A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference

Antonio Vergari[1] **YooJung Choi**[1] **Anji Liu**[1] **Stefano Teso**[2] **Guy Van den Broeck**[1]

[1]Computer Science Department, University of California, Los Angeles, USA,
`{aver|yjchoi|liuanji|guyvdb}@cs.ucla.edu`
[2]Department of Computer Science, University of Trento, Italy, `stefano.teso@unitn.it`

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, powers, logarithms, and exponentials—in terms of sufficient structural constraints of the circuits they operate on. 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 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 for important applications such as approximate inference [39], model compression [25], explainable AI [21, 42, 46] and algorithmic bias detection [20, 5, 7].

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) [25]. 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) [32]—by Shih and Ermon [39]. In the current paradigm, if one were to trace the tractability of a query that has not yet been investigated but still involves the same "building blocks" such as logarithms, integrals and products over distributions, for instance Rényi's alpha divergence [36], they would need to derive a novel custom algorithm for each model class and prove its tractability from scratch.

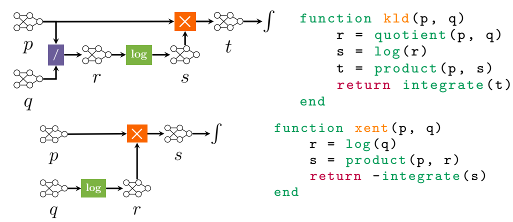


Figure 1: Computational pipelines of the KLD (top) and cross entropy (bottom) over two distributions p and q encoded as circuits, as are the intermediate computations (r, s, t). Their corresponding implementations in Julia are shown on their right.

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 [6] as they subsume many tractable generative models—probabilistic circuits such as Chow-Liu trees [8], hidden Markov models (HMMs) [35], sum-product networks (SPNs) [34], and other deep mixture

Accepted for the 4th Workshop on Tractable Probabilistic Modeling (TPM 2021).

models—as well as discriminative ones—including decision trees [22, 9] and deep regressors [20]—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 code 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 (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 results of complex information-theoretic queries including several widely used entropies and divergences (Tab. 2); and (4) an implementation of these operators in the Juice circuit library [11].[1] 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 [6, 45]. As such, circuits provide *a language for building and reasoning about tractable representations*.

We introduce the basic rules of this language by distinguishing between general circuits and those encoding probability distributions, as some operators in Sec. 3 might 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 $\mathsf{val}(\mathbf{X})$.

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 $\sum_{c\in\mathsf{in}(n)} \theta_c p_c(\phi(n))$ where $\theta_c \in \mathbb{R}$ are the sum parameters; while if it is a product unit, it encodes $\prod_{c\in\mathsf{in}(n)} p_c(\phi(n))$. 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: $\mathsf{supp}(p) = \{\boldsymbol{x} \in \mathsf{val}(\mathbf{X}) \mid p(\boldsymbol{x}) \neq 0\}$.

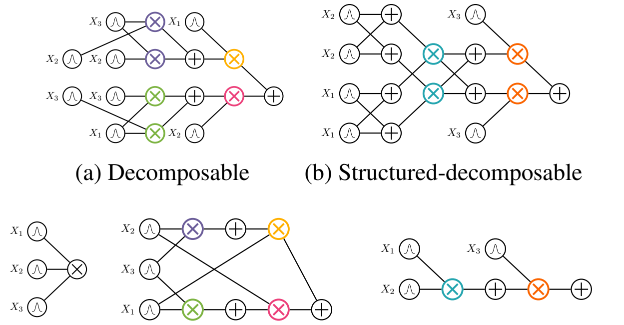
Circuits can be understood as compact representations of polynomials with exponentially many terms, whose indeterminates are the functions encoded by the input units. These functions are assumed to be simple enough to allow tractable computations of the operations discussed in this paper. Fig. 2 shows some examples of circuits. A *probabilistic circuit* (PC) [6] 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 a PC to have 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 [43, 44].

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

[1] Code will be released upon acceptance.



(a) Decomposable (b) Structured-decomposable

(c) Omni-compatible circuits for $p(\mathbf{X}) = p(X_1)p(X_2)p(X_3)$

Figure 2: *Examples of circuit representations* with different structural properties. The feedforward order is from left to right; input units are labeled by their scope; and sum parameters are omitted for visual clarity. Product units of the rearranged omni-compatible factorization (bottom) are color-coded with those of matching scope in the top circuits.

