C.6 as:

$$\mathbb{E}_{p_P}[\log \tilde{p}_{S_1, S_{21} \cup S_{22}}(\sigma_{S_{21} \cup S_{22}}, G_{S_{21} \cup S_{22}})] = \mathbb{E}_{p_P}\left[\log \prod_{i \in S_{21} \cup S_{22}} p_{G_i}(G_i) \mathbb{1}_{G_i \subseteq S_1 \cup \sigma^{<i}_{S_{21} \cup S_{22}}}\right] \tag{C.17}$$

$$= \mathbb{E}_{p_P}\left[\log \prod_{i \in S_{21}} p_{G_i}(G_i) \mathbb{1}_{G_i \subseteq S_1 \cup \sigma^{<i}_{S_{21}}}\right] + \mathbb{E}_{p_P}\left[\log \prod_{i \in S_{22}} p_{G_i}(G_i) \mathbb{1}_{G_i \subseteq S_1 \cup S_{21} \cup \sigma^{<i}_{S_{22}}}\right] \tag{C.18}$$

$$= \mathbb{E}_{p_P}[\log \tilde{p}_{S_1, S_{21}}(\sigma_{S_{21}}, G_{S_{21}})] + \mathbb{E}_{p_P}[\log \tilde{p}_{S_1 \cup S_{21}, S_{22}}(\sigma_{S_{22}}, G_{S_{22}})] \tag{C.19}$$

$$= \mathbb{E}_{p_{N_1}}[\log \tilde{p}_{S_1, S_{21}}(\sigma_{S_{21}}, G_{S_{21}})] + \mathbb{E}_{p_{N_2}}[\log \tilde{p}_{S_1 \cup S_{21}, S_{22}}(\sigma_{S_{22}}, G_{S_{22}})] \tag{C.20}$$

$$H(p_P(\sigma_{S_{21} \cup S_{22}}, G_{S_{21} \cup S_{22}})) = -\mathbb{E}_{p_P}[\log p_P(\sigma_{S_{21} \cup S_{22}}, G_{S_{21} \cup S_{22}})] \tag{C.21}$$

$$= -\mathbb{E}_{p_{N_1}}[\log p_{N_1}(\sigma_{S_{21}}, G_{S_{21}})] - \mathbb{E}_{p_{N_2}}[\log p_{N_2}(\sigma_{S_{22}}, G_{S_{22}})] \tag{C.22}$$

$$= H(p_{N_1}(\sigma_{S_{21}}, G_{S_{21}})) + H(p_{N_2}(\sigma_{S_{22}}, G_{S_{22}})) \tag{C.23}$$

This follows from decomposability, which ensures that the child distributions are over disjoint sets of variables (and are thus independent). Together, we have that:

$$\begin{aligned} ELBO(p_N) &= \mathbb{E}_{p_{N_1}}[\log \tilde{p}_{S_1,S_{21}}(\sigma_{S_{21}}, G_{S_{21}})] + \mathbb{E}_{p_{N_2}}[\log \tilde{p}_{S_1 \cup S_{21}, S_{22}}(\sigma_{S_{22}}, G_{S_{22}})] \quad \text{(C.24)} \\ &\quad + H(p_{N_1}(\sigma_{S_{21}}, G_{S_{21}})) + H(p_{N_2}(\sigma_{S_{22}}, G_{S_{22}})) \quad \text{(C.25)} \\ &= ELBO(p_{N_1}) + ELBO(p_{N_2}) \quad \text{(C.26)} \end{aligned}$$

i.e. the ELBO of a product node can be expressed in terms of the ELBO of its children.

By recursively applying ELBO equalities, we can express the ELBO for the overall OrderSPN $p_\mathcal{C}$ in terms of the weights $\boldsymbol{\omega}$ and ELBO for the leaf node distributions. Since each equality involves a sum/product over the children of the node (i.e., the outgoing edges), the overall computation takes linear time in the size (number of edges) of the SPN. The ELBO for the leaf nodes can be computed as in Proposition 6.7. □

C.2 OrderSPN Structure Learning Oracles

In this section we elaborate further the oracles $\mathcal{O}$ used for generating the structure of the OrderSPN in Section 6.3.1. As previously defined, the $\mathcal{O}$ takes as input some data $\mathcal{D}$, disjoint sets S_1, S_2, and a number of samples K, and returns K partitions $(S_{21,i}, S_{22,i})$ of S_2. The goal of the oracle is to maximize coverage of the posterior distribution, i.e. the posterior mass of orders consistent with at least one of the sampled partitions. Solving such a problem exactly is clearly intractable; thus we would like heuristic methods which can obtain good coverage.

A possible oracle would simply be to take K random partitions of S_2. However, this does not make efficient usage of the capacity of the OrderSPN. Thus, we consider adapting other Bayesian structure learning methods to take the role of the oracle. This can be done by sampling DAGs from the method; each such DAG naturally induces an order over the variables S_2, and thus a partition. Intuitively, we utilize their ability to find promising areas of the space of orders and DAGs to choose a better structure for our SPN.

The key practical challenge is that, is in contrast to the typical use case, we are not just interested in learning a DAG over a set S_2, but also want to allow the variables in

S_2 to have parents from some disjoint set S_1. This will require adaptations specific to the particular method chosen. In the rest of this section, we provide brief descriptions of how this can be done for DIBS and GADGET.

