A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference

Antonio Vergari[*]
School of Informatics
University of Edinburgh
avergari@ed.ac.uk

YooJung Choi
CS Department
UCLA
yjchoi@cs.ucla.edu

Anji Liu
CS Department
UCLA
liuanji@cs.ucla.edu

Stefano Teso
CS Department
University of Trento, Italy
stefano.teso@unitn.it

Guy Van den Broeck
CS Department
UCLA
guyvdb@cs.ucla.edu

Abstract

Circuit representations are becoming the lingua franca to express and reason about tractable generative and discriminative models. In this paper, we show how complex inference scenarios for these models that commonly arise in machine learning—from computing the expectations of decision tree ensembles to information-theoretic divergences of sum-product networks—can be represented in terms of tractable modular operations over circuits. Specifically, we characterize the tractability of simple transformations—sums, products, quotients, powers, logarithms, and exponentials—in terms of sufficient structural constraints of the circuits they operate on, and present novel hardness results for the cases in which these properties are not satisfied. Building on these operations, we derive a unified framework for reasoning about tractable models that generalizes several results in the literature and opens up novel tractable inference scenarios.

1 Introduction

Many core computational tasks in machine learning (ML) and AI involve solving *complex integrals*, such as expectations appearing in training losses or in information-theoretic quantities including entropies or divergences. A fundamental question naturally arises: *under which conditions do these quantities admit tractable computation?* Or equivalently, when can we compute them *reliably and efficiently* without resorting to approximations or heuristics? If we are able to find model classes to tractably compute these quantities of interest—henceforth called *queries*—we can then design efficient algorithms with important applications in learning, approximate inference [44], model compression [28], explainable AI [24, 47, 52] and algorithmic bias detection [23, 7, 6].

This "quest" for tracing the tractability of different queries has been carried out several times, often independently for different model classes in ML and AI and crucially, for each query in isolation. For example, the computation of the Kullback-Leibler divergence (KLD) is known to have a closed form for Gaussians, but only recently has an exact algorithm been derived for a more complex tractable model class such as probabilistic sentential decision diagrams (PSDDs) [28]. On the other hand, tractable computation of the entropy, despite being a sub-routine for the KLD, has only been derived for a different tractable model class—selective sum-product networks (SPNs) [36]—by Shih and Ermon [44]. In the current paradigm, if one were to trace the tractability of a query that has not

[*]This work was done while AV was at the CS Department at UCLA.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

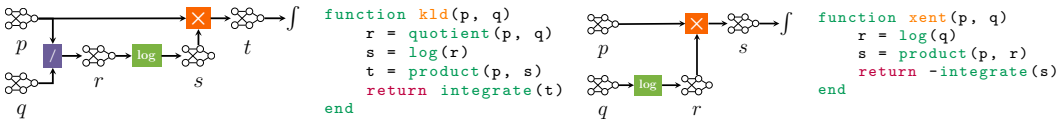


Figure 1: Computational pipelines of the KLD (left) and cross entropy (right) over two distributions p and q encoded as circuits, with the intermediate computations (r, s and t) also represented as circuits. Their corresponding implementations in a few lines of Julia code are shown on their right.

yet been investigated but still involves the same "building blocks" such as logarithms, integrals and products over distributions, for instance Rényi's alpha divergence [40], they would need to derive a novel custom algorithm for each model class and prove its tractability from scratch.

In this paper, we take a different path and introduce a general framework under which the tractability of complex queries can be traced in a *unified and effortless manner over model classes and query classes*. To abstract from the different model formalisms, we carry our analysis over circuit representations [8] as they subsume many tractable generative models—probabilistic circuits such as Chow-Liu trees [9], hidden Markov models (HMMs) [39], sum-product networks (SPNs) [38], and other deep mixture models—as well as discriminative ones, including decision trees [25, 10] and deep regressors [23], thus enabling a unified treatment across model classes.

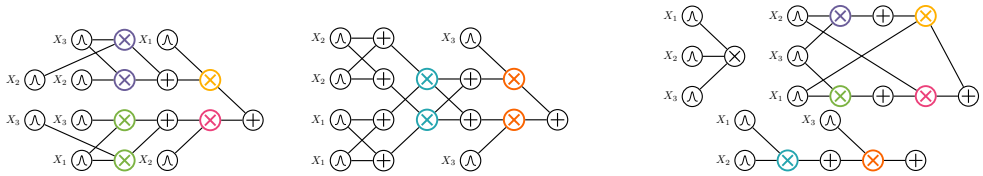
To generalize our analysis across queries, we propose to represent a single query as a *circuit pipeline*: a computational graph whose intermediate operations transform and combine the input circuits into other circuits. We can first build a set of simple tractable circuit transformations—sums, products, powers, logarithms, and exponentials—and then i) analyze the tractability of a single query by propagating the sufficient conditions for tractability of the intermediate operators in the pipeline; and ii) automatically distill a tractable inference algorithm by composing the operators used. For instance, Fig. 1 shows the pipeline for computing the KLD of p and q, two distributions encoded by circuits. We can identify a general class of models that supports its tractable computation: by tracing the conditions for tractable quotient, logarithm, and product over circuits such that the output circuit (i.e., t) admits tractable integration, we can derive a set of sufficient conditions for the input circuits. Moreover, we can *reuse* the logarithm and product operations in the KLD pipeline to reason about the tractability of cross entropy, in the very same way we can reuse the corresponding subroutines we provide in Julia to quickly implement algorithms for the two queries in a couple lines of code as shown in Fig. 1. This compositionality greatly speeds up the design of novel tractable algorithms.

We make the following contributions: (1) a systematic way to compositionally answer many complex queries using simple circuit transformations (Sec. 3), proving sufficient conditions for their tractability and computational hardness when these conditions are unmet (Tab. 1); (2) a unification and generalization of many inference algorithms proposed in the literature so far for specific representations (Sec. 4); (3) novel tractability and hardness results of complex information-theoretic queries including several widely used entropies and divergences (Tab. 2); and (4) a publicly available implementation of these operators in the Juice circuit library [12]. We now start by introducing the circuit language.

2 Circuit Representations

Circuits represent functions as parameterized computational graphs. By imposing certain structural constraints on these graphs, we can guarantee the tractability of certain operations over the encoded functions. Moreover, these constraints help understand how *circuits unify several classical tractable model classes*, such as mixture models, bounded-treewidth probabilistic graphical models (PGMs), decision trees, and compact logical function representations [8, 51]. As such, circuits provide *a language for building and reasoning about tractable representations*, and it follows that all our results in the following sections automatically translate to these model classes.

We introduce the basic rules of this language by distinguishing between general circuits and those encoding probability distributions, as some operators in Sec. 3 may be restricted to the latter. Then, we will review the structural constraints we need to characterize different inference scenarios, also known as classes of queries. We denote random variables by uppercase letters (X) and their assignments by lowercase ones (x). Sets of variables and their assignments are denoted by bold uppercase ($\mathbf{X}$) and bold lowercase ($\boldsymbol{x}$) letters, respectively, and the set of all their values as $\mathsf{val}(\mathbf{X})$.



(a) Decomposable (b) Structured-decomposable (c) Omni-compatible

Figure 2: Examples of circuits with different structural properties. The feedforward order is from left to right; input units are labeled by their scopes; and sum parameters are omitted for visual clarity. Product units of the rearranged omni-compatible circuits encoding $p(X_1) \cdot p(X_2) \cdot p(X_3)$ are shown in (c) and color-coded with those of matching scope in (a) and (b).

Definition 2.1 (Circuit)**.** A circuit p over variables $\mathbf{X}$ is a parameterized computational graph encoding a function $p(\mathbf{X})$ and comprising three kinds of computational units: *input*, *product*, and *sum*. Each inner unit n (i.e., product or sum unit) receives inputs from other units, denoted $\mathsf{in}(n)$. If n is an input unit, it encodes a parameterized function $p_n(\phi(n))$ over variables $\phi(n) \subseteq \mathbf{X}$, also called its *scope*. Instead, if n is a sum unit, it encodes $\sum_{c \in \mathsf{in}(n)} \theta_c p_c(\phi(c))$ where $\theta_c \in \mathbb{R}$ are the sum parameters; while if it is a product unit, it encodes $\prod_{c \in \mathsf{in}(n)} p_c(\phi(c))$. The scope of an inner unit is the union of the scopes of its inputs: $\phi(n) = \bigcup_{c \in \mathsf{in}(n)} \phi(c)$. The output unit of the circuit is the last unit (i.e., with out-degree 0) in the graph, encoding $p(\mathbf{X})$. The *support* of p is the set of all complete states for $\mathbf{X}$ for which the output of p is non-zero: $\mathsf{supp}(p) = \{\boldsymbol{x} \in \mathsf{val}(\mathbf{X}) \mid p(\boldsymbol{x}) \neq 0\}$.

Circuits can be understood as compact representations of polynomials with exponentially many terms, whose indeterminates are the functions encoded by the input units. These functions are assumed to be simple enough to allow tractable computations of the operations discussed in this paper. Fig. 2 shows some examples of circuits. A *probabilistic circuit* (PC) [8] represents a (possibly unnormalized) probability distribution by encoding its probability mass, density, or a combination thereof.

Definition 2.2 (Probabilistic circuit)**.** A PC over variables $\mathbf{X}$ is a circuit encoding a function p that is non-negative for all values of $\mathbf{X}$; i.e., $\forall \boldsymbol{x} \in \mathsf{val}(\mathbf{X}) : p(\boldsymbol{x}) \geq 0$.

From here on, we will assume that a PC has positive sum parameters and input units that model valid (unnormalized) distributions, which is a sufficient condition to satisfy the above definition. Moreover, w.l.o.g. we will assume that each layer of a circuit alternates between sum and product units and that every product unit n receives only two inputs c_1, c_2, i.e., $p_n(\mathbf{X}) = p_{c_1}(\mathbf{X}) \cdot p_{c_2}(\mathbf{X})$. These conditions can easily be enforced on a circuit in exchange for only a polynomial increase in its size [49, 50].

Computing (functions of) $p(\mathbf{X})$, or in other words performing *inference*, can be done by evaluating its computational graph. Hence, the computational cost of inference on a circuit is a function of its *size*, defined as the number of edges in it and denoted as $|p|$. For instance, querying the value of p for a complete assignment $\boldsymbol{x}$ equals its *feedforward* evaluation—inputs before outputs—and therefore is linear in $|p|$. Other common inference scenarios such as function integration—which translate to *marginal inference* in the context of probability distributions—can be tackled in linear time with circuits that exhibit certain structural properties, as discussed next.

Structural Properties of Circuits. Structural constraints on the computational graph of a circuit in terms of its scope or support provide sufficient and/or necessary conditions for certain queries to be tractably computed. We now define the structural properties needed for the query classes that this work will focus on, referring to Choi et al. [8] for more details.

Definition 2.3 (Smoothness)**.** A circuit is *smooth* if for every sum unit n, its inputs depend on the same variables: $\forall\, c_1, c_2 \in \mathsf{in}(n), \phi(c_1) = \phi(c_2)$.

Smooth PCs generalize shallow mixture models [30] to deep and hierarchical models. For instance, a Gaussian mixture model (GMM) can be represented as a smooth PC with a single sum unit over as many input units as mixture components, each encoding a (multivariate) Gaussian density.

Definition 2.4 (Decomposability)**.** A circuit is *decomposable* if the inputs of every product unit n depend on disjoint sets of variables: $\mathsf{in}(n) = \{c_1, c_2\}, \phi(c_1) \cap \phi(c_2) = \emptyset$.

Decomposable product units encode local factorizations. That is, a decomposable product unit n over variables $\mathbf{X}$ encodes $p_n(\mathbf{X}) = p_1(\mathbf{X}_1) \cdot p_2(\mathbf{X}_2)$ where $\mathbf{X}_1$ and $\mathbf{X}_2$ form a partition of $\mathbf{X}$. Taken together, decomposability and smoothness are a sufficient and necessary condition for performing tractable integration over arbitrary sets of variables in a single feedforward pass, as they enable larger integrals to be efficiently decomposed into smaller ones [15, 8]. Next proposition formalizes it.

Proposition 2.1 (Tractable integration, Choi et al. [8])**.** *Let p be a smooth and decomposable circuit over $\mathbf{X}$ with input functions that can be tractably integrated. Then for any variables $\mathbf{Y} \subseteq \mathbf{X}$ and their assignment $\boldsymbol{y}$, the integral $\int_{\boldsymbol{z} \in \mathsf{val}(\mathbf{Z})} p(\boldsymbol{y}, \boldsymbol{z}) d\mathbf{Z}$ can be computed exactly in $\Theta(|p|)$ time, where $\mathbf{Z}$ denotes $\mathbf{X} \setminus \mathbf{Y}$.*

As the complex queries we focus on in this work involve integration as the last step, it is therefore needed that any intermediate operation preserves at least decomposability; smoothness is less of an issue, as it can be enforced in polytime [45]. A key additional constraint over scope decompositions is *compatibility*. Intuitively, two decomposable circuits are compatible if they can be rearranged in polynomial time[2] such that their respective product units, once matched by scope, decompose in the same way. We formalize this with the following inductive definition.

Definition 2.5 (Compatibility)**.** Two circuits p and q over variables $\mathbf{X}$ are *compatible* if (1) they are smooth and decomposable and (2) any pair of product units $n \in p$ and $m \in q$ with the same scope can be rearranged into binary products that are mutually compatible and decompose in the same way: $(\phi(n) = \phi(m)) \implies (\phi(n_i) = \phi(m_i),\ n_i$ and m_i are compatible) for some rearrangement of the inputs of n (resp. m) into n_1, n_2 (resp. m_1, m_2).

We can derive from compatibility the following properties pertaining to a single circuit, which will be useful in our analysis later.

Definition 2.6 (Special types of compatibility)**.** A circuit is *structured-decomposable* if it is compatible with itself. A decomposable circuit p over $\mathbf{X}$ is *omni-compatible* if it is compatible with any smooth and decomposable circuit over $\mathbf{X}$.

Not all decomposable circuits are structured-decomposable (see Figs. 2a and 2b), but some can be rearranged to be compatible with any decomposable circuit. For instance, in Fig. 2c, the fully factorized product unit $p(\mathbf{X}) = p_1(X_1) \cdot p_2(X_2) \cdot p_3(X_3)$ can be rearranged into $p_1(X_1) \cdot (p_2(X_2) \cdot p_3(X_3))$ and $p_2(X_2) \cdot (p_1(X_1) \cdot p_3(X_3))$ to match the yellow and pink products in Fig. 2a. We can easily see that omni-compatible circuits must assume the form of mixtures of fully-factorized models; i.e., $\sum_i \theta_i \prod_j p_{i,j}(X_j)$. For example, an additive ensemble of decision trees over variables $\mathbf{X}$ can be represented as an omni-compatible circuit (see Ex. D.1 in the Appendix). Also note that if two circuits are compatible and neither is omni-compatible, then both must be structured decomposable.

Definition 2.7 (Determinism)**.** A circuit is *deterministic* if the inputs of every sum unit n have disjoint supports: $\forall\, c_1, c_2 \in \mathsf{in}(n), c_1 \neq c_2 \implies \mathsf{supp}(c_1) \cap \mathsf{supp}(c_2) = \emptyset$.

Analogously to decomposability, determinism induces a recursive partitioning, but this time over the support of a circuit. For a deterministic sum unit n, the partitioning of its support can be made explicit by introducing an indicator function per each of its inputs, i.e., $\sum_{c \in \mathsf{in}(n)} \theta_c p_c(\boldsymbol{x}) = \sum_{c \in \mathsf{in}(n)} \theta_c p_c(\boldsymbol{x}) [\![\boldsymbol{x} \in \mathsf{supp}(p_c)]\!]$. Determinism allows for tractable maximization of circuits [13, 8]. While we do not consider maximization queries in this work, determinism will still play a crucial role in the next sections. Moreover, bounded-treewidth PGMs, such as Chow-Liu trees [9] and thin junction trees [1], can efficiently be represented as smooth, deterministic, and decomposable PCs via *compilation* [13, 11]. Probabilistic sentential decision diagrams (PSDDs) [26] are deterministic and structured-decomposable PCs that can be efficiently learned from data [11].

3 From Simple Circuit Transformations...

This section aims to build an atlas of simple operations over circuits which can then be composed into more complex queries via circuit pipelines—computational graphs whose units are tractable operators over circuits. To compose two operators, we would need that the output circuits of one satisfy the structural properties required for the inputs of the other. As such, for each of these operations we are interested in characterizing (1) its tractability in terms of the structural properties of its input circuits,

[2] By changing the order in which n-ary product units are turned into a series of binary product units.

Table 1: ***Tractability and hardness of simple circuit operations***. Tractable properties on inputs translate to properties on outputs. E.g., for the quotient p/q, if p and q are compatible (Cmp) and q is deterministic (Det), then the output is decomposable (Dec); also (+) deterministic if p is deterministic; and structured-decomposable (SD) if both p and q are. Hardness results are for representing the output as a smooth (Sm) and decomposable circuit without some input condition.

Operation		Tractability				Hardness	
		Input properties	Output properties	Time Complexity			
SUM	$\theta_1 p + \theta_2 q$	(+Cmp)	(+SD)	$\mathcal{O}(\lvert p\rvert+\lvert q\rvert)$	(Prop. B.1)	NP-hard for Det output [43]	
PRODUCT	$p \cdot q$	Cmp (+Det, +SD)	Dec (+Det, +SD)	$\mathcal{O}(\lvert p\rvert\lvert q\rvert)$	(Thm. 3.2)	#P-hard w/o Cmp	(Thm. 3.1)
POWER	$p^n, n \in \mathbb{N}$	SD (+Det)	SD (+Det)	$\mathcal{O}(\lvert p^n\rvert)$	(Thm. 3.3)	#P-hard w/o SD	(Thm. 3.3 and B.1)
	$p^\alpha, \alpha \in \mathbb{R}$	Sm, Dec, Det (+SD)	Sm, Dec, Det (+SD)	$\mathcal{O}(\lvert p\rvert)$	(Thm. 3.5)	#P-hard w/o Det	(Thm. 3.4)
QUOTIENT	p/q	Cmp; q Det (+p Det,+SD)	Dec (+Det,+SD)	$\mathcal{O}(\lvert p\rvert\lvert q\rvert)$	(Thm. B.3)	#P-hard w/o Det	(Thm. B.2)
LOG	$\log(p)$	Sm, Dec, Det	Sm, Dec	$\mathcal{O}(\lvert p\rvert)$	(Thm. 3.6)	#P-hard w/o Det	(Thm. 3.6)
EXP	$\exp(p)$	linear	SD	$\mathcal{O}(\lvert p\rvert)$	(Prop. 3.1)	#P-hard	(Thm. 3.7)

and (2) its closure w.r.t. these properties, i.e. whether they are preserved in the output circuit, in order to compose many operations together in a pipeline, while (3) providing an efficient algorithmic implementation for it. As we are interested in pipelines for queries involving integration, we would expect the output circuits to at least retain decomposability (see Prop. 2.1). For a pipeline in which all operators can be computed tractably, a simple tractable algorithm can be then distilled for it. Furthermore, our analysis will highlight if one needs to resort to approximations, by tracing the hardness of representing the output of an operator as a decomposable circuit when some property of its inputs is unmet. Tab. 1 summarizes all our results. For space constraints, we discuss the main theorems next and report their complete statements and proofs in the Appendix.