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. See Choi et al. [6] 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 [26] 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 [14, 6].

Proposition 2.1 (Tractable integration)**.** *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 $\int_{\boldsymbol{z} \in \mathsf{val}(\mathbf{Z})} p(\boldsymbol{y}, \boldsymbol{z}) d\mathbf{Z}$, where $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$, can be computed exactly in $\Theta(|p|)$ time.*

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 [40]. 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 later.

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

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

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 a circuit [12, 6]. 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 [8] and thin junction trees [1], can efficiently be represented as a smooth, deterministic, and decomposable PC via *compilation* [12, 10]. Probabilistic sentential decision diagrams (PSDDs) [23] are deterministic and structured-decomposable PCs that can be efficiently learned from data [10].

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 need the output circuit of one operator to satisfy the structural properties required for the inputs of the other. As such, for each of these operations we characterize (1) its tractability in terms of the structural properties of its input circuits, 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 expect the output circuits to at least retain decomposability (see Prop. 2.1). If all operators in a pipeline can be computed tractably, a simple tractable algorithm can then be distilled for it. Tab. 1 summarizes all our results.

Sum of Circuits. The operation of summing two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ is defined as computing $s(\mathbf{X}) = \theta_1 \cdot p(\mathbf{Z}) + \theta_2 \cdot q(\mathbf{Y})$ 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 can be applied to any input circuits, regardless of structural assumptions, and it preserves several properties (see Prop. A.1). 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 [38], 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. For the general case, Shen et al. [38] 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 3.1 (Tractable product)**.** *Let p and q be two compatible circuits over variables $\mathbf{X}$ Then, computing their product $m(\mathbf{X})$ 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).*

The proof is by construction, showing that computing (1) the product of two smooth sum units p and q and (2) the product of two compatible product units p and q takes $\mathcal{O}(|\mathsf{in}(p)|\,|\mathsf{in}(q)|)$ time, given the product circuits for each pair of child units of p and q. Then the overall time complexity is $\mathcal{O}(|p|\,|q|)$ by recursion. We refer to Alg. 1 in the Appendix for a detailed pseudocode, also applicable when p and q have different scopes.

For (1), suppose 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 as the weighted sum of $|\mathsf{in}(p)| \cdot |\mathsf{in}(q)|$ circuits that represent the products of pairs of their inputs: $m(\boldsymbol{x}) = \sum_{i\in\mathsf{in}(p)} \sum_{j\in\mathsf{in}(q)} \theta_i \theta'_j (p_i q_j)(\boldsymbol{x})$. Note that this Cartesian product of units is a deterministic sum unit if both p and q are deterministic, as $\mathsf{supp}(p_i q_j) = \mathsf{supp}(p_i) \cap \mathsf{supp}(q_j)$ are disjoint for different i, j.

For (2), suppose 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: $m(\boldsymbol{x}) = p_1(\boldsymbol{x}_1) q_1(\boldsymbol{x}_1) \cdot p_2(\boldsymbol{x}_2) q_2(\boldsymbol{x}_2) = (p_1 q_1)(\boldsymbol{x}_1) \cdot (p_2 q_2)(\boldsymbol{x}_2)$. Note that 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.

Powers of Circuits. 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 often needed to compute generalizations of the entropy and related divergences. Let us first consider natural powers ($\alpha \in \mathbb{N}$) which can be defined even for general circuits, not just PCs.

Theorem 3.2 (Natural power)**.** *If p is a structured-decomposable circuit, then for any $\alpha \in \mathbb{N}$, its power can be encoded as a compatible circuit in $\mathcal{O}(|p|^\alpha)$ time.*

The proof easily follows by directly applying the product operation repeatedly. We now turn our attention to non-natural $\alpha \in \mathbb{R}$ and tractable α-powers of PCs. First, as zero raised to a negative power is undefined, we instead consider the *restricted α-power* of a PC, denoted as $a(\boldsymbol{x})|_{\mathsf{supp}(p)}$ and

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

Table 1: ***Tractability of simple circuit operations***. Tractable conditions on inputs translate to conditions 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.