DIBS is a Bayesian structure learning approach based on particle variational inference [109]. In particular, in the marginal form, it assumes the following latent-variable generative model:

$$p(Z, G, \mathcal{D}) = p(Z)p(G|Z)p(\mathcal{D}|G)$$

where $Z = [U, V]$ with $U, V \in \mathbb{R}^{k\times d}$ (for some $k < d$) is a latent variable, generating the graph $G \in \{0, 1\}^{d\times d}$ and $\mathcal{D}$ is the dataset. In particular, the distribution for the graph takes the form:

$$p(G|Z) = \prod_{i,j} p(G_{ij}|Z) = \prod_{i,j} \sigma(\boldsymbol{u}_i^T \boldsymbol{v}_j)$$

Now suppose that we want to learn a DAG over S_2, where variables can additionally have parents in S_1. In this case, the natural generative model is to simply restrict to the components of the graph which are being modelled:

$$p_{S_1,S_2}(G|Z) = \prod_{i\in S_1\cup S_2} \prod_{j\in S_2} \sigma(\boldsymbol{u}_i^T \boldsymbol{v}_j)$$

The marginal likelihood $p(\mathcal{D}|G)$ is modular, so we can additionally restrict the likelihood to only concern the likelihood of S_2:

$$p_{S_2}(\mathcal{D}|G) = \prod_{j\in S_2} p(\mathcal{D}_j|G_j)$$

With these modifications, we have a valid generative model for any (S_1, S_2), to which the DIBS particle variational inference scheme can be applied with no further changes, giving us an oracle.

GADGET [188] is a MCMC method which samples over the space of *ordered partitions* of the set of variables (note this is distinct from the 2-partitions we use in OrderSPNs). Informally speaking, the ordered partition represents a partial ordering of the variables, where a variable must have a parent from the partition directly preceding its partition. For example, for $d = 6$, a partition might be $(\{3, 4, 5\}, \{1\}, \{2, 6\})$, where variable 1 must have one of $3, 4, 5$ as parent (but not any of $2, 6$). A $k-$partition R is scored using a modular score:

$$\pi(R) = \prod_{t=1}^{k} \prod_{j \in R_t} \tau_j(\cup_{i=1}^{t-1} R_i, R_{t-1})$$

where $\tau_j(U, T)$ is the summed score (posterior probability) that variable j has all parents contained in the set U, and at least one parent in the set T.

Gadget uses a similar type of precomputation to that used in Trust to precompute the functions $\tau_j(U, T)$, where a candidate parent set C_j of each variable is chosen in advance (using a heuristic) so that we actually compute $\tau_j(U \cap C_j, T \cap C_j)$.

Now, suppose we are given sets (S_1, S_2), and as usual seek to learn DAGs over S_2 which additionally have parents from S_1. This can be achieved by simply restricting the MCMC to only learn ordered partitions over S_2, while also allowing the parent sets of variables S_2 to be contained in $S_1 \cup S_2$. In particular, if we have precomputed $\tau_j(U \cap C_j, T \cap C_j)$ for all j and $U, T \subseteq \{1, ..., d\}$, this includes all of the necessary scores $\tau_j(U \cap C_j, T \cap C_j)$ for all $j \in S_2$ and $U, T \subseteq S_1 \cup S_2$, for any restriction (S_1, S_2).

The MCMC proceeds as if it were over a $|S_2|$ dimensional problem, over the set of variables S_2, but with modified scores involving S_1 as above, thus providing an oracle for TRUST.

C.3 Experimental Details

Bayesian network hyperparameters In our experiments, we consider linear Gaussian Bayesian networks, and generate Erdos-Renyi random structures, with expected numbers of edges given by $2d$. We generate data using fixed observation noise $\sigma^2 = 0.1$, and edge weights drawn independently from $\mathcal{N}(0, 1)$.

Posterior setup We use the fair prior over graph structures [55], where the prior probability of a mechanism having k edges is proportional to the inverse of the number of different parents sets of size k. In addition, we use the BGe marginal likelihood [95] with hyperparameters $\alpha_\mu = 1$, $\alpha_w = d + 2$, and $T = \frac{1}{2} I$ where I is the $d \times d$ identity matrix.

Implementation details Our implementations of Dibs and Gadget are based on the reference implementations with the default settings of hyperparameters. In particular, we ran Dibs with $N = 30$ particles and 3000 epochs using the marginal inference method, while Gadget was run using 16 coupled chains and for 320000 MCMC iterations, extracting $N = 10000$ samples.

Our implementation of TRUST uses the PyTorch framework to tensorize passes through the SPN, following the regular OrderSPN structure described in the main paper. In the $d = 16, 32$ cases, we used expansion factors of $\boldsymbol{K} = [64, 16, 6, 2], [32, 8, 2, 6, 2]$ respectively; these were chosen empirically to approximately match oracle computation across layers. Parameter learning in the SPN was performed by optimizing the ELBO objective using the Adam optimizer with learning rate 0.1 and for 700 iterations. Operations in the circuit are performed in log-space for numerical stability.

Inference Queries We perform inference for DiBS and Gadget by applying the appropriate calculation over the sample (for instance, the marginal probability of an edge G_ij is simply the proportion of sampled DAGs in which it appears), while for Trust, we perform inference directly on the OrderSPN using the queries described in the Chapter when this is possible, and by sampling otherwise (e.g. E-SHD).