Sum of Circuits. The operation of summing two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ is defined as $s(\mathbf{X}) = \theta_1 \cdot p(\mathbf{Z}) + \theta_2 \cdot q(\mathbf{Y})$ for $\mathbf{X} = \mathbf{Z} \cup \mathbf{Y}$ and two real parameters $\theta_1, \theta_2 \in \mathbb{R}$. This operation, which is at the core of additive ensembles of tractable representations,[3] can be realized by introducing a single sum unit that takes as input p and q. Summation applies to any input circuits, regardless of structural assumptions, and it preserves several properties (see Prop. B.1 in the Appendix). In particular, if p and q are decomposable then s is also decomposable; moreover, if they are compatible then s is structured-decomposable as well as compatible with p and q. However, representing a sum as a deterministic circuit is known to be NP-hard [43], even for compatible and deterministic inputs.

Product of Circuits. The product of two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ can be expressed as $m(\mathbf{X}) = p(\mathbf{Z}) \cdot q(\mathbf{Y})$ for variables $\mathbf{X} = \mathbf{Z} \cup \mathbf{Y}$. If $\mathbf{Z}$ and $\mathbf{Y}$ are disjoint, the product m is already decomposable. Otherwise, Shen et al. [43] proved that representing the product of two decomposable circuits as a decomposable circuit is NP-hard, even if they are deterministic. We prove a novel result: it is even #P-hard to multiply two structured-decomposable and deterministic circuits.

Theorem 3.1 (Hardness of product)**.** *If p and q are two structured-decomposable and deterministic circuits, then computing their product as a decomposable circuit is #P-hard.*

Shen et al. [43] also introduced an efficient algorithm for the product of two structured-decomposable and deterministic PCs that are compatible (namely PSDDs). We generalize this result by proving that compatibility alone is sufficient for the tractable product computation of any two circuits.

Theorem 3.2 (Tractable product)**.** *If p and q are two compatible circuits, then computing their product as a decomposable circuit that is compatible with them can be done in $\mathcal{O}(\lvert p\rvert\,\lvert q\rvert)$ time.*

The proof is by construction and leads to Alg. 3 in the Appendix. In the following, we provide a sketch of the algorithm for the case $\mathbf{X} = \mathbf{Z} = \mathbf{Y}$. Intuitively, the idea is to "break down" the construction of the product circuit in a recursive manner by exploiting compatibility. The base case is where p and q are input units with simple parametric forms. Their product can be represented as a single input unit as long as we can find a simple parametric form for it, as is the case for products of exponential families such as (multivariate) Gaussians. Next, we consider the inductive steps where p and q are two sum or product units. If p and q are compatible product units, they decompose $\mathbf{X}$ the same way for some ordering of inputs; i.e., $p(\mathbf{X}) = p_1(\mathbf{X}_1)p_2(\mathbf{X}_2)$ and $q(\mathbf{X}) = q_1(\mathbf{X}_1)q_2(\mathbf{X}_2)$. Then, their product m as a decomposable circuit can be constructed recursively from the products of their inputs: $m(\mathbf{X}) = (p_1 q_1)(\mathbf{X}_1) \cdot (p_2 q_2)(\mathbf{X}_2)$. On the other hand, if p and q are smooth sum units, written as $p(\mathbf{X}) = \sum_i \theta_i p_i(\mathbf{X})$ and $q(\mathbf{X}) = \sum_j \theta'_j q_j(\mathbf{X})$, we can obtain their product m recursively

[3] If p and q are PCs, then s is a PC encoding a monotonic mixture model if $\theta_1, \theta_2 > 0$ and $\theta_1 + \theta_2 = 1$.

by distributing product over sum. In other words, $m(\mathbf{X}) = \sum_{i,j} \theta_i \theta'_j (p_i q_j)(\mathbf{X})$. Note that if both input circuits are also deterministic, m is also deterministic since $\mathsf{supp}(p_i q_j) = \mathsf{supp}(p_i) \cap \mathsf{supp}(q_j)$ are disjoint for different i, j. Combining these, the algorithm will recursively compute the product of each pair of units in p and q with matching scopes. Assuming efficient products for input units, the overall complexity is $\mathcal{O}(|p|\,|q|)$, which yields a compact circuit m of size $\mathcal{O}(|p|\,|q|)$. This upper bound is loose and in practice product circuits will be much smaller as our experiments show (Sec. 5), especially if inputs are deterministic as products of units with disjoint supports will be "pruned" away.

Powers of a Circuit. The α-power of a PC $p(\mathbf{X})$ for an $\alpha \in \mathbb{R}$ is denoted as $p^\alpha(\mathbf{X})$ and is an operation needed to compute generalizations of the entropy of a PC and related divergences (Sec. 4). Let us first consider natural powers ($\alpha \in \mathbb{N}$) which can be computed even for general circuits.

Theorem 3.3 (Natural powers)**.** *If p is a structured-decomposable circuit, then for any $\alpha \in \mathbb{N}$, its power can be represented as a structured-decomposable circuit in $\mathcal{O}(|p|^\alpha)$ time. Otherwise, if p is only smooth and decomposable, then computing $p^\alpha(\mathbf{X})$ as a decomposable circuit is #P-hard.*

The proof for tractability easily follows by directly applying the product operation repeatedly. However, the exponential dependence on α is unavoidable unless P=NP as we demonstrate in Thm. B.1 in the Appendix, thus rendering the operation intractable for large α.

Turning our attention to non-natural $\alpha \in \mathbb{R}$, and restricting our attention to PCs, structured-decomposability is not sufficient to tractably compute α-powers, which we will show in the next theorem for $\alpha = -1$. First, as zero raised to a negative power is undefined, we instead consider the *restricted α-power* of a PC, denoted as $p^\alpha(\boldsymbol{x})|_{\mathsf{supp}(p)}$ and equal to $(p(\boldsymbol{x}))^\alpha$ if $\boldsymbol{x} \in \mathsf{supp}(p)$ and 0 otherwise. Note that this is equivalent to the α-power if $\alpha \geq 0$. Abusing notation, we will also denote this by $p^\alpha(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$, where $[\![\cdot]\!]$ stands for indicator functions.

Theorem 3.4 (Hardness of reciprocals)**.** *If p is a structured-decomposable circuit over variables $\mathbf{X}$, then computing $p^{-1}(\mathbf{X})\big|_{\mathsf{supp}(p)}$ as a decomposable circuit is #P-hard.*

The key property that enables efficient computation of power circuits is determinism. More interestingly, we do not require structured-decomposability, but only smoothness and decomposability.

Theorem 3.5 (Tractable real powers)**.** *If p is a smooth, decomposable, and deterministic PC, then for any $\alpha \in \mathbb{R}$, its restricted power can be represented as a smooth, decomposable, and deterministic circuit that is compatible with p in $\mathcal{O}(|p|)$ time.*

Again, the proof is done by construction and detailed in Sec. B.3. The key insight is that restricted powers "break down" over a smooth and deterministic sum unit p. That is, $(\sum_i \theta_i p_i(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!])^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] = \sum_i \theta_i^\alpha p_i^\alpha(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!]$. This follows from the fact that for any $\boldsymbol{x}$, at most one indicator $[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!]$ evaluates to 1. As such, when multiplying a deterministic sum unit with itself, each input will only have overlapping support with itself, thus effectively matching product units only with themselves. This is why decomposability suffices. In conclusion, this recursive decomposition of the power of a circuit will result in the power circuit having the same structure as the original circuit, with input functions and sum parameters replaced by their α-powers. The space and time complexity of the algorithm is $\mathcal{O}(|p|)$ for smooth, deterministic, and decomposable PCs, even for natural powers. This will be a key insight to compactly multiply circuits with the same support structure, such as when computing logarithms and entropies (Sec. 4).

We can already see an example of how simple operators can be composed to derive other tractable ones. Consider the quotient of two circuits $p(\mathbf{X})$ and $q(\mathbf{X})$, denoted as $p(\mathbf{X})/q(\mathbf{X})$, and restricted to $\mathsf{supp}(q)$. The quotient, appearing in queries such as KLD or Itakura-Saito divergence (Sec. 4), can be computed by first taking the reciprocal circuit (i.e., the (-1)-power) of q, followed by its product with p. Thus, if q is deterministic and compatible with p, we can take its reciprocal—which will have the same structure as q—and multiply with p to obtain the quotient as a decomposable circuit as we show in Thm. B.3 in the Appendix. There, Thm. B.2 instead proves that the quotient between p and a non-deterministic q is #P-hard even if they are compatible.

Logarithms of a PC. The logarithm of a PC $p(\mathbf{X})$, denoted $\log p(\mathbf{X})$, is fundamental in computing quantities such as entropies and divergences between distributions (Sec. 4). Since the log is undefined for 0 we will again consider the *restricted logarithm*, denoted as $\log p(\boldsymbol{x})|_{\mathsf{supp}(p)}$ and equal to $\log p(\boldsymbol{x})$ if $\boldsymbol{x} \in \mathsf{supp}(p)$ and 0 otherwise.

Theorem 3.6 (Logarithms)**.** *If p is a smooth, deterministic and decomposable PC, then its restricted logarithm $\log p(\mathbf{X})|_{\mathsf{supp}(p)}$ can be represented as a decomposable circuit in $\mathcal{O}(|p|)$ time. Otherwise, if p is only smooth and decomposable, or even structured-decomposable, computing its restricted logarithm as a decomposable circuit is #P-hard.*

Note that while the input of the logarithm operator must be a PC, its output can be a general circuit. Moreover, if p is structured decomposable, then so is its logarithm. The tractability proof is again by construction and is detailed in Sec. B.5 in the Appendix. We point out that determinism again allows the restricted log to decompose over the support of the PC, but this time the output circuit is *not* deterministic, as more than one of its inputs can yield a non-zero output at a time. Nevertheless, the inputs of the newly introduced sum units can be clearly partitioned into groups sharing the same support of the corresponding product units in p. This acts as a relaxed form of determinism when at most three inputs can be non-zero at once, and it implies that whenever we multiply a deterministic circuit and its logarithmic circuit—for instance to compute its Shannon entropy (Sec. 4)—we can leverage the sparsifying effect of non-overlapping supports and perform only a linear number of products (cf. product and power operators). We confirm this empirically in Sec. 5 when evaluating the size of the intermediate circuits for computing entropy pipelines over real-world distributions.

Exponentials of a Circuit. The exponential of a circuit $p(\mathbf{X})$, denoted $\exp(p(\mathbf{X}))$, is the inverse operation of the logarithm and is a fundamental operation when representing distributions such as log-linear models [27]. Similarly to the logarithm, building a decomposable circuit that encodes an exponential of a circuit is hard in general.

Theorem 3.7 (Hardness of exponentials)**.** *If p is a smooth and decomposable circuit, then, computing its exponential as a decomposable circuit is #P-hard, even if p is structured-decomposable.*

Unlike the logarithm however, restricting the operation to deterministic circuits does not help with tractability, since the issue comes from product units: the exponential of a product is neither a sum nor product of exponentials. Nevertheless, it is easy to see that if p encodes a linear sum over its variables, i.e., $p(\mathbf{X}) = \sum_i \theta_i X_i$, we could easily represent its exponential as a circuit comprising a single decomposable product unit, hence tractably.

Proposition 3.1 (Tractable exponential of a linear circuit)**.** *If p is a linear circuit, then its exponential can be represented as an omni-compatible circuit in $\mathcal{O}(|p|)$ time.*

Note that if we were to add an additional deterministic sum unit over many omni-compatible circuits built in this way, we would retrieve a mixture of truncated exponentials [32, 55]. This is the largest class of tractable exponentials we know so far, and enlarging its boundaries is an open problem.

More operators? Our compositional atlas is now complete. In fact, if we were to add an additional circuit operator to the atlas, it would have to take the form of the already discussed operators. First, we require from any functional f to be applied to a circuit $p(\mathbf{X})$ to yield a smooth and decomposable circuit $f(p(\mathbf{X}))$ in order to admit tractable integration and to be added to our atlas. To that end, as usual we can assume to apply f to the input units of p and obtain tractable representations for the new input units; this is generally the case for simple parametric input functions. Next, we would require f to decompose over products and over sums. In other words, we first need that $f(p_1(\mathbf{X}_1) \cdot p_2(\mathbf{X}_2))$ can be broken down to either a product $f(p_1(\mathbf{X}_1)) \cdot f(p_2(\mathbf{X}_2))$ or sum $f(p_1(\mathbf{X}_1)) + f(p_2(\mathbf{X}_2))$. Furthermore, we want f to similarly decompose over sum units; that is, $f(p_1(\mathbf{X}_1) + p_2(\mathbf{X}_2))$ also yields a product or sum of $f(p_1(\mathbf{X}_1))$ and $f(p_2(\mathbf{X}_2))$. As the next lemma states, a non-linear function f that satisfies either of the above two conditions must be a power, logarithmic, or exponential function.

Lemma 3.8. *Let f be a continuous function. If $f : \mathbb{R} \to \mathbb{R}$ satisfies $f(x + y) = f(x) + f(y)$ then it is a linear function $\beta \cdot x$; if $f : \mathbb{R}_+ \to \mathbb{R}_+$ satisfies $f(x \cdot y) = f(x) \cdot f(y)$, then it takes the form x^β; if instead $f : \mathbb{R}_+ \to \mathbb{R}$ satisfies $f(x \cdot y) = f(x) + f(y)$, then it takes the form $\beta \log(x)$; and if $f : \mathbb{R} \to \mathbb{R}_+$ satisfies that $f(x + y) = f(x) \cdot f(y)$ then it is of the form $\exp(\beta \cdot x)$, for a certain $\beta \in \mathbb{R}$.*

4 … to Complex Compositional Queries

In this section, we show how our atlas of simple tractable operators can be effectively used to systematically find a tractable model class for *any advanced query that comprises these operators*.

Table 2: ***Tractability and hardness of information-theoretic queries over circuits.*** Tractability given some conditions over the input circuits; computational hardness when some of these are unmet.

Query		Tract. Conditions	Hardness	Reference
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x}) \, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det	Thm. C.1
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det	Thm. C.2
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD	Thm. C.5
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det	#P-hard w/o Det	Thm. C.6
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD	Thm. C.3
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x})) d\mathbf{X}$	Cmp, Det	#P-hard w/o Det	Thm. C.4
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det	Thm. 4.1&C.7
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det	Thm. 4.1&C.7
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det	Thm. C.8
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp	Thm. C.9
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp	Thm. C.10

We will show its practical utility by quickly coming up with tractability proofs as well as distilling efficient algorithms for several entropy and divergence queries that are largely used in ML. We will then discuss how our discovered tractable circuit classes subsume some previously known results in the literature and prove novel hardness results for when the structural properties of these circuits are unmet. Tab. 2 summarizes our results.

We now showcase how a short tractability proof can be easily distilled, using Rényi's α-divergence[4] [40] as an example. Note that no tractable algorithm was available for it yet. A proof can be built by inferring the sufficient conditions to tractably compute each operator in the pipeline—starting from the last before the integral and proceeding backwards according to Tab. 1.

Theorem 4.1 (Tractable alpha divergence)**.** *The Rényi's α-divergence between two distributions p and q, defined as $(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}$, can be computed exactly in $\mathcal{O}(|p|^{\alpha} |q|)$ time for $\alpha \in \mathbb{N}, \alpha > 1$ if p and q are compatible and q is deterministic, or in $\mathcal{O}(|p| \, |q|)$ time for $\alpha \in \mathbb{R}, \alpha \neq 1$ if p and q are both deterministic and compatible.*

Proof. A circuit pipeline for Rényi's α-divergence involves first computing $r = p^{\alpha}$ and $s = q^{1-\alpha}$, then $t = r \cdot s$ and finally integrate it.[5] Therefore we require t to be a smooth and decomposable circuit (Prop. 2.1), which in turn requires r and s to be compatible (Thm. 3.2). To conclude the proof, we need to compute two compatible circuits r and s in polytime, which can be done according to Thm. 3.5 or Thm. 3.3 depending on the value of α. As these theorems state, p^{α} and $q^{1-\alpha}$ will be compatible with p and q , respectively, with sizes $\mathcal{O}(|p|^{\alpha})$ and $\mathcal{O}(|q|)$ for a natural power α or $\mathcal{O}(|p|)$ and $\mathcal{O}(|q|)$ for a real-valued α. As such, t could be computed in $\mathcal{O}(|p|^{\alpha} \, |q|)$ time for $\alpha \in \mathbb{N}$ or $\mathcal{O}(|p| \, |q|)$ for $\alpha \in \mathbb{R}$ (Thm. 3.2). □

We leave the formal theorems and proofs for the other queries listed in Tab. 2 to Sec. C in the Appendix for space constraints. We remark again that our technique can be used beyond this query list and *can be applied to any complex query that involves a pipeline comprising the operations we discussed in Sec. 3 and culminating in an integration*.

Shannon entropy Smooth, decomposable and deterministic PCs enable the exact computation of Shannon entropy (Thm. C.2) and this tractability result translates to bounded-treewidth PGMs such as Chow-Liu trees and polytrees as they are special cases (Sec. 2). Our framework provides a more succinct tractability proof for the computation of Shannon entropy derived by Shih and Ermon [44], which we complete by demonstrating in Thm. C.2 that it is coNP-hard for non-deterministic PCs.

Rényi entropy For non-deterministic PCs we can employ the tractable computation of Rényi entropy of order $\alpha \in \mathbb{N}$ [40], which recovers Shannon Entropy for $\alpha \to 1$. As the logarithm is taken after integration of the power circuit, the tractability and hardness follow directly from those of the power operation (Thm. 3.3 and 3.5).