Operation		Tractability			
		Input conditions	Output conditions	Time Complexity	
SUM	$\theta_1 p + \theta_2 q$	(+Cmp)	(+SD)	$\mathcal{O}(\|p\|+\|q\|)$	(Prop. A.1)
PRODUCT	$p \cdot q$	Cmp (+Det, +SD)	Dec (+Det, +SD)	$\mathcal{O}(\|p\|\|q\|)$	(Thm. 3.1)
POWER	$p^n, n \in \mathbb{N}$	SD (+Det)	SD (+Det)	$\mathcal{O}(\|p\|^n)$	(Thm. 3.2)
	$p^\alpha, \alpha \in \mathbb{R}$	Sm, Dec, Det (+SD)	Sm, Dec, Det (+SD)	$\mathcal{O}(\|p\|)$	(Thm. 3.3)
QUOTIENT	p/q	Cmp; q Det (+p Det,+SD)	Dec (+Det,+SD)	$\mathcal{O}(\|p\|\|q\|)$	(Thm. 3.4)
LOG	$\log(p)$	Sm, Dec, Det	Sm, Dec	$\mathcal{O}(\|p\|)$	(Thm. 3.5)
EXP	$\exp(p)$	linear	SD	$\mathcal{O}(\|p\|)$	(Prop. 3.1)

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 write it as $p^\alpha(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$, where $[\![\cdot]\!]$ stands for indicator functions. 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 3.3 (Tractable real powers)**.** *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.*

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 each input unit of p and represent it as a single new unit. For a smooth and 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

$$\Big(\sum_{i \in \mathsf{in}(p)} \theta_i p_i(\boldsymbol{x})\Big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] = \sum_{i \in \mathsf{in}(p)} \theta_i^\alpha p_i^\alpha(\boldsymbol{x}) [\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!].$$

Above construction is made possible by determinism: only one p_i will be non-zero for any input $\boldsymbol{x}$. Thus, the power circuit retains the same structure as 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

$$\big(p_1(\boldsymbol{x}_1) \cdot p_2(\boldsymbol{x}_2)\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] = \big(p_1(\boldsymbol{x}_1)\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p_1)]\!] \cdot \big(p_2(\boldsymbol{x}_2)\big)^\alpha [\![\boldsymbol{x} \in \mathsf{supp}(p_2)]\!].$$

Note that even this construction preserves the structure of p, and thus its scope partitioning is retained throughout the whole algorithm. Hence, if p was also structured-decomposable, then a would also be structured-decomposable. Alg. 2 in the Appendix illustrates the whole algorithm in detail.

Quotients of Circuits. 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.

Theorem 3.4 (Tractable quotient)**.** *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 $\mathbf{X}$ in $\mathcal{O}(|p|\,|q|)$ time. Moreover, if p is also deterministic, then the quotient circuit is deterministic.*

We know from Thm. 3.3 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. 3.1 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. 3.1).

Logarithms of PCs. 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 $l(\boldsymbol{x})|_{\mathsf{supp}(p)}$ and equal to $\log p(\boldsymbol{x})$ if $\boldsymbol{x} \in \mathsf{supp}(p)$ and 0 otherwise.

Theorem 3.5 (Tractable logarithms)**.** *Let p be a smooth, deterministic and decomposable PC over variables $\mathbf{X}$. Then its restricted logarithm, defined as $l(\boldsymbol{x})|_{\mathsf{supp}(p)} = \log p(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$, can be represented as a smooth* *and decomposable circuit that shares the scope partitioning of p in $\mathcal{O}(|p|)$ time. Moreover, if p is structured decomposable, then so is its logarithm circuit.*

Note that while the input of the logarithm operator must be a PC, its output can be a general circuit. 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 smooth and 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 $\{\theta_i\}_{i=1}^{|\mathsf{in}(p)|}$:

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

For a decomposable product unit $p(\boldsymbol{x}) = p_1(\boldsymbol{x}_1) p_2(\boldsymbol{x}_2)$, its logarithm circuit can be decomposed as the sum of logarithm circuits of its child units:

$$
\begin{aligned}
& l(\boldsymbol{x})|_{\mathsf{supp}(\boldsymbol{x})} = \log\left(p_1(\boldsymbol{x}_1) p_2(\boldsymbol{x}_2)\right) \cdot [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] \\
& = \log p_1(\boldsymbol{x}_1) [\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!] [\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!] \\
& \quad + \log p_2(\boldsymbol{x}_2) [\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!] [\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!] \\
& = l(\boldsymbol{x}_1)|_{\mathsf{supp}(p_1)} [\![\boldsymbol{x}_2 \in \mathsf{supp}(p_2)]\!] \\
& \quad + l(\boldsymbol{x}_2)|_{\mathsf{supp}(p_2)} [\![\boldsymbol{x}_1 \in \mathsf{supp}(p_1)]\!]. \qquad (1)
\end{aligned}
$$

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. 3 in the Appendix illustrates the algorithm in detail, showing that the construction of these support circuits can be done in linear time by caching intermediate sub-circuits. 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. 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 (see Fig. 3 for an illustrative example). This implies that whenever we have to 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).

Exponentials of Circuits. 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 [24]. Similar to

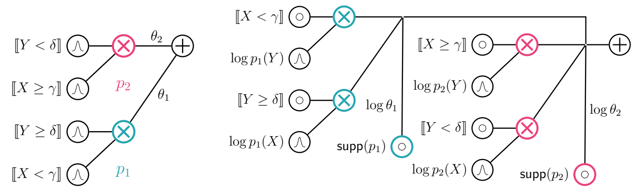


Figure 3: Building the logarithmic circuit (right) for a deterministic PC (left) whose input units are labeled by their supports. A single sum unit is introduced over smoothed product units and additional dummy input units; units with shared support have the same color.

the logarithm, building a decomposable circuit that encodes an exponential of a circuit is hard in general. 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 cannot be broken down to either a sum or a 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)**.** *Let p be a linear circuit over variables $\mathbf{X}$, i.e., $p(\mathbf{X}) = \sum_i \theta_i \cdot X_i$. Then $\exp(p(\mathbf{X}))$ can be represented as an omni-compatible circuit with a single product unit in $\mathcal{O}(|p|)$ time.*

The proof follows immediately by the properties of exponentials of sums. Alg. 4 in the Appendix formalizes it. 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 [28, 49]. This is the largest class of tractable exponentials we know so far, and enlarging its boundaries is an interesting open problem.

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*. We 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 then discuss how our discovered tractable circuit classes subsume some previously known results. Tab. 2 summarizes our results.

We now showcase how a short tractability proof can be easily distilled, using Rényi's α-divergence[4] [36] as an exam-

[4]Several alternative formulations of α-divergences can be found in the literature such as Amari's [27] and Tsallis's [30] divergences. However, as they share the same core operations—real powers and products—our results easily extend to them as well.

Table 2: Tractability of information-theoretic queries given some conditions over the input circuits. See Appendix B.

	Query	Conditions
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x})$	Cmp, q Det
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}), \alpha \in \mathbb{N}$	SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}), \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x}))$	Cmp, Det
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}), \alpha \in \mathbb{N}$	Cmp, q Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}), \alpha \in \mathbb{R}$	Cmp, Det
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1]$	Cmp, Det
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x})}{\sqrt{\int p^2(\boldsymbol{x}) \int q^2(\boldsymbol{x})}}$	Cmp
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2$	Cmp

ple. 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. 3.1). To conclude the proof, we need to compute two compatible circuits r and s in polytime, which can be done according to Thm. 3.3 or Thm. 3.2 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. Hence, t can 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. 3.1). □

We leave the formal theorems and proofs for the other queries listed in Tab. 2 to Sec. B 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.*

Information-theoretic queries. Smooth, decomposable and deterministic PCs enable the exact computation of Shannon entropy 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 [39]. For non-deterministic PCs we can employ the tractable computation of Rényi entropy of order α [36], which recovers Shannon Entropy for $\alpha \to 1$. As the logarithm is taken after integration of the power circuit, the tractability follows directly from that of the power operation (Thm. 3.2 and 3.3). Moreover, using our atlas, the cross entropy can be tractably computed in $\mathcal{O}(|p|\,|q|)$ if p and q are deterministic and compatible. 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.

Divergences. Liang and Van den Broeck [25] proposed an efficient algorithm to compute the KLD tailored for PSDDs.[6] 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 $\alpha = 2$ [15]. As Thm. 4.1 states, they are tractable for compatible and deterministic PCs, as is the Itakura-Saito divergence [47]. For non-deterministic PCs, we characterize the tractability of the squared loss and the Cauchy-Schwarz divergence [18]. The latter has applications in mixture models for approximate inference [41] and has been derived in closed-form only for mixtures of simple parametric forms like Gaussians [19], Weibull and Rayligh distributions [29]. Our results generalize them to deep mixture models [34].