[4]Several alternative formulations of α-divergences can be found in the literature such as Amari's [31] and Tsallis's [34] divergences. However, as they share the same core operations—real powers and products of circuits—our results easily extend to them as well.

[5]Note that all the operations outside integration are tractable, therefore we can skip them.

Table 3: ***Efficient algorithms for several query classes quickly distilled using our compositional atlas.*** Times in seconds to compute the Shannon entropy (ENT), cross-entropy (XENT), Kullback-Leibler divergence (KLD), Alpha divergence (AlphaDiv) for $\alpha = 1.5$, Rényi entropy (RényiEnt), and Cauchy-Schwarz divergence (CSDiv) over the circuits learned from 7 real-world datasets (complete results for 10 datasets in Tab. 5) by using algorithms distilled by our pipelines and comparing them to the highly-optimized implementations of the ENT [44] and KLD [28] algorithms available in Juice.jl [12]. No tractable implementation was available for XENT, AlphaDiv, RényiEnt, and CSDiv.

DATASET	ENT		KLD		XENT		ALPHADIV		RÉNYIENT		CSDIV	
	OURS	JUICE	OURS	JUICE	OURS	JUICE	OURS	JUICE	OURS	JUICE	OURS	JUICE
KDD	0.157	0.001	3.154	0.790	2.180	-	0.885	-	0.016	-	1.136	-
PLANTS	0.679	0.005	3.983	3.909	3.739	-	1.160	-	0.088	-	1.572	-
AUDIO	0.406	0.003	2.736	1.681	1.873	-	0.537	-	0.029	-	0.771	-
JESTER	0.764	0.003	1.019	0.432	0.805	-	0.351	-	0.024	-	0.476	-
NETFLIX	0.106	0.002	0.352	0.175	0.264	-	0.100	-	0.017	-	0.201	-
DNA	4.365	0.027	64.664	220.377	52.997	-	15.609	-	0.255	-	22.901	-
AD	0.193	0.007	0.346	0.046	0.281	-	0.151	-	0.031	-	0.207	-

Cross entropy As hinted by the presence of a logarithm, the cross entropy is #P-hard to compute without determinism, even for compatible PCs (Thm. C.1). Nevertheless, given our atlas the cross entropy can be tractably computed in $\mathcal{O}(|p|\,|q|)$ if p and q are deterministic and compatible.

Mutual information Let a joint distribution $p(\mathbf{X}, \mathbf{Y})$ and its marginals $p(\mathbf{X})$ and $p(\mathbf{Y})$ be represented as PCs. Then the mutual information (MI) over these three PCs can be computed via a pipeline involving product, quotient, and log operators and it is tractable if all circuits are compatible and deterministic. On the other hand, if the marginal distributions cannot be represented as compact deterministic PCs, we prove it to be coNP-hard (Thm. C.3).

Divergences Liang and Van den Broeck [28] proposed an efficient algorithm to compute the KLD tailored for PSDDs. This has been the only tractable divergence available for PCs so far. We greatly extend this panorama with our atlas by introducing Rényi's α-divergences which generalize several other divergences such as the KLD when $\alpha \to 1$, Hellinger's squared divergence when $\alpha = 2^{-1}$, and the $\mathcal{X}^2$-divergence when $\alpha = 2$ [16]. As Thm. 4.1 states, they are tractable for compatible and deterministic PCs, as is the Itakura-Saito divergence [53] (Thm. C.8). For non-deterministic PCs, we characterize the tractability of the squared loss and the Cauchy-Schwarz divergence [19]. The latter has applications in mixture models for approximate inference [46] and has been derived in closed-form only for mixtures of simple parametric forms like Gaussians [22], Weibull and Rayligh distributions [33]. Our results generalize them to deep mixture models [38].

Expectation queries Among other complex queries that can be abstracted into the general form of an expectation of a circuit f w.r.t. a PC p, i.e., $\mathbb{E}_{\boldsymbol{x}\sim p(\mathbf{X})}\left[f(\boldsymbol{x})\right]$, there are the moments of distributions, such as means and variances. They can be efficiently computed for any smooth and decomposable PC, as f is an omni-compatible circuit (Prop. D.1). This result generalizes the moment computation for simple models such as GMMs and HMMs as they can be encoded as smooth and decomposable PCs (Sec. 2). If f is the indicator function of a logical formula, the expectation computes its probability w.r.t. the distribution p. Choi et al. [4] proposed an algorithm tailored to formulas f over binary variables, encoded as SDDs [14] w.r.t. distributions that are PSDDs. We generalize this result to mixed continuous-discrete distributions encoded as structured-decomposable PCs that are not necessarily deterministic and to logical formulas in the language of satisfiability modulo theories [2] over linear arithmetics with univariate literals (Prop. D.2). Lastly, if f encodes constraints over the output distribution of a deep network we retrieve the *semantic loss* [54]. If f encodes a classifier or a regressor, then $\mathbb{E}_p[f]$ refers to computing its expected predictions w.r.t. p [24]. Our results generalize the results reported in Van den Broeck et al. [47] such as computing the expectations of decision trees and their ensembles [25] (cf. Prop. D.3) as well as those of *deep regression circuits* [23].[6]

5 Experiments

We prototyped the tractable operators defined in Sec. 3 as subroutines in Julia in the Juice.jl framework [12] to showcase how our modular atlas can practically and quickly help implement tractable

[6]Despite the name, regression circuits do not conform to our definition of circuits in Def. 2.1. Nevertheless, we can translate them to our format in polytime as we illustrate in Alg. 8 in the Appendix.

algorithms for novel query classes. [7] To this extent, we distilled algorithms for the Shannon, Rényi, and cross entropies and for the KL, Alpha, and Cauchy-Schwarz divergences. We then ran them on deterministic and structured-decomposable circuits learned as in Dang et al. [11] from 20 publicly available real-world benchmark datasets [29, 48].

Tab. 3 shows the time taken to build and execute the pipelines on a subset of the datasets, while Tab. 4 and Tab. 5 in Sec. E in the Appendix report all the intermediate circuit sizes in their respective pipelines and times for all datasets. First, the intermediate circuits created in the pipeline do not blow up in size: as predicted by our theoretical analysis, the size of the logarithm circuit grows by a linear factor ($\sim$3–4x). Moreover, the size of the product circuit $p \cdot q$ is only slightly larger than $\max(|p|, |q|)$ when p and q are deterministic, much smaller than the theoretical bound of $\mathcal{O}(|p|\,|q|)$.

In terms of execution time, our algorithms run in less than a second for most circuits and peak at slightly more than one minute to compute a pipeline of the KLD, whose output circuit has more than 3 million edges on the DNA dataset (Tab. 4). A custom and highly optimized implementation of the KLD for PSDDs by Liang and Van den Broeck [28] runs up to ten times faster on smaller circuits but surprisingly takes $\sim$220 seconds for DNA, highlighting that our compositional atlas is a promising way to distill tractable algorithms. We emphasize that the aim of these experiments is not to distill the fastest algorithm for a query class, but to demonstrate that *our compositional framework empowers practitioners to quickly distill new tractable algorithms for queries that were not available before*, such as the Rényi entropy and α and Cauchy-Schwarz divergences.

6 Discussion and Conclusions

This work introduced a unified framework to reason about tractable model classes for complex queries composed of simpler operations. This rich atlas of operators can be used to solve many queries common in probabilistic ML and AI as well as novel inference scenarios.

Darwiche and Marquis [15] is the work most closely related to ours: they define operators over *logical circuits*, encoding Boolean functions as computational graphs with AND and OR gates, for which structural properties analogous to those discussed in Sec. 2 can be defined. Our results generalize their work on logical tractable operators such as disjunctions and conjunctions—the analogous to our (deterministic) sums and products—while also extending it to powers, logarithms and exponentials as well as complex queries such as divergences, which have no direct counterpart in the logical domain. Algorithms to tractably multiply two probabilistic models have been proposed for probabilistic decision graphs (PDGs) first [17] and PSDDs later [43]. Despite the different syntax, both model classes can be encoded as structured-decomposable and deterministic circuits in our language [8]. Historically, algorithms for the tractable product of PDGs and PSDDs relied on a special case of compatibility, when two structured-decomposable models exactly share the same hierarchical scope partitioning in terms of special graphical representations such as pseudo forests [17, 18] and vtrees [37]. They also implicitly entangle compatibility with determinism. As we showed in Thm. 3.2, compatibility is sufficient for tractable multiplication, which interestingly, has been noted in the context of logical circuits [37]. As discussed in the previous section, many algorithms tailored for PSDDs [4, 43, 23] can therefore be generalized to *non-deterministic* distributions in our framework.

Our property-driven analysis closes many open questions about the tractability and hardness of queries for several model classes that are special cases of circuits. At the same time, our hardness results might limit its general applicability by restricting tractable inference of some query classes to only certain inputs (Sec. 3). However, we see this as an opportunity: now that we can analyze which operators in a pipeline are tractable or not, we could substitute the latter by an approximate inference algorithm distilled from the pipeline. Lastly, we plan to extend our analysis to other queries involving not only integration but also maximization—that is, understanding what are the operators that make MAP inference over probabilistic circuits or optimization over general circuits tractable.

Acknowledgements AV would like to thank Yujia Shen and Arthur Choi for insightful discussions about the product algorithm for PSDDs and Zhe Zeng for proofreading an initial version of this work. This work is partially supported by NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, a Sloan Fellowship, and gifts from Intel and Facebook Research. The research of ST was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

[7] Code publicly available at `https://github.com/UCLA-StarAI/circuit-ops-atlas`.

References

[1] Francis R Bach and Michael I Jordan. Thin junction trees. In *NIPS*, volume 14, pages 569–576, 2001.

[2] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[3] Arthur Choi, Doga Kisa, and Adnan Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 121–132. Springer, 2013.

[4] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2015, pages 2861–2868, 2015.

[5] YooJung Choi, Adnan Darwiche, and Guy Van den Broeck. Optimal feature selection for decision robustness in bayesian networks. In *IJCAI*, pages 1554–1560, 2017.

[6] YooJung Choi, Meihua Dang, and Guy Van den Broeck. Group fairness by probabilistic modeling with latent fair decisions. *arXiv preprint arXiv:2009.09031*, 2020.

[7] YooJung Choi, Golnoosh Farnadi, Behrouz Babaki, and Guy Van den Broeck. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10077–10084, 2020.

[8] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020.

[9] C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[10] Alvaro H. C. Correia, Robert Peharz, and Cassio P. de Campos. Joints in random forests. In *NeurIPS*, 2020.

[11] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *PGM*, Proceedings of Machine Learning Research, 2020.

[12] Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, Feb 2021.

[13] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

[14] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[15] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[16] Alison L Gibbs and Francis Edward Su. On choosing and bounding probability metrics. *International statistical review*, 70(3):419–435, 2002.

[17] 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.

[18] Manfred Jaeger, Jens D Nielsen, and Tomi Silander. Learning probabilistic decision graphs. *International Journal of Approximate Reasoning*, 42(1-2):84–100, 2006.

[19] Robert Jenssen, Jose C Principe, Deniz Erdogmus, and Torbjørn Eltoft. The cauchy–schwarz divergence and parzen windowing: Connections to graph theory and mercer kernels. *Journalof the Franklin Institute*, 343(6):614–629, 2006.

[20] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.

[21] Wolfgang B Jurkat. On cauchy's functional equation. *Proceedings of the American Mathematical Society*, 16(4):683–686, 1965.

[22] Kittipat Kampa, Erion Hasanbelliu, and Jose C Principe. Closed-form cauchy-schwarz pdf divergence for mixture of gaussians. In *The 2011 International Joint Conference on Neural Networks*, pages 2578–2585. IEEE, 2011.

[23] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems*, pages 11169–11180, 2019.

[24] Pasha Khosravi, Yitao Liang, YooJung Choi, and Guy Van den Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. In *IJCAI*, pages 2716–2724, 2019.

[25] Pasha Khosravi, Antonio Vergari, YooJung Choi, Yitao Liang, and Guy Van den Broeck. Handling missing data in decision trees: A probabilistic approach. In *Proceedings of The Art of Learning with Missing Values, Workshop at ICML*, 2020.

[26] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th international conference on principles of knowledge representation and reasoning (KR)*, pages 1–10, 2014.

[27] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[28] Yitao Liang and Guy Van den Broeck. Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, August 2017.

[29] Daniel Lowd and Jesse Davis. Learning markov network structure with decision trees. In *2010 IEEE International Conference on Data Mining*, pages 334–343. IEEE, 2010.

[30] Geoffrey J McLachlan, Sharon X Lee, and Suren I Rathnayake. Finite mixture models. *Annual review of statistics and its application*, 6:355–378, 2019.

[31] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *UAI*, pages 362–369. Morgan Kaufmann, 2001.

[32] Serafín Moral, Rafael Rumí, and Antonio Salmerón. Mixtures of truncated exponentials in hybrid bayesian networks. In *ECSQARU*, volume 2143 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 2001.

[33] Frank Nielsen. Closed-form information-theoretic divergences for statistical mixtures. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1723–1726. IEEE, 2012.

[34] Manfred Opper, Ole Winther, and Michael J Jordan. Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6(12), 2005.

[35] Umut Oztok, Arthur Choi, and Adnan Darwiche. Solving PP^{PP}-complete problems using knowledge compilation. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 94–103, 2016.

[36] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *LTPM workshop*, volume 32, 2014.

[37] Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 517–522, 2008.

[38] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[39] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[40] Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

[41] Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In *International Conference on Machine Learning*, pages 710–718. PMLR, 2014.

[42] Prasanna K Sahoo and Palaniappan Kannappan. *Introduction to functional equations*. CRC Press, 2011.

[43] Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Advances in Neural Information Processing Systems*, pages 3936–3944, 2016.

[44] Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. In *NeurIPS*, 2020.

[45] Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *NeurIPS*, pages 11412–11422, 2019.

[46] Linh Tran, Maja Pantic, and Marc Peter Deisenroth. Cauchy-schwarz regularized autoencoder. *arXiv preprint arXiv:2101.02149*, 2021.

[47] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of shap explanations. In *Proceedings of the 35th Conference on Artificial Intelligence (AAAI)*, 2021.

[48] Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[49] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

[50] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and understanding sum-product networks. *Machine Learning*, 108(4):551–573, 2019.

[51] Antonio Vergari, Nicola Di Mauro, and Guy Van den Broeck. Tractable probabilistic models: Representations, algorithms, learning, and applications. *Tutorial at UAI*, 2019.

[52] Eric Wang, Pasha Khosravi, and Guy Van den Broeck. Probabilistic sufficient explanations. *arXiv preprint arXiv:2105.10118*, 2021.

[53] Bo Wei and Jerry D Gibson. Comparison of distance measures in discrete spectral modeling. Master's thesis, Citeseer, 2001.

[54] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5498–5507. PMLR, 2018.

[55] Zhe Zeng, Paolo Morettin, Fanqi Yan, Antonio Vergari, and Guy Van den Broeck. Scaling up hybrid probabilistic inference with logical and arithmetic constraints via message passing. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 10990–11000. PMLR, 2020.

Algorithm 1 SUPPORT(p, cache)

Input: a smooth, deterministic, and decomposable circuit p over variables $\mathbf{X}$ and a cache for memorization
Output: a smooth, deterministic, and decomposable circuit s over $\mathbf{X}$ encoding $s(\boldsymbol{x}) = [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$
if $p \in \mathsf{cache}$ **then return** $\mathsf{cache}(p)$
if p is an input unit **then** $s \leftarrow$ INPUT$([\![\boldsymbol{x} \in \mathsf{supp}(p)]\!], \phi(p))$
else if p is a sum unit **then** $s \leftarrow$ SUM$(\{\text{SUPPORT}(p_i, \mathsf{cache})\}_{i=1}^{|\mathsf{in}(p)|}, \{1\}_{i=1}^{|\mathsf{in}(p)|})$
else if p is a product unit **then** $s \leftarrow$ PRODUCT$(\{\text{SUPPORT}(p_i, \mathsf{cache})\}_{i=1}^{|\mathsf{in}(p)|})$
$\mathsf{cache}(p) \leftarrow s$
return s

A Useful Sub-Routines

This section introduces the algorithmic construction of gadget circuits that will be adopted in our proofs of tractability as well as hardness. We start by introducing three primitive functions for constructing circuits—INPUT, SUM, and PRODUCT.

• INPUT$(l_p, \phi(p))$ constructs an input unit p that encodes a parameterized function l_p over variables $\phi(p)$. For example, INPUT$([\![X = \text{True}]\!], X)$ and INPUT$([\![X = \text{False}]\!], X)$ represent the positive and negative literals of a Boolean variable X, respectively. On the other hand, INPUT$(\mathcal{N}(\mu, \sigma), X)$ defines a Gaussian pdf with mean μ and standard deviation σ over variable X as an input function.

• SUM$(\{p_i\}_{i=1}^k, \{\theta_i\}_{i=1}^k)$ constructs a sum unit that represents the weighted combination of k circuit units $\{p_i\}_{i=1}^k$ encoded as an ordered set w.r.t. the correspondingly ordered weights $\{\theta_i\}_{i=1}^k$.

• PRODUCT$(\{p_i\}_{i=1}^k)$ builds a product unit that encodes the product of k circuit units $\{p_i\}_{i=1}^k$.

A.1 Support circuit of a deterministic circuit

Given a smooth, decomposable, and deterministic circuit $p(\mathbf{X})$, its support circuit $s(\mathbf{X})$ is a smooth, decomposable, and deterministic circuit that evaluates 1 iff the input $\boldsymbol{x}$ is in the support of p (i.e., $\boldsymbol{x} \in \mathsf{supp}(p)$) and otherwise evaluates 0, as defined below.

Definition A.1 (Support circuit)**.** Let p be a smooth, decomposable, and deterministic PC over variables $\mathbf{X}$. Its support circuit is the circuit s that computes $s(\boldsymbol{x}) = [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$, obtained by replacing every sum parameter of p by 1 and every input distribution l by the function $[\![\boldsymbol{x} \in \mathsf{supp}(l)]\!]$.

A construction algorithm for the support circuit is provided in Alg. 1. This algorithm will later be useful in defining some circuit operations such as the logarithm.

A.2 Circuits encoding uniform distributions

We can build a deterministic and omni-compatible PC that encodes a (possibly unnormalized) uniform distribution over binary variables $\mathbf{X} = \{X_1, \ldots, X_n\}$: i.e., $p(\boldsymbol{x}) = c$ for a constant $c \in \mathbb{R}_+$ for all $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$. Specifically, p can be defined as a single sum unit with weight c that receives input from a product unit over n univariate input distribution units that always output 1 for all values $\mathsf{val}(X_i)$. This construction is summarized in Alg. 2. It is a key component in the algorithms for many tractable circuit transformations/queries as well as in several hardness proofs.