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. 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. [3] proposed an algorithm tailored to formulas f over binary variables, encoded as SDDs [13] 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. Lastly, if f encodes constraints over the output distribution of a deep network we retrieve the *semantic loss* [48]. If f encodes a classifier or a regressor, then $\mathbb{E}_{p}[f]$ refers to computing its

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

[6] Note that our tractability proof is only a few lines long and does not require the ad-hoc algebraic derivations of [25].

expected predictions w.r.t. p [21]. Our results generalize the results reported in Van den Broeck et al. [42] such as computing the expectations of decision trees and their ensembles [22] as well as those of *deep regression circuits* [20].[7]

5 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 [14] 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 [16] and PSDDs later [38]. Despite the different syntax, both model classes can be encoded as structured-decomposable and deterministic circuits in our language [6].[8]

Our property-driven analysis closes many open questions about the tractability of queries for several model classes that are special cases of circuits. At the same time, it opens new ones, for instance how to characterize 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.

References

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

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

[3] A. Choi, G. Van den Broeck, and A. Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *IJCAI*, 2015.

[4] Y. Choi, A. Darwiche, and G. V. den Broeck. Optimal feature selection for decision robustness in bayesian networks. In *IJCAI*, pages 1554–1560, 2017.

[5] Y. Choi, G. Farnadi, B. Babaki, and G. Van den Broeck. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. In *AAAI*, 2020.

[6] Y. Choi, A. Vergari, and G. Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020.

[7] Y. Choi, M. Dang, and G. Van den Broeck. Group fairness by probabilistic modeling with latent fair decisions. In *AAAI*, Feb 2021.

[8] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEETIT*, 1968.

[9] A. H. C. Correia, R. Peharz, and C. P. de Campos. Joints in random forests. In *NeurIPS*, 2020.

[10] M. Dang, A. Vergari, and G. Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *PGM*, 2020.

[11] M. Dang, P. Khosravi, Y. Liang, A. Vergari, and G. Van den Broeck. Juice: A julia package for logic and probabilistic circuits. In *AAAI (Demo Track)*, Feb 2021.

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

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

[14] A. Darwiche and P. Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

[15] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *ISR*, 2002.

[16] M. Jaeger. Probabilistic decision graphscombining verification and ai techniques for probabilistic inference. *IJUFKS*, 2004.

[17] M. Jaeger, J. D. Nielsen, and T. Silander. Learning probabilistic decision graphs. *IJAR*, 2006.

[18] R. Jenssen, J. C. Principe, D. Erdogmus, and T. Eltoft. The cauchyschwarz divergence and parzen windowing: Connections to graph theory and mercer ker-

[7] 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. 5 in the Appendix.

[8] PDGs and PSDDs entangle compatibility with determinism in terms of special notions of hierarchical scope partitioning, namely pseudo forests [16, 17] and vtrees [33], respectively. As we showed in Thm. 3.1, compatibility is sufficient for tractable multiplication, and as discussed in the previous section many algorithms tailored for PSDDs [3, 38, 20] can be generalized to *non-deterministic* distributions in our framework.

nels. *Journalof the Franklin Institute*, 343(6):614629, 2006.

[19] K. Kampa, E. Hasanbelliu, and J. C. Principe. Closed-form cauchy-schwarz pdf divergence for mixture of gaussians. In *IJCNN*, 2011.

[20] P. Khosravi, Y. Choi, Y. Liang, A. Vergari, and G. Van den Broeck. On tractable computation of expected predictions. In *NeurIPS*, pages 11169–11180, 2019.

[21] P. Khosravi, Y. Liang, Y. Choi, and G. V. den Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. In *IJCAI*, pages 2716–2724, 2019.

[22] P. Khosravi, A. Vergari, Y. Choi, Y. Liang, and G. V. 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.

[23] D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *KR*, 2014.

[24] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

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

[26] G. J. McLachlan, S. X. Lee, and S. I. Rathnayake. Finite mixture models. *Annual rev. of stat. and its app.*, 2019.

[27] T. P. Minka. Expectation propagation for approximate bayesian inference. In *UAI*, 2001.

[28] S. Moral, R. Rumí, and A. Salmerón. Mixtures of truncated exponentials in hybrid bayesian networks. In *ECSQARU*, 2001.

[29] F. Nielsen. Closed-form information-theoretic divergences for statistical mixtures. In *ICPR*, 2012.

[30] M. Opper, O. Winther, and M. J. Jordan. Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6(12), 2005.