A.3 A circuit representation of the #3SAT problem

We define a circuit representation of the #3SAT problem, following the construction in Khosravi et al. [23]. Specifically, we represent each instance in the #3SAT problem as two poly-sized structured-decomposable and deterministic circuits p_β and p_γ, such that the partition function of their product equals the solution of the original #3SAT problem.

#3SAT is defined as follows: given a set of n boolean variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and a CNF that contains m clauses $\{c_1, \ldots, c_m\}$ (each clause contains exactly 3 literals), count the number of satisfiable worlds in $\mathsf{val}(\mathbf{X})$.

Algorithm 2 UNIFORMCIRCUIT($\mathbf{X}, c$)

Input: a set of variables $\mathbf{X}$ and constant $c \in \mathbb{R}_+$.
Output: a deterministic and omni-compatible PC encoding an unnormalized uniform distribution over $\mathbf{X}$.
$n \leftarrow \{\}$
for $i = 1$ **to** $|\mathbf{X}|$ **do**
  $m \leftarrow \{\}$
  for x_i **in** $\mathsf{val}(X_i)$ **do**
    $m \leftarrow m \cup \{\textsc{Input}([\![X_i = x_i]\!], X_i)\}$
  $n \leftarrow n \cup \{\textsc{Sum}(m, \{1\}_{j=1}^{|\mathsf{val}(X_i)|})\}$
return $\textsc{Sum}(\{\textsc{Product}(n)\}, \{c\})$

For every variable X_i in clause c_j, we introduce an auxiliary variable X_{ij}. Intuitively, $\{X_{ij}\}_{j=1}^m$ are copies of the variable X_i, one for each clause. Therefore, for any i, $\{X_{ij}\}_{j=1}^m$ share the same value (i.e., true or false), which can be represented by the following formula β:

$$\beta \equiv \bigwedge_{i=1}^{n} (X_{i1} \Leftrightarrow X_{i2} \Leftrightarrow \cdots \Leftrightarrow X_{im}).$$

Then we can encode the original CNF in the following formula γ by substituting X_i with the respective X_{ij} in each clause:

$$\gamma \equiv \bigwedge_{j=1}^{m} \bigvee_{i:X_i \in \phi(c_j)} l(X_{ij}),$$

where $\phi(c)$ denotes the variable scope of clause c, and $l(X_{ij})$ denotes the literal of X_i in clause c_j. Since β restricts the variables $\{X_{ij}\}_{j=1}^m$ to have the same value, the model count of $\beta \wedge \gamma$ is equal to the model count of the original CNF.

We are left to show that both β and γ can be compiled into a poly-sized structured-decomposable and deterministic circuit. We start from compiling β into a circuit p_β. Note that for each i, $(X_{i1} \Leftrightarrow \cdots \Leftrightarrow X_{im})$ has exactly two satisfiable variable assignments (i.e., all true or all false), it can be compiled as a sum unit a_i over two product units b_{i1} and b_{i2} (both weights of a are set to 1), where b_{i1} takes inputs from the positive literals $\{X_{i1}, \ldots, X_{im}\}$ and b_{i2} from the negative literals $\{\neg X_{i1}, \ldots, \neg X_{im}\}$. Then p_β is represented by a product unit over $\{a_1, \ldots, a_n\}$. Note that by definition this p_β circuit is structured-decomposable and deterministic.

We proceed to compile γ into a polysized structured-decomposable and deterministic circuit p_γ. Note that in #3SAT, each clause c_j contains 3 literals. Therefore, for any $j \in \{1, \ldots, m\}$, $\bigvee_{X_i \in \phi(c_j)} l(X_{ij})$ has exactly 7 models w.r.t. the variable scope $\phi(c_j)$. Hence, we compile $\bigvee_{X_i \in \phi(c_j)} l(X_{ij})$ into a circuit d_j, which is a sum unit with 7 inputs $\{e_{j1}, \ldots, e_{j7}\}$. Each e_{jh} is constructed as a product unit over variables $\{X_{1j}, \ldots, X_{nj}\}$ that represents the h-th model of clause c_j. More formally, we have $e_{jh} \leftarrow \textsc{Product}(\{g_{ijh}\}_{i=1}^n)$, where g_{ijh} is a sum unit over literals X_{ij} and $\neg X_{ij}$ (with both weights being 1) if $i \notin \phi(c_j)$ and otherwise g_{ijh} is the literal unit corresponds to the h-th model of clause c_j. The circuit p_γ representing the formula γ is constructed by a product unit with inputs $\{d_j\}_{j=1}^m$. By construction this circuit is also structured-decomposable and deterministic.

B Circuit Operations

This section formally presents the tractability and hardness results w.r.t. circuit operations summarized in Tab. 1—sums, products, quotients, powers, logarithms, and exponentials. For each circuit operation, we provide both its proof of tractability by constructing a polytime algorithm given sufficient structural constraints and novel hardness results that identify necessary structural constraints for the operation to yield a decomposable circuit as output.

Throughout this paper, we will show hardness of operations to output a decomposable circuit by proving hardness of computing the partition function of the output of the operation. This follows from

the fact that we can smooth and integrate a decomposable circuit in polytime (Prop. 2.1), thereby making the former problem at least as hard as the latter.

For the tractability theorems, we will assume that the operation referenced by the theorem is tractable over input units of circuit or pairs of compatible input units. For example, for Thm. 3.2 we assume tractable product of input units sharing the same scope and for Thm. 3.5 we assume that the powers of the input units can be tractably represented as a single new unit. Note that this is generally easy to realize for simple parametric forms e.g., multivariate Gaussians and for univariate distributions, unless specified otherwise.

Moreover, in the following results, we will adopt a more general definition of compatibility that can be applied to circuits with different variable scopes, which is often useful in practice. Formally, consider two circuits p and q with variable scope $\mathbf{Z}$ and $\mathbf{Y}$. Analogous to Def. 2.5, we say that p and q are compatible over variables $\mathbf{X} = \mathbf{Z} \cap \mathbf{Y}$ if (1) they are smooth and decomposable and (2) any pair of product units $n \in p$ and $m \in q$ with the same overlapping scope with $\mathbf{X}$ can be rearranged into mutually compatible binary products. Note that since our tractability results hold for this extended definition of compatibility, they are also satisfied under Def. 2.5.

B.1 Sum of Circuits

The hardness of the sum of two circuits to yield a deterministic circuit has been proven by Shen et al. [43] in the context of arithmetic circuits (ACs) [15]. ACs can be readily turned into circuits over binary variables according to our definition by translating their input parameters into sum parameters as done in Rooshenas and Lowd [41].

A sum of circuits will preserve decomposability and related properties as the next proposition details.

Proposition B.1 (Closure of sum of circuits)**.** *Let $p(\mathbf{Z})$ and $q(\mathbf{Y})$ be decomposable circuits. Then their sum circuit $s(\mathbf{Z} \cup \mathbf{Y}) = \theta_1 \cdot p(\mathbf{Z}) + \theta_2 \cdot q(\mathbf{Y})$ for two reals $\theta_1, \theta_2 \in \mathbb{R}$ is decomposable. If p and q are structured-decomposable and compatible, then s is structured-decomposable and compatible with both p and q. Lastly, if both inputs are also smooth, s can be smoothed in polytime.*

Proof. If p and q are decomposable, s is also decomposable by definition (no new product unit is introduced). If they are also structured-decomposable and compatible, s would be structured-decomposable and compatible with p and q as well, as summation does not affect their hierarchical scope partitioning. Note that if one input is decomposable and the other omni-compatible, then s would only be decomposable.

If $\mathbf{Z} = \mathbf{Y}$ then s is smooth; otherwise we can smooth it in polytime [13, 45], by realizing the circuit

$$s(\boldsymbol{x}) = \theta_1 \cdot p(\boldsymbol{z}) \cdot [\![q(\boldsymbol{x}|_{\mathbf{Y} \setminus \mathbf{Z}}) \neq 0]\!] + \ \theta_2 \cdot q(\boldsymbol{y}) \cdot [\![p(\boldsymbol{x}|_{\mathbf{Z} \setminus \mathbf{Y}}) \neq 0]\!]$$

where $[\![q(\boldsymbol{x}|_{\mathbf{Y} \setminus \mathbf{Z}}) \neq 0]\!]$ (resp. $[\![p(\boldsymbol{x}|_{\mathbf{Z} \setminus \mathbf{Y}}) \neq 0]\!]$) can be encoded as an input distribution over variables $\mathbf{Y} \setminus \mathbf{Z}$ (resp.$\mathbf{Z} \setminus \mathbf{Y}$). Note that if the supports of $p(\mathbf{Z} \setminus \mathbf{Y})$ and $q(\mathbf{Y} \setminus \mathbf{Z})$ are not bounded, then integrals over them would be unbounded as well. □

B.2 Product of Circuits

Theorem 3.1 (Hardness of product)**.** Let p and q be two structured-decomposable and deterministic circuits over variables $\mathbf{X}$. Computing their product $m(\mathbf{X}) = p(\mathbf{X}) \cdot q(\mathbf{X})$ as a decomposable circuit is #P-hard.[8]

Proof. As noted earlier, we will prove hardness of computing the product by showing hardness of computing the partition function of a product of two circuits. In particular, let p and q be two structured-decomposable and deterministic circuits over binary variables $\mathbf{X}$. Then, computing the following quantity is #P-hard:

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) \cdot q(\boldsymbol{x}). \tag{MULPC}$$

[8]Note that this implies that product of decomposable circuits is also #P-hard, as decomposability is a weaker condition than structured-decomposability. The hardness results throughout this paper translate directly when input properties are relaxed.

The following proof is adapted from the proof of Thm. 2 in Khosravi et al. [23]. We reduce the #3SAT problem defined in Sec. A.3, which is known to be #P-hard, to MULPC. Recall that p_β and p_γ, as constructed in Sec. A.3, are structured-decomposable and deterministic; additionally, the partition function of $p_\beta \cdot p_\gamma$ is the solution of the corresponding #3SAT problem. In other words, computing MULPC of two structured-decomposable and deterministic circuits p_β and p_γ exactly solves the original #3SAT problem. Therefore, computing the product of two structured-decomposable and deterministic circuits is #P-hard. □

Theorem 3.2 (Tractable product of circuits)**.** Let $p(\mathbf{Z})$ and $q(\mathbf{Y})$ be two compatible circuits over variables $\mathbf{X} = \mathbf{Z} \cap \mathbf{Y}$. Then, computing their product $m(\mathbf{X}) = p(\mathbf{Z}) \cdot q(\mathbf{Y})$ as a decomposable circuit can be done in $\mathcal{O}(|p|\,|q|)$ time. If both p and q are also deterministic, then so is m, moreover if p and q are structured-decomposable then m is compatible with p (and q) over $\mathbf{X}$.

Proof. The proof proceeds by showing that computing the product of (i) two smooth and compatible sum units p and q and (ii) two smooth and compatible product units p and q given the product circuits w.r.t. pairs of child units from p and q (i.e., $\forall r \in \mathsf{in}(p)\, s \in \mathsf{in}(q), (r \cdot s)(\mathbf{X})$) takes time $\mathcal{O}(|\mathsf{in}(p)|\,|\mathsf{in}(q)|)$. Then, by recursion, the overall time complexity is $\mathcal{O}(|p|\,|q|)$. Alg. 3 illustrates the overall process in detail.

If p and q are two sum units defined as $p(\boldsymbol{x}) = \sum_{i \in \mathsf{in}(p)} \theta_i p_i(\boldsymbol{x})$ and $q(\boldsymbol{x}) = \sum_{j \in \mathsf{in}(q)} \theta'_j q_j(\boldsymbol{x})$, respectively. Then, their product $m(\boldsymbol{x})$ can be broken down to the weighted sum of $|\mathsf{in}(p)| \cdot |\mathsf{in}(q)|$ circuits that represent the products of pairs of their inputs:

$$m(\boldsymbol{x}) = \left(\sum_{i \in \mathsf{in}(p)} \theta_i p_i(\boldsymbol{x})\right)\left(\sum_{j \in \mathsf{in}(q)} \theta'_j q_j(\boldsymbol{x})\right) = \sum_{i \in \mathsf{in}(p)} \sum_{j \in \mathsf{in}(q)} \theta_i \theta'_j (p_i q_j)(\boldsymbol{x}).$$

Note that this Cartesian product of units is a deterministic sum unit if both p and q were deterministic sum units, as $\mathsf{supp}(p_i q_j) = \mathsf{supp}(p_i) \cap \mathsf{supp}(q_j)$ are disjoint for different i, j.

If p and q are two product units defined as $p(\mathbf{X}) = p_1(\mathbf{X}_1)p_2(\mathbf{X}_2)$ and $q(\mathbf{X}) = q_1(\mathbf{X}_1)q_2(\mathbf{X}_2)$, respectively. Then, their product $m(\boldsymbol{x})$ can be constructed recursively from the product of their inputs:

$$m(\boldsymbol{x}) = p_1(\boldsymbol{x}_1)p_2(\boldsymbol{x}_2) \cdot q_1(\boldsymbol{x}_1)q_2(\boldsymbol{x}_2) = p_1(\boldsymbol{x}_1)q_1(\boldsymbol{x}_1) \cdot p_2(\boldsymbol{x}_2)q_2(\boldsymbol{x}_2) = (p_1 q_1)(\boldsymbol{x}_1) \cdot (p_2 q_2)(\boldsymbol{x}_2).$$

Note that by this construction m retains the same scope partitioning of p and q, hence if they were structured-decomposable, m will be structured-decomposable and compatible with p and q. □

Possessing additional structural constrains can lead to sparser output circuits as well as efficient algorithms to construct them. First, if one among p and q is omni-compatible, it suffices that the other is just decomposable to obtain a tractable product, whose size this time is going to be linear in the size of the decomposable circuit.

Corollary B.1. *Let p be a smooth and decomposable circuit over $\mathbf{X}$ and q an omni-compatible circuit over $\mathbf{X}$ comprising a sum unit with k inputs, hence its size is $k\,|\mathbf{X}|$. Then, $m(\mathbf{X}) = p(\mathbf{X})q(\mathbf{X})$ is a smooth and decomposable circuit constructed in $\mathcal{O}(k\,|p|)$ time.*

Second, if p and q have inputs with restricted supports, their product is going to be sparse, i.e., only a subset of their inputs is going to yield a circuit that does not constantly output zero. Note that in Alg. 3 we can check in polytime if the supports of two units to be multiplied are overlapping by a depth-first search (realized with a Boolean indicator s in Alg. 3), thanks to decomposability. Therefore, for two compatible sum units p and q we will effectively build a number of units that is

$$\mathcal{O}(|\{(p_i, q_j) | p_i \in \mathsf{in}(p), q_i \in \mathsf{in}(q), \mathsf{supp}(p_i) \cap \mathsf{supp}(q_j) \neq \emptyset\}|).$$

In practice, this sparsifying effect will be more prominent when both p and q are deterministic. This is because having disjoint supports is required for deterministic circuits. This "decimation" of product units will be maximum if p and q partition the support in the very same way, for instance when we have $p = q$, i.e., we are multiplying one circuit with itself, or we are dealing with a logarithmic circuit (cf. Sec. B.5). In such a case, we can omit the depth-first check for overlapping supports of the product units participating in the product of a sum unit. If both p and q have an identifier for their supports, we can simply check for equality of their identifiers. This property and algorithmic insight will be key when computing powers of a deterministic circuit and its entropies (cf. Sec. C.2), as it would suffice the input circuit p to be decomposable (cf. Sec. 3) to obtain a linear time complexity.

Algorithm 3 MULTIPLY(p, q, cache)

Input: two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ that are compatible over $\mathbf{X} = \mathbf{Z} \cap \mathbf{Y}$ and a cache for memoization
Output: their product circuit $m(\mathbf{Z} \cup \mathbf{Y}) = p(\mathbf{Z})q(\mathbf{Y})$
if $(p, q) \in \mathsf{cache}$ **then return** $\mathsf{cache}(p, q)$
if $\phi(p) \cap \phi(q) = \emptyset$ **then**
  $m \leftarrow \textsc{Product}(\{p, q\})$; $s \leftarrow \text{True}$
else if p, q are input units **then**
  $m \leftarrow \textsc{Input}(p(\mathbf{Z}) \cdot q(\mathbf{Y}), \mathbf{Z} \cup \mathbf{Y})$
  $s \leftarrow [\![\mathsf{supp}(p(\mathbf{X})) \cap \mathsf{supp}(q(\mathbf{X})) \neq \emptyset]\!]$
else if p is an input unit **then**
  $n \leftarrow \{\}; s \leftarrow \text{False}$ $//q(\mathbf{Y}) = \sum_j \theta'_j q_j(\mathbf{Y})$
  for $j = 1$ **to** $|\mathsf{in}(q)|$ **do**
    $n', s' \leftarrow \textsc{multiply}(p, q_j, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\}$; $s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, \{\theta'_j\}_{j=1}^{|\mathsf{in}(q)|})$ **else** $m \leftarrow null$
else if q is an input unit **then**
  $n \leftarrow \{\}; s \leftarrow \text{False}$ $//p(\mathbf{Z}) = \sum_i \theta_i p_i(\mathbf{Z})$
  for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
    $n', s' \leftarrow \textsc{multiply}(p_i, q, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\}$; $s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, \{\theta_i\}_{i=1}^{|\mathsf{in}(p)|})$ **else** $m \leftarrow null$
else if p, q are product units **then**
  $n \leftarrow \{\}; s \leftarrow \text{True}$
  $\{p_i, q_i\}_{i=1}^{k} \leftarrow \mathsf{sortPairsByScope}(p, q, \mathbf{X})$
  for $i = 1$ **to** k **do**
    $n', s' \leftarrow \textsc{multiply}(p_i, q_i, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\}$; $s \leftarrow s \wedge s'$
  if s **then** $m \leftarrow \textsc{Product}(n)$ **else** $m \leftarrow null$
else if p, q are sum units **then**
  $n \leftarrow \{\}$; $w \leftarrow \{\}$; $s \leftarrow \text{False}$
  for $i = 1$ **to** $|\mathsf{in}(p)|, j = 1$ **to** $|\mathsf{in}(q)|$ **do**
    $n', s' \leftarrow \textsc{multiply}(p_i, q_j, \mathsf{cache})$
    $n \leftarrow n \cup n'; w \leftarrow w \cup \{\theta_i \theta'_j\}; s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, w)$ **else** $m \leftarrow null$
$\mathsf{cache}(p, q) \leftarrow (m, s)$
return m, s

B.3 Power Function of Circuits

Theorem 3.3 (Natural powers)**.** If p is a structured-decomposable circuit, then for any $\alpha \in \mathbb{N}$, its power can be represented as a structured-decomposable circuit in $\mathcal{O}(|p|^{\alpha})$ time. Otherwise, if p is only smooth and decomposable, then computing $p^{\alpha}(\mathbf{X})$ as a decomposable circuit is #P-hard.