[31] U. Oztok, A. Choi, and A. Darwiche. Solving $\mathrm{PP}^{\mathrm{PP}}$-complete problems using knowledge compilation. In *KR*, 2016.

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

[33] K. Pipatsrisawat and A. Darwiche. New compilation languages based on structured decomposability. In *AAAI*, 2008.

[34] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *UAI*, 2011.

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

[36] A. Rényi et al. On measures of entropy and information. In *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, 1961.

[37] A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, 2014.

[38] Y. Shen, A. Choi, and A. Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *NeurIPS*, pages 3936–3944, 2016.

[39] A. Shih and S. Ermon. Probabilistic circuits for variational inference in discrete graphical models. In *NeurIPS*, 2020.

[40] A. Shih, G. V. den Broeck, P. Beame, and A. Amarilli. Smoothing structured decomposable circuits. In *NeurIPS*, pages 11412–11422, 2019.

[41] L. Tran, M. Pantic, and M. P. Deisenroth. Cauchy-schwarz regularized autoencoder. *arXiv:2101.02149*, 2021.

[42] G. Van den Broeck, A. Lykov, M. Schleich, and D. Suciu. On the tractability of shap explanations. In *AAAI*, 2021.

[43] A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *ECML PKDD*, pages 343–358. Springer, 2015.

[44] A. Vergari, N. Di Mauro, and F. Esposito. Visualizing and understanding sum-product networks. *MLJ*, 2019.

[45] A. Vergari, N. Di Mauro, and G. Van den Broeck. Tractable probabilistic models: Representations, algorithms, learning, and applications. *Tutorial at UAI*, 2019.

[46] E. Wang, P. Khosravi, and G. V. d. Broeck. Probabilistic sufficient explanations. *IJCAI*, 2021.

[47] B. Wei and J. D. Gibson. Comparison of distance measures in discrete spectral modeling. Master's thesis, Citeseer, 2001.

[48] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, volume 80, pages 5498–5507. PMLR, 2018.

[49] Z. Zeng, P. Morettin, F. Yan, A. Vergari, and G. V. den Broeck. Scaling up hybrid probabilistic inference with logical and arithmetic constraints via message passing. In *ICML*, 2020.

A CIRCUIT OPERATIONS

This section contains the complete algorithms for the operators summarized in Tab. 1—sums, products, powers, logarithms, and exponentials. 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. 3.1 we assume tractable product of input units sharing the same scope and for Thm. 3.3 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.

We start by introducing some useful sub-routines.

Support 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 $[\![\boldsymbol{x} \in \mathsf{supp}(l)]\!]$.

Uniform distribution circuit. We can build a deterministic and omni-compatible PC that encodes a (possibly unnormalized) uniform distribution over binary variables $\mathbf{X} = \{X_1, \dots, X_n\}$: 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 $\mathsf{val}(X_i)$.

Sum of circuits. The hardness of the sum of two circuits to yield a deterministic circuit has been proven by Shen et al. [38] in the context of arithmetic circuits (ACs) [14]. 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 [37]. A sum of circuits will preserve decomposability and related properties as the next proposition details.

Proposition A.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 [12, 40], 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.$\mathbf{Z} \setminus \mathbf{Y}$). 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. □

Algorithm 4 EXPONENTIAL(p)

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

B COMPLEX QUERIES

This section collects the complete tractability results in Tab. 2. Note that the proofs are succinct thanks to our atlas which allows to define a tractable model class effortlessly. Below, p and q denote PCs over variables $\mathbf{X}$, unless specified otherwise.

Cross Entropy Suppose p and q are compatible, and q is deterministic. Then their cross entropy, defined as $-\int_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) \log(q(\boldsymbol{x})) d\mathbf{X}$, restricted to the support of q can be exactly computed in $\mathcal{O}(|p|\,|q|)$ time. From Thm. 3.5, we can compute the logarithm of q in $\mathcal{O}(|q|)$ time as a circuit that is compatible with q and hence with p. Then we can multiply p and $\log q$ according to Thm. 3.1 in $\mathcal{O}(|p|\,|q|)$ time, returning a circuit that is still smooth and decomposable, hence we can tractably compute its partition function.