Proof. The proof for tractability easily follows by directly applying the product operation repeatedly.

We prove hardness for the special case of discrete variables, and by showing the hardness of computing the partition function of $p^2(\mathbf{X})$. In particular, let $\mathbf{X}$ be a collection of binary variables and let p be a smooth and decomposable circuit over $\mathbf{X}$, then computing the quantity

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^2(\boldsymbol{x}) \tag{POW2PC}$$

is #P-hard.

The proof builds a reduction from the #3SAT problem, which is known to be #P-hard. We employ the same setting of Sec. A.3, where a CNF over n Boolean variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and containing

Algorithm 4 SORTPAIRSBYSCOPE($p, q, \mathbf{X}$)

Input: two decomposable and compatible product units p and q, and a variable scope $\mathbf{X}$.
Output: Pairs of compatible sum units $\{(p_i, q_i)\}_{i=1}^{k}$.
children_$p \leftarrow \{p_i\}_{i=1}^{|\mathsf{in}(p)|}$, children_$q \leftarrow \{q_i\}_{i=1}^{|\mathsf{in}(q)|}$
pairs $\leftarrow \{\}$. // "pairs" stores circuit pairs with matched scope.
cmp_$p \leftarrow \{\{\}\}_{i=1}^{|\mathsf{in}(p)|}$, cmp_$q \leftarrow \{\{\}\}_{j=1}^{|\mathsf{in}(q)|}$.
// cmp_$p[i]$ (resp. cmp_$q[j]$) stores the children of q (resp. p) whose scopes are subsets of p_i's (resp. q_j's) scope.
for i = 1 **to** $|\mathsf{in}(p)|$ **do**
  for j = 1 **to** $|\mathsf{in}(q)|$ **do**
    if $\phi(p_i) \cap \mathbf{X} = \phi(q_j) \cap \mathbf{X}$ **then**
      pairs.$append((p_i, q_j))$
      children_p.$pop(p_i)$, children_q.$pop(q_j)$
    else if $\phi(p_i) \cap \mathbf{X} \subset \phi(q_j) \cap \mathbf{X}$ **then**
      cmp_$q[j]$.$append(p_i)$
      children_p.$pop(p_i)$, children_q.$pop(q_j)$
    else if $\phi(q_j) \cap \mathbf{X} \subset \phi(p_i) \cap \mathbf{X}$ **then**
      cmp_$p[i]$.$append(q_j)$
      children_p.$pop(p_i)$, children_q.$pop(q_j)$
for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
  if $len(\text{cmp\_}p[i]) \neq 0$ **then**
    $s \leftarrow$ SUM($\{$PRODUCT(cmp_$p[i]$)$\}, \{1\}$)
    pairs.$append((p_i, s))$
for $j = 1$ **to** $|\mathsf{in}(q)|$ **do**
  if $len(\text{cmp\_}q[j]) \neq 0$ **then**
    $r \leftarrow$ SUM($\{$PRODUCT(cmp_$q[j]$)$\}, \{1\}$)
    pairs.$append((r, q_j))$
for r, s **in** $zip(\text{children\_}p, \text{children\_}q)$ **do**
  pairs.$append((r, s))$
if $len(\text{children\_}p) > len(\text{children\_}q)$ **then**
  for $i = len(\text{children\_}q) + 1$ **to** $len(\text{children\_}p)$ **do**
    pairs.$append((\text{children\_}p[i], \text{children\_}q[1]))$
else if $len(\text{children\_}p) < len(\text{children\_}q)$ **then**
  for $j = len(\text{children\_}p) + 1$ **to** $len(\text{children\_}q)$ **do**
    pairs.$append((\text{children\_}p[1], \text{children\_}q[j]))$
return $pairs$

m clauses $\{c_1, \ldots, c_m\}$, each with exactly 3 literals, is encoded into two structured-decomposable and deterministic circuits p_β and p_γ over variables $\hat{\mathbf{X}} = \{X_{11}, \ldots, X_{1m}, \ldots, X_{n1}, \ldots, X_{nm}\}$.

Then, we construct circuit p_α as the sum of p_β and p_γ, i.e., $p_\alpha(\hat{\boldsymbol{x}}) := p_\beta(\hat{\boldsymbol{x}}) + p_\gamma(\hat{\boldsymbol{x}})$. By definition p_α is smooth and decomposable, but not structured-decomposable. We proceed to show that if we can represent $p_\alpha^2(\hat{\boldsymbol{x}})$ as a smooth and decomposable circuit in polytime, we could solve POW2PC and hence #3SAT. That would mean that computing POW2PC is #P-hard.

By definition, $p_\alpha^2(\hat{\boldsymbol{x}}) = (p_\beta(\hat{\boldsymbol{x}}) + p_\gamma(\hat{\boldsymbol{x}}))^2 = p_\beta^2(\hat{\boldsymbol{x}}) + p_\gamma^2(\hat{\boldsymbol{x}}) + 2p_\beta(\hat{\boldsymbol{x}}) \cdot p_\gamma(\hat{\boldsymbol{x}})$, and hence

$$\sum_{\hat{\boldsymbol{x}} \in \mathsf{val}(\hat{\mathbf{X}})} p_\alpha^2(\hat{\boldsymbol{x}}) = \sum_{\hat{\boldsymbol{x}} \in \mathsf{val}(\hat{\mathbf{X}})} p_\beta^2(\hat{\boldsymbol{x}}) + \sum_{\hat{\boldsymbol{x}} \in \mathsf{val}(\hat{\mathbf{X}})} p_\gamma^2(\hat{\boldsymbol{x}}) + \sum_{\hat{\boldsymbol{x}} \in \mathsf{val}(\hat{\mathbf{X}})} p_\beta(\hat{\boldsymbol{x}}) \cdot p_\gamma(\hat{\boldsymbol{x}}).$$

Since p_β and p_γ are both structured-decomposable and deterministic the first two summations over the squared circuits can be computed in time $\mathcal{O}(|p_\beta| + |p_\gamma|)$ (see Thm. 3.5). It follows that if we could efficiently solve POW2PC we could then solve the that third summation, i.e., $\sum_{\hat{\boldsymbol{x}} \in \mathsf{val}(\hat{\mathbf{X}})} p_\beta(\hat{\boldsymbol{x}}) \cdot p_\gamma(\hat{\boldsymbol{x}})$. However, since such a summation is the instance of MULPC between p_β and p_γ reduced from #3SAT (see Thm. 3.1), it would mean that we could solve #3SAT. We can conclude that computing POW2PC is #P-hard. □

Theorem B.1 (Hardness of natural power of a structured-decomposable circuit)**.** *Let p be a structured-decomposable circuit over variables $\mathbf{X}$. Let k be a natural number. Then there is no polynomial $f(x, y)$ such that the power p^k can be computed in $\mathcal{O}(f(|p|, k))$ time unless P=NP.*

Proof. We construct the proof by showing that for a structured-decomposable circuit p, if we could compute

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^k(\boldsymbol{x}). \tag{POWkPC}$$

in $\mathcal{O}(f(|p|, k))$ time, then we could solve the 3SAT problem in polytime, which is known to be NP-hard.

The 3SAT problem is defined as follows: given a set of n Boolean variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and a CNF that contains m clauses $\{c_1, \ldots, c_m\}$, each one containing exactly 3 literals, determine whether there exists a satisfiable configuration in $\mathsf{val}(\mathbf{X})$.

We start by constructing m gadget circuits $\{d_j\}_{j=1}^m$ for the m clauses such that $d_j(\boldsymbol{x})$ evaluates to $\frac{1}{m}$ iff $\boldsymbol{x}$ satisfies c_j and otherwise evaluates to 0, respectively.

Since each clause c_j contains exactly 3 literals, it comprises exactly 7 models w.r.t. the variables appearing in it, i.e., its scope $\phi(c_j)$. Therefore, following a similar construction in Sec. A.3, we can compile d_j as a weighted sum of 7 circuits that represent the 7 models of c_j, respectively. By choosing all weights of d_j as $\frac{1}{m}$, the circuit d_j outputs $\frac{1}{m}$ iff c_j is satisfied; otherwise it outputs 0.

The gadget circuits $\{d_j\}_{j=1}^m$ are then summed together to represent a circuit p. That is, $p = \textsc{Sum}(\{d_j\}_{j=1}^m, \{1\}_{j=1}^m)$. In the following, we complete the proof by showing that if the power circuit p^k (we will pick later $k = \lceil \max(m, n)^2 \cdot \log 2 \rceil$) can be computed in $\mathcal{O}(f(|p|, k))$ time, then the corresponding 3SAT problem can be solved in $\mathcal{O}(f(|p|, k))$ time.

If the original CNF is satisfiable, then there exists at least 1 world such that all clauses are satisfied. In this case, all circuits in $\{d_j\}_{j=1}^m$ will evaluate $\frac{1}{m}$. Since p is the sum of the circuits $\{d_j\}_{j=1}^m$, it will evaluate 1 for any world that satisfies the CNF. We obtain the bound

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^k(\boldsymbol{x}) > m \cdot \frac{1}{m} = 1.$$

In contrast, if the CNF is unsatisfiable, each variable assignment $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ satisfies at most $m - 1$ clauses, so the circuit p will output at most $\frac{m-1}{m}$. Therefore , we retrieve the following bound

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^k(\boldsymbol{x}) \leq 2^n \left(\frac{m-1}{m} \right)^k.$$

Then, we can retrieve a value for k to separate the two bounds as follows.

$$2^n \left(\frac{m-1}{m} \right)^k < 1 \;\Leftrightarrow k > \frac{\log(2^{-n})}{\log \frac{m-1}{m}} \Leftrightarrow \; k > \frac{n \log 2}{\log(m) - \log(m-1)} \overset{(a)}{\Leftrightarrow} \; k > m \cdot n \cdot \log 2,$$

where (a) follows the fact that $\log\left(\frac{m}{m-1}\right) \leq \frac{1}{m-1}$. Let $l = \max(m, n)$. If we choose $k = \lceil l^2 \cdot \log 2 \rceil$, then we can separate the two bounds above.

Therefore, if there exists a polynomial $f(x, y)$ such that the power p^k ($k = \lceil l^2 \cdot \log 2 \rceil$) can be computed in $\mathcal{O}(f(|p|, k))$ time, then we can solve 3SAT in $\mathcal{O}(f(|p|, k))$ time since the CNF is satisfiable iff $\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^k(\boldsymbol{x}) > 1$, which is impossible unless P=NP. □

Theorem 3.4 (Hardness of reciprocal of a circuit)**.** Let p be a smooth and decomposable circuit over variables $\mathbf{X}$. Then computing $p^{-1}(\mathbf{X})\big|_{\mathsf{supp}(p)}$ as a decomposable circuit is #P-hard, even if p is structured-decomposable.

Proof. We prove it for the case of PCs over discrete variables. We will prove hardness of computing the reciprocal by showing hardness of computing the partition of the reciprocal of a circuit. In

particular, let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a collection of binary variables and let p be a smooth and decomposable PC over $\mathbf{X}$, then computing the quantity

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \frac{1}{p(\boldsymbol{x})} \tag{INVPC}$$

is #P-hard.

Proof is by reduction from the EXPLR problem as defined in Thm. B.2. Similarly to Thm. B.2, the reduction is built by constructing a smooth and decomposable unnormalized circuit $p(x) = 2^n \cdot 1 + 2^n e^{-(w_0 + \sum_i w_i x_i)}$. The circuit p comprises a sum unit over two sub-circuits. The first is a uniform (unnormalized) distribution over $\mathbf{X}$ defined as a product unit over n univariate input distribution units that always output 1 for all values $\mathsf{val}(X_i)$ (see Sec. A.2 for a construction algorithm). The second is an exponential of a linear circuit (Alg. 7) and encodes $e^{-(w_0 + \sum_i w_i x_i)}$ via a product unit over n univariate input distributions, where one of them encodes $e^{-w_0 - w_1 x_1}$ and the rest $e^{-w_j x_j}$ for $j = 2, \ldots, n$. Both sub-circuits participates in the sum with parameters 2^n.

The size of the constructed circuit is linear in n, and INVPC of this circuit corresponds to the solution of the EXPLR problem. If you can represent the reciprocal of this circuit as a decomposable circuit, you can compute its marginals (including the partition function) which solves INVPC and hence EXPLR. Furthermore, the circuit is also omni-compatible because mixture of fully-factorized distributions. □

Theorem 3.5 (Tractable real power of a deterministic circuit)**.** Let p be a smooth, decomposable, and deterministic circuit over variables $\mathbf{X}$. Then, for any real number $\alpha \in \mathbb{R}$, its restricted power, defined as $a(\boldsymbol{x})|_{\mathsf{supp}(p)} = p^\alpha(\boldsymbol{x}) [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$ can be represented as a smooth, decomposable, and deterministic circuit over variables $\mathbf{X}$ in $\mathcal{O}(|p|)$ time. Moreover, if p is structured-decomposable, then a is structured-decomposable as well.

Proof. The proof proceeds by construction and recursively builds $a(\boldsymbol{x})|_{\mathsf{supp}(p)}$. As the base case, we can assume to compute the restricted α-power of the input units of p and represent it as a single new unit. When we encounter a deterministic sum unit, the power will decompose into the sum of the powers of its inputs. Specifically, let p be a sum unit: $p(\mathbf{X}) = \sum_{i \in \mathsf{in}(p)} \theta_i p_i(\mathbf{X})$. Then, its restricted real power circuit $a(\boldsymbol{x})|_{\mathsf{supp}(p)}$ can be expressed as

$$a(\boldsymbol{x})|_{\mathsf{supp}(p)} = \left(\sum_{i \in \mathsf{in}(p)} \theta_i p_i(\boldsymbol{x}) \right)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] = \sum_{i \in \mathsf{in}(p)} \theta_i^\alpha \big(p_i(\boldsymbol{x})\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!].$$

Note that this construction is possible because only one input of p will be non-zero for any input (determinism). As such, the power circuit is retaining the same structure of the original sum unit.

Next, for a decomposable product unit, its power will be the product of the powers of its inputs. Specifically, let p be a product unit: $p(\mathbf{X}) = p_1(\mathbf{X}_1) \cdot p_2(\mathbf{X}_2)$. Then, its restricted real power circuit $a(\boldsymbol{x})|_{\mathsf{supp}(p)}$ can be expressed as

$$\begin{aligned} a(\boldsymbol{x})|_{\mathsf{supp}(p)} &= \big(p_1(\boldsymbol{x}_1) \cdot p_2(\boldsymbol{x}_2)\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] \\ &= \big(p_1(\boldsymbol{x}_1)\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p_1)]\!] \cdot \big(p_2(\boldsymbol{x}_2)\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p_2)]\!]. \end{aligned}$$

Note that even this construction preserves the structure of p and hence its scope partitioning is retained throughout the whole algorithm. Hence, if p were also structured-decomposable, then a would be structured-decomposable. Alg. 5 illustrates the whole algorithm in detail. □

B.4 Quotient of Circuits

Theorem B.2 (Hardness of quotient of two circuits)**.** *Let p and q be two smooth and decomposable circuits over variables $\mathbf{X}$, and let $q(\boldsymbol{x}) \neq 0$ for every $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$. Then, computing their quotient $p(\mathbf{X})/q(\mathbf{X})$ as a decomposable circuit is #P-hard, even if they are compatible.*

Algorithm 5 POWER($p, \alpha, \mathsf{cache}$)

Input: a smooth, deterministic and decomposable circuit $p(\mathbf{X})$, a scalar $\alpha \in \mathbb{R}$, and a cache for memoization
Output: a smooth, deterministic and decomposable circuit $a(\mathbf{X})$ encoding $p^\alpha(\mathbf{X})|_{\mathsf{supp}(p)}$
if $p \in \mathsf{cache}$ **then return** $\mathsf{cache}(p)$
if p is an input unit **then** $a \leftarrow \textsc{Input}(p^\alpha(\mathbf{X})|_{\mathsf{supp}(p)}, \phi(p))$
else if p is a sum unit **then** $a \leftarrow \textsc{Sum}(\{\textsc{power}(p_i, \alpha, \mathsf{cache})\}_{i=1}^{|\mathsf{in}(p)|}, \{\theta_i^\alpha\}_{i=1}^{|\mathsf{in}(p)|})$
else if p is a product unit **then** $a \leftarrow \textsc{Product}(\{\textsc{power}(p_i, \alpha, , \mathsf{cache})\}_{i=1}^{|\mathsf{in}(p)|})$
$\mathsf{cache}(p) \leftarrow a$
return a

Proof. This result follows from Thm. 3.4 by noting that computing the reciprocal of a circuit is a special case of computing the quotient of two circuits. In particular, let p be an omni-compatible circuit representing the constant function 1 over variables $\mathbf{X}$, constructed as in Sec. A.2. Then computing the reciprocal of a structured-decomposable circuit q as a decomposable circuit reduces to computing the quotient p/q. □

Theorem B.3 (Tractable restricted quotient of two circuits)**.** *Let p and q be two compatible circuits over variables $\mathbf{X}$, and let q be also deterministic. Then, their quotient restricted to $\mathsf{supp}(q)$ can be represented as a circuit compatible with p (and q) over variables $\mathbf{X}$ in $\mathcal{O}(|p|\,|q|)$ time. Moreover, if p is also deterministic, then the quotient circuit is deterministic as well.*

Proof. We know from Thm. 3.5 that we can obtain the reciprocal circuit q^{-1} that is also compatible with q (and by extension p) in $\mathcal{O}(|q|)$ time. Then we can multiply p and q^{-1} in $\mathcal{O}(|p|\,|q|)$ time using Thm. 3.2 to compute their quotient circuit that is still compatible with p and q. If p is also deterministic, then we are multiplying two deterministic circuits and therefore their product circuit is deterministic (Thm. 3.2). □

B.5 Logarithm of a PC

Theorem 3.6 (Logarithms)**.** ***(Tractability)*** Let p be a smooth, deterministic and decomposable PC over variables $\mathbf{X}$. Then its logarithm circuit, restricted to the support of p and defined as

$$l(\boldsymbol{x})|_{\mathsf{supp}(p)} = \begin{cases} \log p(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in \mathsf{supp}(p) \\ 0 & \text{otherwise} \end{cases}$$

for every $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ can be represented as a smooth and decomposable circuit that shares the scope partitioning of p in $\mathcal{O}(|p|)$ time. ***(Hardness)*** Otherwise, if p is a smooth and decomposable PC, then computing its logarithm circuit $l(\mathbf{X}) := \log p(\mathbf{X})$ as a decomposable circuit is #P-hard, even if p is structured-decomposable.

We will provide the proofs for tractability and hardness separately below.

Proof of tractability. The proof proceeds by recursively constructing $l(\boldsymbol{x})|_{\mathsf{supp}(p)}$. In the base case, we assume computing the logarithm of an input unit can be done in $\mathcal{O}(1)$ time. When we encounter a deterministic sum unit $p(\boldsymbol{x}) = \sum_{i \in |\mathsf{in}(p)|} \theta_i p_i(\boldsymbol{x})$, its logarithm circuit consists of the sum of (i) the logarithm circuits of its child units and (ii) the support circuits of its children weighted by their respective weights $\{\theta_i\}_{i=1}^{|\mathsf{in}(p)|}$:

$$\begin{aligned} l(\boldsymbol{x})|_{\mathsf{supp}(\boldsymbol{x})} &= \log\left(\sum_{i \in \mathsf{in}(p)} \theta_i p_i(\boldsymbol{x})\right) \cdot [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] = \sum_{i \in |\mathsf{in}(p)|} \log\Big(\theta_i p_i(\boldsymbol{x})\Big)[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!] \\ &= \sum_{i \in |\mathsf{in}(p)|} \log\theta_i [\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!] + \sum_{i \in |\mathsf{in}(p)|} l_i(\boldsymbol{x})|_{\mathsf{supp}(p_i)}\,. \end{aligned}$$

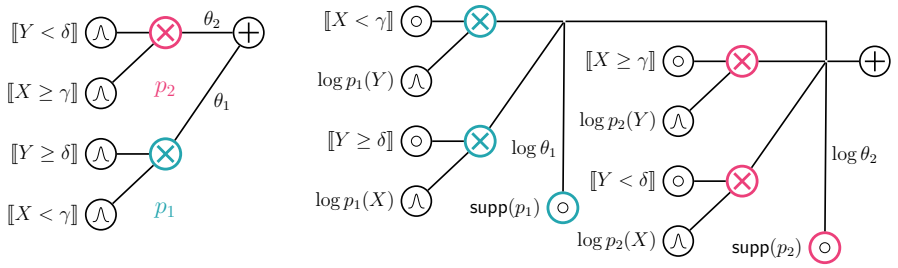


Figure 3: Building the logarithmic circuit (right) for a deterministic PC (left) whose input units are labeled by their supports. A single sum unit is introduced over smoothed product units and additional dummy input units which share the same support across circuits if they have the same color.

For a smooth, decomposable, and deterministic product unit $p(\boldsymbol{x}) = p_1(\boldsymbol{x})p_2(\boldsymbol{x})$, its logarithm circuit can be decomposed as sum of the logarithm circuits of its child units:

$$
\begin{aligned}
&l(\boldsymbol{x})|_{\mathsf{supp}(\boldsymbol{x})} = \log\left(p_1(\boldsymbol{x}_1)p_2(\boldsymbol{x}_2)\right) \cdot [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] \\
&= \log p_1(\boldsymbol{x}_1)[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] + \log p_2(\boldsymbol{x}_2)[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] \\
&= \log p_1(\boldsymbol{x}_1)[\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!][\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!] + \log p_2(\boldsymbol{x}_2)[\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!][\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!] \\
&= l(\boldsymbol{x}_1)|_{\mathsf{supp}(p_1)}\,[\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!] + l(\boldsymbol{x}_2)|_{\mathsf{supp}(p_2)}\,[\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!].
\end{aligned}
$$

Note that in both case, the support circuits (e.g., $[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$) are used to enforce smoothness in the output circuit. Alg. 6 illustrates the whole algorithm in detail, showing that the construction of these support circuits can be done in linear time by caching intermediate sub-circuits while calling Alg. 1. Furthermore, the newly introduced product units, i.e., $l(\boldsymbol{x}_1)|_{\mathsf{supp}(p_1)}\,[\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!]$, $l(\boldsymbol{x}_2)|_{\mathsf{supp}(p_2)}\,[\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!]$, and the additional support input unit $\log\theta_i[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!]$ share the same support of p by construction. Fig. 3 illustrates this property with one example. This implies that when a deterministic circuit and its logarithmic circuit are going to be multiplied, e.g., when computing entropies (Sec. C.2), we can check for their support to overlap in linear time (Alg. 3). □

Proof of hardness. We will prove hardness of computing the logarithm by showing hardness of computing the partition function of the logarithm of a circuit. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a collection of binary variables, and p a smooth and decomposable PC over $\mathbf{X}$ where $p(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$. Then computing the quantity

$$
\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \log p(\boldsymbol{x}) \tag{LOGPC}
$$

is #P-hard.

The proof is by reduction from #NUMPAR, the counting problem of the number partitioning problem (NUMPAR) defined as follows. Given n positive integers $k_1, \ldots, k_n$, we want to decide whether there exists a subset $S \subset [n]$ such that $\sum_{i \in S} k_i = \sum_{i \notin S} k_i$. NUMPAR is NP-complete, and #NUMPAR which asks for the number of solutions is known to be #P-hard.

We will show that we can solve #NUMPAR using an oracle for LOGPC, which will imply that LOGPC is also #P-hard. First, consider the following quantity SL for a given weight function $w(\cdot)$:

$$
\begin{aligned}
\mathsf{SL} &:= \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \log(\sigma(w(\boldsymbol{x})) + 1) = \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \log\left(\frac{1}{1 + e^{-w(\boldsymbol{x})}} + 1\right) = \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \log\left(\frac{2 + e^{-w(\boldsymbol{x})}}{1 + e^{-w(\boldsymbol{x})}}\right) \\
&= \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \log(2 + e^{-w(\boldsymbol{x})}) - \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \log(1 + e^{-w(\boldsymbol{x})}).
\end{aligned}
$$

Similar to the construction in the proof of Thm. 3.4, we can construct smooth and decomposable, unnormalized PCs for $2 + e^{-w(\boldsymbol{x})}$ and $1 + e^{-w(\boldsymbol{x})}$ of size linear in n. Then, we can compute SL via two calls to the oracle for LOGPC on these PCs.

Next, we choose the weight function $w(\cdot)$ such that SL can be used to answer #NUMPAR. For a given instance of NUMPAR described by $k_1, \ldots, k_n$ and a large integer m, which will be chosen

later, we define the following weight function:

$$w(\boldsymbol{x}) := -\frac{m}{2} - m\sum_i k_i + 2m\sum_i k_i x_i.$$

In other words, $w(\boldsymbol{x}) = w_0 + \sum_i w_i x_i$ where $w_0 = -m/2 - m\sum_i k_i$ and $w_i = 2mk_i$ for $i = 1, \ldots, n$. Here, an assignment $\boldsymbol{x}$ corresponds to a subset $S_{\boldsymbol{x}} = \{i | x_i = 1, x_i \in \boldsymbol{x}\}$. Then the assignment $1 - \boldsymbol{x}$ corresponds to the complement $S_{1-\boldsymbol{x}} = \overline{S_{\boldsymbol{x}}}$. In the following, we will consider pairs of assignments $(\boldsymbol{x}, 1-\boldsymbol{x})$ and say that it is a solution to NUMPAR if $S_{\boldsymbol{x}}$ and by extension $S_{1-\boldsymbol{x}}$ are solutions to NUMPAR.

Observe that if $(\boldsymbol{x}, 1-\boldsymbol{x})$ is a solution to NUMPAR, then $w(\boldsymbol{x}) = w(1-\boldsymbol{x}) = -m/2$. Otherwise, one of their weights must be $\geq m/2$ and the other $\leq -3m/2$. We can then deduce the following facts about the *contribution* of each pair to SL, defined as $c(\boldsymbol{x}, 1-\boldsymbol{x}) = \log(\sigma(w(\boldsymbol{x}))+1) + \log(\sigma(w(1-\boldsymbol{x}))+1)$.

If the pair $(\boldsymbol{x}, 1-\boldsymbol{x})$ is a solution to NUMPAR, then its contribution to SL is going to be:

$$c(\boldsymbol{x}, 1-\boldsymbol{x}) = 2\log(\sigma(-m/2)+1).$$

Otherwise, we can bound its contribution as follows:

$$\log(\sigma(m/2)+1) \leq c(\boldsymbol{x}, 1-\boldsymbol{x}) \leq 1 + \log(\sigma(-3m/2)+1)$$

If there are k pairs that are solutions to the NUMPAR problem, then using the above observations we have the following bounds on SL:

$$\mathsf{SL} \geq (2^{n-1}-k)\log\left(\sigma(m/2)+1\right) + 2k\log\left(\sigma(-m/2)+1\right) \geq (2^{n-1}-k)\log\left(\sigma(m/2)+1\right), \tag{1}$$

$$\mathsf{SL} \leq (2^{n-1}-k)(1+\log\left(\sigma(-3m/2)+1\right)) + 2k\log(\sigma(-m/2)+1). \tag{2}$$

Suppose for some given $\epsilon > 0$, we select m such that it satisfies both $1-\epsilon \leq \log(\sigma(m/2)+1)$ and $\log(\sigma(-m/2)+1) \leq \epsilon$. First, this implies that m also satisfies the following:

$$1 + \log\left(\sigma(-3m/2)+1\right)) \leq 1 + \log(\sigma(-m/2)+1) \leq 1+\epsilon.$$

Plugging in above inequalities to Eqs. (1) and (2), we get the following bounds on SL w.r.t. ϵ and k:

$$(2^{n-1}-k)(1-\epsilon) \leq \mathsf{SL} \leq (2^{n-1}-k)(1+\epsilon) + 2k\epsilon.$$

We can alternatively express this as the following bounds on k:

$$\frac{2^{n-1}(1-\epsilon) - \mathsf{SL}}{1-\epsilon} \leq k \leq \frac{2^{n-1}(1+\epsilon) - \mathsf{SL}}{1-\epsilon}.$$

The difference between the upper and lower bounds on k is equal to $2^n\epsilon/(1-\epsilon)$. If this difference is less than 1—e.g. by setting $\epsilon = 1/(2^n+2)$—we can exactly solve for k. In particular, it must be equal to the ceiling of the lower bound as well as the floor of the upper bound. Moreover, the answer to #NUMPAR is given by $2k$. This concludes the proof that computing LOGPC is #P-hard. □

B.6 Exponential Function of a Circuit

Theorem 3.7 (Hardness of the exponential of a circuit)**.** Let p be a smooth and decomposable circuit over variables $\mathbf{X}$. Then, computing its exponential $\exp(p(\mathbf{X}))$ as a decomposable circuit is #P-hard, even if p is structured-decomposable.

Proof. We will prove hardness of computing the exponential by showing hardness of computing the partition function of the exponential of a circuit. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a collection of binary variables with values in $\{-1, +1\}$ and let p be a smooth and decomposable PC over $\mathbf{X}$ then computing the quantity

$$\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \exp\left(p(\boldsymbol{x})\right) \tag{EXPOPC}$$

Algorithm 6 LOGARITHM($p, \mathsf{cache}_l, \mathsf{cache}_s$)

Input: a smooth, deterministic and decomposable PC $p(\mathbf{X})$ and two caches for memoization (cache_l for the logarithmic circuit and cache_s for the support circuit).
Output: a smooth and decomposable circuit $l(\mathbf{X})$ encoding $\log\left(p(\mathbf{X})\right)$
if $p \in \mathsf{cache}_l$ **then return** $\mathsf{cache}_l(p)$
if p is an input unit **then**
  $l \leftarrow \textsc{Input}(\log\left(p_{|\mathsf{supp}(p)}\right), \phi(p))$
else if p is a sum unit **then**
  $n \leftarrow \{\}$
  for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
    $n \leftarrow n \cup \{\textsc{Support}(p_i, \mathsf{cache}_s)\} \cup \{\textsc{Logarithm}(p_i, \mathsf{cache}_l)\}$
  $l \leftarrow \textsc{Sum}(n, \{\log\theta_1, 1, \log\theta_2, 1, \ldots, \log\theta_{|\mathsf{in}(p)|}, 1\})$
else if p is a product unit **then**
  $n \leftarrow \{\}$
  for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
    $n \leftarrow n \cup \{\textsc{Product}(\{\textsc{Logarithm}(p_i, \mathsf{cache}_l)\} \cup \{\textsc{Support}(p_j, \mathsf{cache}_s)\}_{j\neq i})\}$
  $l \leftarrow \textsc{Sum}(n, \{1\}_{i=1}^{|\mathsf{in}(p)|})$
$\mathsf{cache}_l(p) \leftarrow l$
return l

is #P-hard.

The proof is a reduction from the problem of computing the partition function of an Ising model, ISING which is known to be #P-complete [20]. Given a graph $G = (V, E)$ with n vertexes, computing the partition function of an Ising model associated to G and equipped with potentials associated to its edges ($\{w_{u,v}\}_{(u,v)\in E}$) and vertexes ($\{w_v\}_{v\in V}$) equals to

$$\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \exp\left(\sum_{(u,v)\in E} w_{u,v}x_u x_v + \sum_{v\in V} w_v x_v\right). \tag{ISING}$$

The reduction is made by constructing a smooth and decomposable circuit $p(\mathbf{X})$ that computes $\sum_{(u,v)\in E} w_{u,v}x_u x_v + \sum_{v\in V}$. This can be done by introducing a sum units with $|E| + |V|$ inputs that are product units and with weights $\{w_{u,v}\}_{(u,v)\in E} \cup \{w_v\}_{v\in V}$. The first $|E|$ product units receive inputs from n input distributions where only 2 corresponds to the binary indicator inputs X_u and X_v for an edge $(u, v) \in E$ while the remaining $n - 2$ are uniform distributions outputting 1 for all the possible states of variables $\mathbf{X} \setminus \{X_u, X_v\}$. Analogously, the remaining $|V|$ product units receive input from n of which only one, corresponding to the vertex $v \in V$ is an indicator unit over X_v, while the remaining are uniform distributions for variables in $\mathbf{X} \setminus \{X_v\}$. □

Proposition 3.1 (Tractable exponential of a linear circuit)**.** Let p be a linear circuit over variables $\mathbf{X}$, i.e., $p(\mathbf{X}) = \sum_i \theta_i \cdot X_i$. Then $\exp\left(p(\mathbf{X})\right)$ can be represented as an omni-compatible circuit with a single product unit in $\mathcal{O}(|p|)$ time.

Proof. The proof follows immediately by the properties of exponentials of sums. Alg. 7 formalizes the construction. □

Algorithm 7 EXPONENTIAL(p)

Input: a smooth circuit p encoding $p(\mathbf{X}) = \theta_0 + \sum_{i=1}^n \theta_i X_i$
Output: its exponential circuit encoding $\exp\left(p(\mathbf{X})\right)$
$e \leftarrow \{\textsc{Input}(\exp\left(\theta_0 + \theta_1 X_1\right), X_1)\}$
for $i = 2$ **to** n **do**
  $e \leftarrow e \cup \{\textsc{Input}(\exp\left(\theta_i X_i\right), X_i)\}$
return $\textsc{Product}(e)$

B.7 Other tractable operators over circuits

This section proves Lemma 3.8, which states that any operator over circuits that should yield a decomposable and smooth circuit as output must take the form of a sum, power, logarithm or exponential.

Lemma 3.8 (Atlas Completeness)**.** Let f be a continuous function. If (1) $f : \mathbb{R} \to \mathbb{R}$ satisfies $f(x+y) = f(x) + f(y)$ then it is a linear function $\beta \cdot x$; if (2) $f : \mathbb{R}_+ \to \mathbb{R}_+$ satisfies $f(x \cdot y) = f(x) \cdot f(y)$, then it takes the form x^β; if (3) instead $f : \mathbb{R}_+ \to \mathbb{R}$ satisfies $f(x \cdot y) = f(x) + f(y)$, then it takes the form $\beta \log(x)$; and if (4) $f : \mathbb{R} \to \mathbb{R}_+$ satisfies that $f(x+y) = f(x) \cdot f(y)$ then it is of the form $\exp(\beta \cdot x)$, for a certain $\beta \in \mathbb{R}$.

Proof. The proof of all properties follows from constructing f such that we obtain a *Cauchy functional equation* [21, 42].

The condition (1) exactly takes the form of a Cauchy functional equation, then it must hold that $f(x) = \beta \cdot x$.

For condition (2), let $g(x) = \log(f(\exp(x)))$ for all $x \in \mathbb{R}$, which is continuous because f is. Then, it follows that

$$\begin{aligned} g(x+y) &= \log(f(\exp(x+y))) = \log(f(\exp(x) \cdot \exp(y))) = \log(f(\exp(x))) + \log(f(\exp(y))) \\ &= g(x) + g(y). \end{aligned}$$

Therefore, $g(x)$ assumes the Cauchy functional form and, as in case (1), it is equal to $\beta \cdot x$. β can be retrieved by solving $\beta \cdot x = \log(f(\exp(x)))$ for $x = 1$. This gives $\beta = \log(f(e))$. Applying the definition of g, we can hence write

$$f(\exp(x)) = e^{g(x)} = e^{\beta \cdot x} = (e^x)^\beta$$

Let $y \in \mathbb{R}_+$. Using the identity $y = e^{\log(y)}$ it follows that:

$$f(y) = f(e^{\log(y)}) = \left(e^{\log(y)}\right)^\beta = y^\beta.$$

Condition (3) follows an analogous pattern. Let $g(x) = f(\exp(x))$ for all $x \in \mathbb{R}$, which is continuous as f is. Once again, g satisfies the Cauchy functional form:

$$g(x+y) = f(\exp(x+y)) = f(\exp(x) \cdot \exp(y)) = f(\exp(x)) + f(\exp(y)) = g(x) + g(y).$$

Therefore, $g(x)$ must be of the form $\beta \cdot x$ for $\beta = f(e)$. Hence, $f(y) = \beta \log(y)$.