Algorithm 1 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**
    $n', s' \leftarrow \text{MULTIPLY}(p, q_j, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\};\ s \leftarrow s \vee s'$
  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**
    $n', s' \leftarrow \text{MULTIPLY}(p_i, q, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\};\ s \leftarrow s \vee s'$
  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**
    $n', s' \leftarrow \text{MULTIPLY}(p_i, q_i, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\};\ s \leftarrow s \wedge s'$
  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**
    $n', s' \leftarrow \text{MULTIPLY}(p_i, q_j, \mathsf{cache})$
    $n \leftarrow n \cup n'; w \leftarrow w \cup \{\theta_i \theta'_j\}; s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, w)$ **else** $m \leftarrow null$
$\mathsf{cache}(p, q) \leftarrow (m, s)$
return m, s

Entropy If p is smooth, deterministic, and decomposable, 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. Again, using Thm. 3.5 we can compute the logarithm of p in $\mathcal{O}(|p|)$ time as a smooth and decomposable PC with the same support partitioning as p. Thus, multiplying p and $\log p$ according to Alg. 1 yields a smooth and decomposable circuit in $\mathcal{O}(|p|)$ time exploiting the shared support structure; then we can take its partition function in time linear in its size.

Mutual Information 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

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

Algorithm 2 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** $a \leftarrow \textsc{Sum}(\{\text{POWER}(p_i, \alpha, \mathsf{cache})\}_{i=1}^{|\mathsf{in}(p)|}), \{\theta_i^{\alpha}\}_{i=1}^{|\mathsf{in}(p)|})$

else if p is a product unit **then** $a \leftarrow \textsc{Product}(\{\text{POWER}(p_i, \alpha, , \mathsf{cache})\}_{i=1}^{|\mathsf{in}(p)|})$
$\mathsf{cache}(p) \leftarrow a$
return a

$\mathbf{X}$ and $\mathbf{Y}$, defined as $\int_{\mathsf{val}(\mathbf{Z})} p(\boldsymbol{x}, \boldsymbol{y}) \log \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x}) \cdot p(\boldsymbol{y})} d\mathbf{X} d\mathbf{Y}$, 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] From Thm. 3.5 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. 3.1 to yield circuits of size $\mathcal{O}(|p|)$. As these are still smooth and decomposable, we can compute their partition functions and compute the mutual information between $\mathbf{X}$ and $\mathbf{Y}$ w.r.t. p.

Kullback-Leibler (KL) Divergence Suppose p and q are deterministic and compatible. Then, their intersectional KL Divergence, defined as $\int_{\mathsf{supp}(p) \cap \mathsf{supp}(q)} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\mathbf{X}$, can exactly be computed in $\mathcal{O}(|p|\,|q|)$ time. This can be concluded directly from the tractability of cross entropy and entropy, by observing that the KLD is equal to the difference between the entropy of p and cross entropy of p and q.

Rényi Entropy The Rényi entropy of order α of a PC p is defined as $\frac{1}{1-\alpha} \log \int_{\mathsf{supp}(p)} p^{\alpha}(\boldsymbol{x}) d\mathbf{X}$. For $\alpha \in \mathbb{N}$, the Rényi entropy of a structured-decomposable PC p can be computed in $\mathcal{O}(|p|^{\alpha})$ time, by computing the natural power circuit of p in $\mathcal{O}(|p|^{\alpha})$ time according to Thm. 3.2. For $\alpha \in \mathbb{R}_{+}$, if p is smooth, decomposable, and deterministic, then its Rényi entropy can be computed in $\mathcal{O}(|p|)$ time, by computing the power circuit of p in $\mathcal{O}(|p|)$ time using Thm. 3.3.

[10]This structural property of circuits is also known as marginal determinism [6] and has been introduced in the context of marginal MAP inference and the computation of same-decision probabilities of Bayesian classifiers [31, 4].

Algorithm 3 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 $\log(p(\mathbf{X}))$
if $p \in \mathsf{cache}_l$ **then return** $\mathsf{cache}_l(p)$
if p is an input unit **then**
  $l \leftarrow \text{INPUT}(\log(p_{|\mathsf{supp}(p)}), \phi(p))$
else if p is a sum unit **then**
  $n \leftarrow \{\}$
  for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
    $n \leftarrow n \cup \{\text{SUPPORT}(p_i, \mathsf{cache}_s)\} \cup \{\text{LOGARITHM}(p_i, \mathsf{cache}_l)\}$
  $l \leftarrow \text{SUM}(n, \{\log\theta_1, 1, \log\theta_2, 1, \ldots, \log\theta_{|\mathsf{in}(p)|}, 1\})$
else if p is a product unit **then**
  $n \leftarrow \{\}$
  for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
    $n \leftarrow n \cup \{\text{PRODUCT}(\{\text{LOGARITHM}(p_i, \mathsf{cache}_l)\} \cup \{\text{SUPPORT}(p_j, \mathsf{cache}_s)\}_{j \neq i})\}$
  $l \leftarrow \text{SUM}(n, \{1\}_{i=1}^{|\mathsf{in}(p)|})$
$\mathsf{cache}_l(p) \leftarrow l$
return l