Lastly, for condition (4), $g(x) = \log(f(x))$ for all $x \in \mathbb{R}$, which is continuous if f is. Then, we can retrieve the Cauchy functional by

$$g(x+y) = \log(f(x+y)) = \log(f(x) \cdot f(y)) = \log(f(x)) + \log(f(y)) = g(x) + g(y).$$

Therefore, $g(x)$ must be of the form $\beta \cdot x$. Hence, $f(y) = \exp(\beta \cdot y)$. □

In summary, Lemma 3.8 states that if we want to enlarge our atlas beyond sum and product circuit operators, we need to focus our attention over powers, logarithms and exponentials. At the same time, it states that no operator with a different functional form and yet yielding a circuit made of sum and product units can be found. Extending our atlas to deal with a new language of circuits is an interesting future research direction.

C Complex Information-Theoretic Queries

This section collects the complete tractability and hardness results for the queries in Tab. 2. Note that the tractability proofs are succinct thanks to our atlas which allows to define a tractable model class effortlessly. Some hardness proofs also benefit from the hardness results we provided for the simple operators in the previous section.

C.1 Cross Entropy

Theorem C.1. *Let p and q be two compatible PCs over variables $\mathbf{X}$, and also let q be deterministic. Then their cross-entropy, i.e.,*

$$-\int_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) \log(q(\boldsymbol{x}))d\mathbf{X},$$

restricted to the support of q can be exactly computed in $\mathcal{O}(|p|\,|q|)$ time. If q is not deterministic, then computing their cross-entropy is #P-hard, even if p and q are compatible over $\mathbf{X}$.

Proof. ***(Tractability)*** From Thm. 3.6 we know that we can compute the logarithm of q in polytime, which is a PC of size $\mathcal{O}(|q|)$ that is compatible with q and hence with p. Therefore, multiplying p and $\log q$ according to Thm. 3.1 can be done exactly in polytime and yields a circuit of size $\mathcal{O}(|p|\,|q|)$ that is still smooth and decomposable, hence we can tractably compute its partition function.

(Hardness) The proof consists of a simple reduction from LOGPC from Thm. 3.6. We know that computing LOGPC for a smooth and decomposable PC over binary variables $\mathbf{X}$ is #P-hard. We can reduce this to computing the cross entropy between $p = 1$, which can be constructed as an omni-compatible circuit (Sec. A.2), and the original PC of the LOGPC problem. Thus, the cross-entropy of two compatible circuits is a #P-hard problem. □

C.2 Entropy

Theorem C.2. *Let p be a smooth, deterministic, and decomposable PC over variables $\mathbf{X}$. Then its entropy,*[9] *defined as*

$$-\int_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) \log p(\boldsymbol{x})\; d\mathbf{X}$$

can be exactly computed in $\mathcal{O}(|p|)$ time. If p is smooth and decomposable but not deterministic, then computing its Shannon entropy, defined as

$$\textsc{ent}(p) := -\sum_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) \log(p(\boldsymbol{x}))d\mathbf{X} \tag{ENTPC}$$

is coNP-hard.

Proof. ***(Tractability)*** Using Thm. 3.6 we can compute the logarithm of p in polytime as a smooth and decomposable PC of size $\mathcal{O}(|p|)$ which furthermore shares the same support partitioning with p. Therefore, multiplying p and $\log p$ according to Alg. 3 can be done in polytime and yields a smooth and decomposable circuit of size $\mathcal{O}(|p|)$ since $\log p$ shares the same support structure of p (Thm. 3.6). Therefore, we can compute the partition function of the resulting circuit in time linear in its size.

(Hardness) The hardness proof contains a polytime reduction from the coNP-hard 3UNSAT problem, defined as follows: given a set of n Boolean variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and a CNF with m clauses $\{c_1, \ldots, c_m\}$ (each clause contains exactly 3 literals), decide whether the CNF is unsatisfiable.

The reduction borrows two gadget circuits p_β and p_γ defined in Sec. A.3. They each represent a logical formula over an auxiliary set of variables, which we denote here $\mathbf{X}'$, and thus outputs 0 or 1 for all values of $\mathbf{X}'$. Moreover, by construction, $p_\beta \cdot p_\gamma$ is the constant function 0 if and only if the original CNF is unsatisfiable.

We further construct a circuit p_α as the summation over p_β and p_γ. Recall that p_β and p_γ can efficiently be constructed as smooth and decomposable circuits, and thus their sum can be represented as a smooth and decomposable circuit in polynomial time. We will now show that 3UNSAT can be reduced to checking whether the entropy of p_α is zero.

First, observe that for any assignment $\boldsymbol{x}'$ to $\mathbf{X}'$, $p_\alpha(\boldsymbol{x}')$ evaluates to 0, 1, or 2, because p_β and p_γ always evaluates to either 0 or 1. Moreover, if p_α only outputs 0 or 1 for all values of $\mathbf{X}'$, then $p_\beta \cdot p_\gamma$ must always be 0, implying that the original CNF is unsatisfiable. Lastly, in such a case, the entropy of p_α must be 0, whereas the entropy will be nonzero if there is an assignment $\boldsymbol{x}'$ such that

[9] For the continuous case this quantity refers to the *differential entropy*, while for the discrete case it is the Shannon entropy.

$p_\alpha(\boldsymbol{x}') = 2$. This concludes the proof that computing the entropy of a smooth and decomposable PC is coNP-hard. □

C.3 Mutual Information

Theorem C.3. *Let p be a deterministic and structured-decomposable PC over variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$ ($\mathbf{X} \cap \mathbf{Y} = \emptyset$). Then the mutual information between $\mathbf{X}$ and $\mathbf{Y}$, defined as*

$$\text{MI}(p; \mathbf{X}, \mathbf{Y}) := \int_{\mathsf{val}(\mathbf{Z})} p(\boldsymbol{x}, \boldsymbol{y}) \log \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x}) \cdot p(\boldsymbol{y})} d\mathbf{X} d\mathbf{Y},$$

can be exactly computed in $\mathcal{O}(|p|)$ time if p is still deterministic after marginalizing out $\mathbf{Y}$ as well as after marginalizing out $\mathbf{X}$.[10] If p is instead smooth, decomposable, and deterministic, then computing the mutual information between $\mathbf{X}$ and $\mathbf{Y}$ is coNP-hard.

Proof. ***(Tractability)*** From Thm. 3.6 we know that the logarithm circuits of $p(\mathbf{X}, \mathbf{Y})$, $p(\mathbf{X})[\![\boldsymbol{y} \in \mathsf{supp}(p(\mathbf{Y}))]\!]$, and $p(\mathbf{Y})[\![\boldsymbol{x} \in \mathsf{supp}(p(\mathbf{X}))]\!]$ can be computed in polytime and are smooth and decomposable circuits of size $\mathcal{O}(|p|)$ that furthermore share the same support partitioning with $p(\mathbf{Y}, \mathbf{Z})$. Therefore, we can multiply $p(\mathbf{X}, \mathbf{Y})$ with each of these logarithm circuits efficiently according to Thm. 3.2 to yield circuits of size $\mathcal{O}(|p|)$. These are still smooth and decomposable circuits. Hence we can compute their partition functions and compute the mutual information between $\mathbf{X}$ and $\mathbf{Y}$ w.r.t. p.

(Hardness) We show hardness for the case of Boolean inputs, which implies hardness in the general case. This proof largely follows the hardness proof of Thm. C.2 to show that there is a polytime reduction from 3UNSAT to the mutual information of PCs. For a given CNF, suppose we construct p_β, p_γ, and $p_\alpha = p_\beta + p_\gamma$ over a set of Boolean variables, say $\mathbf{X}$, as shown in Sec. A.2 and Thm. C.2.

Let $\mathbf{Y} = \{Y\}$ be a single Boolean variable, and define p_δ as:

$$p_\delta := p_\beta \times [\![Y = 1]\!] + p_\gamma \times [\![Y = 0]\!].$$

That is, we first construct two product units q_1, q_2 with inputs $\{p_\beta, [\![Y = 1]\!]\}$ and $\{p_\gamma, [\![Y = 0]\!]\}$, respectively, and build a sum unit p_δ with inputs $\{q_1, q_2\}$ and weights $\{1, 1\}$. Then p_δ has the following properties: **(1)** p_δ is smooth, decomposable, and deterministic, following from the fact that p_β and p_γ are also smooth, decomposable, and deterministic, and that q_1 and q_2 have no overlapping support. **(2)** $\text{ENT}(p_\delta)$ can be computed in linear-time w.r.t. the circuit size by Thm. C.2. **(3)** $p_\delta(Y = 1)$ and $p_\delta(Y = 0)$ can be computed in linear time (w.r.t. size of the circuit p_δ), as p_δ admits tractable marginalization. **(4)** For any $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$, $p_\delta(\boldsymbol{x}) = p_\beta(\boldsymbol{x}) + p_\gamma(\boldsymbol{x}) = p_\alpha(\boldsymbol{x})$.

We can express the mutual information $\text{MI}(p_\delta; \mathbf{X}, \mathbf{Y})$ as:

$$\text{MI}(p_\delta; \mathbf{X}, \mathbf{Y}) = \text{ENT}(p_\delta) - p_\delta(Y=1) \log p_\delta(Y=1) - p_\delta(Y=0) \log p_\delta(Y=0) - \text{ENT}(p_\alpha).$$

Therefore, given an oracle that computes $\text{MI}(p_\delta; \mathbf{X}, \mathbf{Y})$, we can check if it is equal to $\text{ENT}(p_\delta) - p_\delta(Y = 1) \log p_\delta(Y = 1) - p_\delta(Y = 0) \log p_\delta(Y = 0)$, which is equivalent to checking $\text{ENT}(p_\alpha) = 0$, and decide whether the original CNF is unsatisfiable. Hence, computing the mutual information of smooth, deterministic, and decomposable PCs is a coNP-hard problem. □

C.4 Kullback-Leibler Divergence

Theorem C.4. *Let p and q be two deterministic and compatible PCs over variables $\mathbf{X}$. Then, their intersectional Kullback-Leibler divergence (KLD), defined as*

$$\mathbb{D}_{\mathsf{KL}}(p \parallel q) = \int_{\mathsf{supp}(p) \cap \mathsf{supp}(q)} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\mathbf{X},$$

can exactly be computed in $\mathcal{O}(|p|\,|q|)$ time. If p and q are not deterministic, then computing their KLD is #P-hard, even if they are compatible.

[10] This structural property of circuits is also known as marginal determinism [8] and has been introduced in the context of marginal MAP inference and the computation of same-decision probabilities of Bayesian classifiers [35, 5].

Proof. ***(Tractability)*** Tractability of the intersectional KLD can be concluded directly from the tractability of cross entropy and entropy (Thm. C.1 and C.2). Specifically, KLD can be expressed as the difference between cross entropy and entropy:

$$\int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\mathbf{X} = \int p(\boldsymbol{x}) \log p(\boldsymbol{x})\, d\mathbf{X} - \int p(\boldsymbol{x}) \log q(\boldsymbol{x})\, d\mathbf{X}.$$

We can compute the entropy of a smooth, decomposable, and deterministic PC p in $\mathcal{O}(|p|)$; and the cross entropy between two deterministic and compatible PCs p and q in $\mathcal{O}(|p|\,|q|)$ time.

(Hardness) The proof proceeds similarly to the hardness proof of Thm. C.1. Recall that the LOGPC problem from Thm. 3.6 is #P-hard for a smooth and decomposable PC over binary variables. We can reduce this to computing the negative of KL divergence between $p = 1$, which can be constructed as an omni-compatible circuit (Sec. A.2), and q the original PC of the LOGPC problem. Thus, the KLD of two compatible circuits is a #P-hard problem. □

C.5 Rényi Entropy

Definition C.1 (Rényi entropy)**.** The Rényi entropy of order $\alpha \in \mathbb{R}$ of a PC p is defined as

$$\frac{1}{1-\alpha} \log \int_{\mathsf{supp}(p)} p^{\alpha}(\boldsymbol{x}) d\mathbf{X}.$$

Theorem C.5 (Rényi entropy for natural α)**.** *Let p be a structured-decomposable PC over variables $\mathbf{X}$ and $\alpha \in \mathbb{N}$. Its Rényi entropy can be computed in $\mathcal{O}(|p|^{\alpha})$ time. If p is instead smooth and decomposable, then computing its Rényi entropy of order α is #P-hard.*

Proof. ***(Tractability)*** Tractability easily follows from computing the natural power circuit of p, which takes $\mathcal{O}(|p|^{\alpha})$ time according to Thm. 3.3.

(Hardness) We show hardness for the case of discrete inputs. The hardness of computing the Rényi entropy for natural number α is implied by the hardness of computing the natural power of smooth and decomposable PCs. Specifically, we conclude the proof by observing that there exists a polytime reduction from POW2PC, defined as $\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^2(\boldsymbol{x})$, a #P-hard problem as proved in Thm. 3.3, to Rényi entropy with $\alpha = 2$. □

Theorem C.6 (Rényi entropy for real α)**.** *Let p be a smooth, decomposable, and deterministic PC over variables $\mathbf{X}$ and $\alpha \in \mathbb{R}_+$. Its Rényi entropy can be computed in $\mathcal{O}(|p|)$ time. If p is not deterministic, then computing its Rényi entropy of order α is #P-hard, even if p is structured-decomposable.*

Proof. ***(Tractability)*** Tractability easily follows from computing the power circuit of p, which takes $\mathcal{O}(|p|)$ time according to Thm. 3.5.

(Hardness) Similar to the hardness proof of Thm. C.5, this hardness result follows from the fact that computing the reciprocal of a structured-decomposable circuit is #P-hard (Thm. 3.4). Again, this is demonstrated by a polytime reduction from INVPC (i.e., $\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} p^{-1}(\boldsymbol{x})$) to Rényi entropy with $\alpha = -1$. □

C.6 Rényi's α-divergence

Definition C.2 (Rényi's α-divergence)**.** The Rényi's α-divergence of two PCs p and q is defined as

$$\mathbb{D}_{\alpha}(p \parallel q) = \frac{1}{1-\alpha} \log \int_{\mathsf{supp}(p) \cap \mathsf{supp}(q)} p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) d\mathbf{X}.$$

Theorem C.7 (Hardness of alpha divergence of two PCs)**.** *Let p and q be two smooth and decomposable PCs over variables $\mathbf{X}$. Then computing their Rényi's α-divergence for $\alpha \in \mathbb{R} \setminus \{1\}$ is #P-hard, even if p and q are compatible.*

Proof. Suppose p is a smooth and decomposable PC $\mathbf{X}$ representing the constant function 1, which can be constructed as in Sec. A.2. Then p^{α} is also a constant 1. Hence, computing Rényi's 2-divergence between p and another smooth and decomposable PC q is as hard as computing the reciprocal of q, which is #P-hard (Thm. 3.4). □

Theorem 4.1 (Tractable alpha divergence of two PCs)**.** Let p and q be compatible PCs over variables $\mathbf{X}$. Then their Rényi's α-divergence can be exactly computed in $\mathcal{O}(|p|^{\alpha}\,|q|)$ time for $\alpha \in \mathbb{N}, \alpha > 1$ if q is deterministic or in $\mathcal{O}(|p|\,|q|)$ for $\alpha \in \mathbb{R}, \alpha \neq 1$ if p and q are both deterministic.

Proof. The proof easily follows from first computing the power circuit of p and q according to Thm. 3.5 or Thm. 3.3 in polytime. Depending on the value of α, the resulting circuits will have size $\mathcal{O}(|p|^{\alpha})$ and $\mathcal{O}(|q|)$ for $\alpha \in \mathbb{N}$ or $\mathcal{O}(|p|)$ and $\mathcal{O}(|q|)$ for $\alpha \in \mathbb{R}$ and will be compatible with the input circuits. Then, since they are compatible between themselves, their product can be done in polytime (Thm. 3.2) and it is going to be a smooth and decomposable PC of size $\mathcal{O}(|p|^{\alpha}\,|q|)$ (for $\alpha \in \mathbb{N}$) or $\mathcal{O}(|p|\,|q|)$ (for $\alpha \in \mathbb{R}$), for which the partition function can be computed in time linear in its size. □

C.7 Itakura-Saito Divergence

Theorem C.8. *Let p and q be two deterministic and compatible PCs over variables $\mathbf{X}$, with bounded intersectional support $\mathsf{supp}(p) \cap \mathsf{supp}(q)$. Then their Itakura-Saito divergence, defined as*

$$\mathbb{D}_{\mathsf{IS}}(p \parallel q) = \int_{\mathsf{supp}(p)\cap\mathsf{supp}(q)} \left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - \log\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - 1\right) d\mathbf{X}, \tag{3}$$

can be exactly computed in $\mathcal{O}(|p|\,|q|)$ time. If p and q are instead compatible but not deterministic, then computing their Itakura-Saito divergence is #P-hard.

Proof. ***(Tractability)*** The proof easily follows from noting that the integral decomposes into three integrals over the inner sum: $\int_{\mathsf{supp}(p)\cap\mathsf{supp}(q)} \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\mathbf{X} - \int_{\mathsf{supp}(p)\cap\mathsf{supp}(q)} \log\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\mathbf{X}$ - $\int_{\mathsf{supp}(p)\cap\mathsf{supp}(q)} 1\, d\mathbf{X}$.. Then, the first integral over the quotient can be solved $\mathcal{O}(|p|\,|q|)$ (Thm. B.3); the second integral over the log of a quotient of two PCs can be computed in time $\mathcal{O}(|p|\,|q|)$ (Thm. 3.6 and B.3) and finally the last one integrates to the dimensionality of $|\mathsf{supp}(p) \cap \mathsf{supp}(q)|$, which we assume to exist.

(Hardness) We show hardness for the case of binary variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Suppose q is an omni-compatible circuit representing the constant function 1, which can be constructed as in Sec. A.2. As such, integration in Eq. (3) becomes the summation $\sum_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) - \sum_{\mathsf{val}(\mathbf{X})} \log p(\boldsymbol{x}) - 2^n$. Hence, computing $\mathbb{D}_{\mathsf{IS}}$ must be as hard as computing $\sum_{\mathsf{val}(\mathbf{X})} \log p(\boldsymbol{x})$, since the first sum can be efficiently computed as p must be smooth and decomposable by assumption and the last one is a constant. That is, we reduced the problem of computing the logarithm of the non-deterministic circuit (LOGPC, Thm. 3.6) to computing $\mathbb{D}_{\mathsf{IS}}$. □

C.8 Cauchy-Schwarz Divergence

Theorem C.9. *Let p and q be two structured-decomposable and compatible PCs over variables $\mathbf{X}$. Then their Cauchy-Schwarz divergence, defined as*

$$\mathbb{D}_{\mathsf{CS}}(p \parallel q) = -\log \frac{\int_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) q(\boldsymbol{x})\, d\mathbf{X}}{\sqrt{\int_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} p^2(x)\, d\mathbf{X} \int_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} q^2(x)\, d\mathbf{X}}},$$

can be exactly computed in time $\mathcal{O}(|p|\,|q| + |p|^2 + |q|^2)$. If p and q are instead structured-decomposable but not compatible, then computing their Cauchy-Schwarz divergence is #P-hard.

Proof. ***(Tractability)*** The proof easily follows from noting that the numerator inside the log can be computed in $\mathcal{O}(|p|\,|q|)$ time as a product of two compatible circuits (Thm. 3.2); and the integrals inside the square root at the denominator can both be solved in $\mathcal{O}(|p|^2)$ and $\mathcal{O}(|q|^2)$ respectively as natural powers of structured-decomposable circuits (Thm. 3.3).

(Hardness) The proof follows by noting that if p and q are structured-decomposable, then computing the denominator inside the log can be exactly done in $|p|^2 + |q|^2$ because they are natural powers of structured-decomposable circuits (Thm. 3.3). Then $\mathbb{D}_{\mathsf{CS}}$ must be as hard as a the product of two non-compatible circuits. Therefore we can reduce MULPC (Thm. 3.1) to computing $\mathbb{D}_{\mathsf{CS}}$. □

C.9 Squared Loss Divergence

Theorem C.10. *Let p and q be two structured-decomposable and compatible PCs over variables $\mathbf{X}$. Then their squared loss, defined as*

$$\mathbb{D}_{\mathsf{SL}}(p \parallel q) = \int_{\mathsf{val}(\mathbf{X})} \left(p(\boldsymbol{x}) - q(\boldsymbol{x})\right)^2 \, d\mathbf{X},$$

can be computed exactly in time $\mathcal{O}(|p|\,|q| + |p|^2 + |q|^2)$. If p and q are structured-decomposable but not compatible, then computing their squared loss is #P-hard.

Proof. ***(Tractability)*** Proof follows by noting that the integral decomposes over the expanded square as $\int_{\mathsf{val}(\mathbf{X})} p^2(\boldsymbol{x}) \, d\mathbf{X} + \int_{\mathsf{val}(\mathbf{X})} q^2(\boldsymbol{x}) \, d\mathbf{X} - 2\int_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x})q(\boldsymbol{x}) \, d\mathbf{X}$ and as such each integral can be computed by leveraging the tractable power of structured-decomposable circuits (Thm. 3.3) and the tractable product of compatible circuits (Thm. 3.2) and therefore the overall complexity is given by the maximum of the three.

(Hardness) Proof follows by noting that the integral decomposes over the expanded square as $\int_{\mathsf{val}(\mathbf{X})} p^2(\boldsymbol{x}) \, d\mathbf{X} + \int_{\mathsf{val}(\mathbf{X})} q^2(\boldsymbol{x}) \, d\mathbf{X} - 2\int_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x})q(\boldsymbol{x}) \, d\mathbf{X}$ and that the first two terms can be computed in polytime as natural powers of structured-decomposable circuits (Thm. 3.3), hence computing $\mathbb{D}_{\mathsf{SL}}$ must be as hard as computing the product of two non-compatible circuits. Therefore we can reduce MULPC (Thm. 3.1) to computing $\mathbb{D}_{\mathsf{SL}}$. □

D Expectation-based queries

This section completes the discussion around the complex queries that can be dealt with our atlas and details the expectations briefly discussed at the end of Sec. 4.

D.1 Moments of a distribution

Proposition D.1 (Tractable moments of a PC)**.** *Let $p(\mathbf{X})$ be a smooth and decomposable PC over variables $\mathbf{X} = \{X_1, \ldots, X_d\}$, then for a set of natural numbers $\mathbf{k} = (k_1, \ldots, k_d)$, its $\mathbf{k}-$moment, defined as*

$$\int_{\mathsf{val}(\mathbf{X})} x_1^{k_1} x_2^{k_2} \ldots x_d^{k_d} p(\boldsymbol{x}) \, d\mathbf{X}$$

can be computed exactly in time $\mathcal{O}(|p|)$.

Proof. The proof directly follows from representing $x_1^{k_1} x_2^{k_2} \ldots x_d^{k_d}$ as an omni-compatible circuit comprising a single product unit over d input units, each encoding $x_i^{k_i}$, and then applying Cor. B.1. □

D.2 Probability of logical formulas

Proposition D.2 (Tractable probability of a logical formula)**.** *Let p be a smooth and decomposable PC over variables $\mathbf{X}$ and f an indicator function that represents a logical formula over $\mathbf{X}$ that can be compiled into a circuit compatible with p.*[11] *Then computing $\mathbb{P}_p[f]$ can be done in $\mathcal{O}(|p|\,|f|)$ time.*

Proof. It follows directly from Thm. 3.1, by noting that $\mathbb{P}_p[f] = \mathbb{E}_{\boldsymbol{x}\sim p(\mathbf{X})}[f(\boldsymbol{x})]$ and hence a tractable product between p and f suffices. □

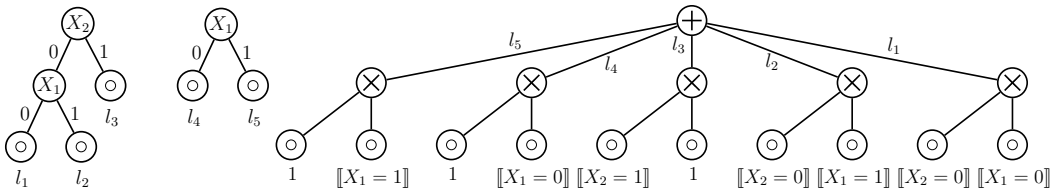


Figure 4: Encoding an additive ensemble of two trees over $\mathbf{X} = \{X_1, X_2\}$ (left) in an omni-compatible circuit over $\mathbf{X}$ (right).

D.3 Expected predictions

Example D.1 (Decision trees as circuits)**.** *Let $\mathcal{F}$ be an additive ensemble of (decision or regression) trees over variables $\mathbf{X}$, also called a forest, and computing*

$$\mathcal{F}(\boldsymbol{x}) = \sum_{\mathcal{T}_i \in \mathcal{F}} \theta_i \mathcal{T}_i(\boldsymbol{x})$$

for some input configuration $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ and each $\mathcal{T}_i$ realizing a tree, i.e., a function of the form

$$\mathcal{T}(\boldsymbol{x}) = \sum_{p_j \in \mathsf{paths}(\mathcal{T})} l_j \cdot \prod_{X_k \in \phi(p_j)} [\![x_k \leq \delta_k]\!]$$

where the outer sum ranges over all possible paths in tree $\mathcal{T}$, $l_j \in \mathbb{R}$ is the label (class or predicted real) associated to the leaf of that path, and the product is over indicator functions encoding the decision to take one branch of the tree in path p_j if x_k, the observed value for variable X_k appearing in the decision node, i.e., satisfies the condition $[\![x_k \leq \delta_k]\!]$ for a certain threshold $\delta_k \in \mathbb{R}$.

Then, it is easy to transform $\mathcal{F}$ into an omni-compatible circuit $p(\mathbf{X})$ of the form

$$p(\boldsymbol{x}) = \sum_{\mathcal{T}_i \in \mathcal{F}, p_j \in \mathsf{paths}(\mathcal{T}_\rangle)} l_j \cdot \prod_{X_k \in \phi(p_j)} [\![x_k \leq \delta_k]\!] \cdot \prod_{X'_k \notin \phi(p_j)} 1$$

with a single sum unit realizing the outer sum and as many input product units as paths in the forest, each of which realizing a fully-factorized model over $\mathbf{X}$, and weighted by l_j. One example is shown in Fig. 4.

Proposition D.3 (Tractable expected predictions of additive ensembles of trees)**.** *Let p be a smooth and decomposable PC and f an additive ensemble of k decision trees over variables $\mathbf{X}$ and bounded depth. Then, its expected predictions can be exactly computed in $\mathcal{O}(k\,|p|)$.*

Proof. Recall that an additive ensemble of decision trees can be encoded as an omni-compatible circuit. Then, proof follows from Cor. B.1. □

Proposition D.4 (Tractable expected predictions of deep regressors (regression circuits))**.** *Let p be a structured-decomposable PC over variables $\mathbf{X}$ and f be a regression circuit [23] compatible with p over $\mathbf{X}$, and defined as*

$$f_n(\boldsymbol{x}) = \begin{cases} 0 & \text{if } n \text{ is an input} \\ f_{n_\mathsf{L}}(\boldsymbol{x}_\mathsf{L}) + f_{n_\mathsf{R}}(\boldsymbol{x}_\mathsf{R}) & \text{if } n \text{ is an AND} \\ \sum_{c \in \mathsf{in}(n)} s_c(\boldsymbol{x})\,(\phi_c + f_c(\boldsymbol{x})) & \text{if } n \text{ is an OR} \end{cases}$$

where $s_c(\boldsymbol{x}) = [\![\boldsymbol{x} \in \mathsf{supp}(c)]\!]$. Then, its expected predictions can be exactly computed in $\mathcal{O}(|p|\,|h|)$ time, where h is its circuit representation as computed by Alg. 8.

Proof. Proof follows from noting that Alg. 8 outputs a polysize circuit representation h in polytime. Then, computing $\mathbb{E}_{\boldsymbol{x} \sim p(\mathbf{X})}\,[h(\boldsymbol{x})]$ can be done in $\mathcal{O}(|p|\,|h|)$ time by Thm. 3.2. □

Table 4: Sizes of the intermediate and final circuits as processed by the operators in the pipelines of the Shannon and Rényi (for $\alpha = 1.5$) entropies and Kullback-Leibler and Alpha (for $\alpha = 1.5$) divergences when computed for two input circuits p and q learned from 20 different real-world datasets as in [11].

DATASET	p	q	p^α	$q^{1-\alpha}$	$r = \log(q)$	$s = p/q$	$t = \log(s)$	$p \times q$	$p \times r$	$p \times t$	$p^\alpha \times q^{1-\alpha}$
NLTCS	2779	7174	2779	7174	26155	7202	26239	7202	26183	26239	7202
MSNBC	2765	6614	2765	6614	24111	6634	24171	6634	24131	24171	6634
KDD	4963	50377	4963	50377	184575	50417	184695	50417	184615	184695	50417
PLANTS	12909	64018	12909	64018	234661	64070	234817	64070	234713	234817	64070
AUDIO	10278	45864	10278	45864	168062	45950	168320	45950	168148	168320	45950
JESTER	6475	35369	6475	35369	129579	35479	129909	35479	129689	129909	35479
NETFLIX	5068	14636	5068	14636	53571	14706	53781	14706	53641	53781	14706
ACCIDENTS	3193	8183	3193	8183	29891	8299	30239	8299	30007	30239	8299
RETAIL	4790	14926	4790	14926	54554	14994	54758	14994	54622	54758	14994
PUMSB	4277	12461	4277	12461	45500	12595	45902	12595	45634	45902	12595
DNA	73828	856955	73828	856955	3141981	857029	3142203	857029	3142055	3142203	857029
KOSAREK	5115	12988	5115	12988	47354	13106	47708	13106	47472	47708	13106
MSNWEB	4859	9025	4859	9025	32675	9175	33125	9175	32825	33125	9175
BOOK	7718	12731	7718	12731	45985	12943	46621	12943	46197	46621	12943
MOVIE	8309	11732	8309	11732	42374	11926	42956	11926	42568	42956	11926
WEBKB	10598	13397	10598	13397	47859	13653	48627	13653	48115	48627	13653
CR52	10912	14348	10912	14348	51094	14546	51688	14546	51292	51688	14546
C20NG	11386	14630	11386	14630	52120	14886	52888	14886	52376	52888	14886
BBC	13884	17016	13884	17016	60857	17282	61655	17282	61123	61655	17282
AD	17744	21676	17744	21676	76870	21920	77602	21920	77114	77602	21920

Table 5: Times in seconds to compute the Shannon entropy (ENT), the cross-entropy (XENT), Kullback-Leibler (KLD), Alpha (for $\alpha = 1.5$) divergence, Rényi entropy (RényiEnt), and Cauchy-Schwarz divergence (CSDiv) over the circuits learned from 20 different real-world datasets by either using the algorithm distilled by our pipelines (see Tab. 4 and Fig. 5) compared to the custom and highly-optimized implementations of the same ENT [44] and KLD [28] algorithms as available in Juice.jl [12].

DATASET	ENT		KLD		XENT		ALPHADIV		RÉNYIENT		CSDIV	
	OURS	JUICE	OURS	JUICE	OURS	JUICE	OURS	JUICE	OURS	JUICE	OURS	JUICE
NLTCS	0.143	0.001	0.830	0.207	0.422	-	0.140	-	0.013	-	0.300	-
MSNBC	0.109	0.001	0.369	0.182	0.297	-	0.105	-	0.018	-	0.227	-
KDD	0.157	0.001	3.154	0.790	2.180	-	0.885	-	0.016	-	1.136	-
PLANTS	0.679	0.005	3.983	3.909	3.739	-	1.160	-	0.088	-	1.572	-
AUDIO	0.406	0.003	2.736	1.681	1.873	-	0.537	-	0.029	-	0.771	-
JESTER	0.764	0.003	1.019	0.432	0.805	-	0.351	-	0.024	-	0.476	-
NETFLIX	0.106	0.002	0.352	0.175	0.264	-	0.100	-	0.017	-	0.201	-
ACCIDENTS	0.055	0.001	0.207	0.039	0.542	-	0.091	-	0.009	-	0.124	-
RETAIL	0.108	0.001	0.508	0.153	0.415	-	0.184	-	0.013	-	0.197	-
PUMSB	0.092	0.001	0.701	0.133	0.316	-	0.119	-	0.012	-	0.214	-
DNA	4.365	0.027	64.664	220.377	52.997	-	15.609	-	0.255	-	22.901	-
KOSAREK	0.182	0.002	0.477	0.106	0.379	-	0.139	-	0.011	-	0.735	-
MSNWEB	0.128	0.002	0.261	0.047	0.211	-	0.342	-	0.015	-	0.135	-
BOOK	0.086	0.003	0.215	0.036	0.202	-	0.075	-	0.020	-	0.115	-
MOVIE	0.272	0.002	0.443	0.063	0.373	-	0.172	-	0.015	-	0.194	-
WEBKB	0.138	0.003	0.241	0.031	0.164	-	0.079	-	0.023	-	0.098	-
CR52	0.141	0.004	0.260	0.035	0.188	-	0.087	-	0.031	-	0.143	-
C20NG	0.118	0.003	0.264	0.034	0.194	-	0.088	-	0.032	-	0.101	-
BBC	0.205	0.005	0.308	0.037	0.225	-	0.110	-	0.038	-	0.189	-
AD	0.193	0.007	0.346	0.046	0.281	-	0.151	-	0.031	-	0.207	-

Algorithm 8 RGCTOCIRCUIT($r, \mathsf{cache}_r, \mathsf{cache}_s$)

Input: a regression circuit r over variables $\mathbf{X}$ and two caches for memoization (i.e., cache_r and cache_s).
Output: its representation as a circuit $p(\mathbf{X})$.
if $r \in \mathsf{cache}_r$ **then return** $\mathsf{cache}_r(r)$
if r is an input gate **then**
  $p \leftarrow \textsc{Input}(0, \phi(r))$
else if r is a sum gate **then**
  $n \leftarrow \{\}$
  for $i = 1$ **to** $|\mathsf{in}(r)|$ **do**
    $n \leftarrow n \cup \{\textsc{Support}(r_i, \mathsf{cache}_s)\} \cup \{\textsc{RGCtoCircuit}(r_i, \mathsf{cache}_r)\}$
  $p \leftarrow \textsc{Sum}(n, \{\theta_i, 1_1, \ldots, 1_{\mathsf{in}(p)}\}_{i=1}^{|\mathsf{in}(r)|})$
else if r is a product gate **then**
  for $i = 1$ **to** $|\mathsf{in}(r)|$ **do**
    $p \leftarrow \textsc{Product}(\{\textsc{RGCtoCircuit}(r_i, \mathsf{cache}_r)\} \cup \{\textsc{Support}(r_j, \mathsf{cache}_s)\}_{j \neq i})$
$\mathsf{cache}_r(r) \leftarrow p$
return p

```
function kld(p, q)
    r = quotient(p, q)
    s = log(r)
    t = product(p, s)
    return integrate(t)
end
```

```
function xent(p, q)
    r = log(q)
    s = product(p, r)
    return -integrate(s)
end
```

```
function csdiv(p, q)
    r = product(p, q)
    s = real_pow(p, 2.0)
    t = real_pow(q, 2.0)
    a = integrate(r)
    b = integrate(s)
    c = integrate(t)
    return -log(a / sqrt(b * c))
end
```

```
function ent(p)
    q = log(p)
    r = product(p, q)
    return -integrate(s)
end
```

```
function alphadiv(p, q, alpha=1.5)
    r = real_pow(p, alpha)
    s = real_pow(q, 1.0-alpha)
    t = product(r, s)
    return log(integrate(t)) / (1.0-alpha)
end
```

Figure 5: The modular operators defined in Sec. 3 can be easily composed to implement tractable algorithms for novel query classes. Here we show the code snippet for five queries: Kullback-Leibler divergence (`kld`), Cross Entropy (`xent`), Entropy (`ent`), Alpha divergence (`alphadiv`), and Cauchy-Schwarz divergence (`csdiv`).

E Experiments

Generated PCs All adopted PCs were generated by running Strudel [11] on the twenty density estimation benchmarks [48]. For every dataset, we ran Strudel twice with 200 and 500 iterations, respectively. All other hyperparameters were selected following Dang et al. [11].

Server specifications All our experiments were run on a server with 72 CPUs, 512G Memory, and 2 TITAN RTX GPUs.

Implementations Code snippet for the five adopted queries (i.e., Kullback-Leibler divergence, Cross Entropy, Entropy, Alpha divergence, and Cauchy-Schwarz divergence) are shown in Fig. 5. Note that they are simple compositions of the modular operators introduced in Sec. 3.

[11] E.g. by compiling it into an SDD [14, 3] whose vtree encodes the hierarchical scope partitioning of p.