Alpha Divergence Let p and q be compatible PCs over variables $\mathbf{X}$. Then their Rényi's α-divergence, defined as

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

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. The proof easily follows from first computing the power circuit of p and q according to Thm. 3.3 or Thm. 3.2 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. 3.1) 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.

Itakura-Saito Divergence 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

$$\int_{\mathsf{supp}(p) \cap \mathsf{supp}(q)} \left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - 1 \right) d\mathbf{X},$$

can be exactly computed in $\mathcal{O}(|p| |q|)$ time. Note that the integral decomposes into three integrals over the inner sum: $\int \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\mathbf{X} - \int \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\mathbf{X}$ - $\int 1 \, d\mathbf{X}$.. Then, the first integral over the quotient can be solved $\mathcal{O}(|p| |q|)$ (Thm. 3.4); the second integral over the log of a quotient of two PCs can be computed in time $\mathcal{O}(|p| |q|)$ (Thm. 3.4 and 3.5) and finally the last one integrates to the dimensionality of $|\mathsf{supp}(p) \cap \mathsf{supp}(q)|$, which we assume to exist.

Algorithm 5 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**
  $p \leftarrow \text{INPUT}(0, \phi(r))$
else if r is a sum gate **then**
  $n \leftarrow \{\}$
  for $i = 1$ **to** $|\mathsf{in}(r)|$ **do**
    $n \leftarrow n \cup \{\text{SUPPORT}(r_i, \mathsf{cache}_s)\} \cup \{\text{RGCTOCIRCUIT}(r_i, \mathsf{cache}_r)\}$
  $p \leftarrow \text{SUM}(n, \{\theta_i, 1_1, \ldots, 1_{\mathsf{in}(p)}\}_{i=1}^{|\mathsf{in}(r)|})$
else if r is a product gate **then**
  for $i = 1$ **to** $|\mathsf{in}(r)|$ **do**
    $p \leftarrow \text{PRODUCT}(\{\text{RGCTOCIRCUIT}(r_i, \mathsf{cache}_r)\} \cup \{\text{SUPPORT}(r_j, \mathsf{cache}_s)\}_{j \neq i})$
$\mathsf{cache}_r(r) \leftarrow p$
return p

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

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

can be exactly computed in time $\mathcal{O}(|p| |q| + |p|^2 + |q|^2)$. This 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. 3.1); 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. 3.2).

Squared Loss Divergence Suppose p and q are structured-decomposable and compatible. Then their squared loss, defined as $\int_{\mathsf{val}(\mathbf{X})} (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 \, d\mathbf{X}$, can be computed exactly in time $\mathcal{O}(|p| |q| + |p|^2 + |q|^2)$. This follows by noting that the integral decomposes over the expanded square as $\int_{\mathsf{val}(\mathbf{X})} p^2(\boldsymbol{x}) \, d\mathbf{X} + \int_{\mathsf{val}(\mathbf{X})} q^2(\boldsymbol{x}) \, d\mathbf{X} - 2 \int_{\mathsf{val}(\mathbf{X})} p(\boldsymbol{x}) q(\boldsymbol{x}) \, d\mathbf{X}$. Each integral can be computed by leveraging the tractable natural power of structured-decomposable circuits (Thm. 3.2) and the tractable product of compatible circuits (Thm. 3.1).

Expected predictions Let p be a structured-decomposable PC over variables $\mathbf{X}$ and f be a regression circuit [20] 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. 5. Proof follows from noting that Alg. 5 outputs a polysize circuit representation h in polytime. Then, computing $\mathbb{E}_{\boldsymbol{x} \sim p(\mathbf{X})}\left[h(\boldsymbol{x})\right]$ can be done in $\mathcal{O}(|p|\,|h|)$ time by Thm. 3.1.