Polynomial Semantics of Tractable Probabilistic Circuits

Oliver Broadrick[1] **Honghua Zhang**[1] **Guy Van den Broeck**[1]

[1]Computer Science Dept., University of California, Los Angeles, California, USA

Abstract

Probabilistic circuits compute multilinear polynomials that represent probability distributions. They are tractable models that support efficient marginal inference. However, various polynomial semantics have been considered in the literature (e.g., network polynomials, likelihood polynomials, generating functions, Fourier transforms, and characteristic polynomials). The relationships between these polynomial encodings of distributions is largely unknown. In this paper, we prove that for binary distributions, each of these probabilistic circuit models is equivalent in the sense that any circuit for one of them can be transformed into a circuit for any of the others with only a polynomial increase in size. They are therefore all tractable for marginal inference on the same class of distributions. Finally, we explore the natural extension of one such polynomial semantics, called probabilistic generating circuits, to categorical random variables, and establish that marginal inference becomes #P-hard.

1 INTRODUCTION

Modeling probability distributions in a way that allows efficient probabilistic inference (e.g. computing marginal probabilities) is a key challenge in machine learning. Decades of research towards meeting this challenge have led to the development of families of tractable models including bounded-treewidth graphical models such as hidden Markov models [Rabiner and Juang, 1986] and (mixtures of) Chow-Liu Trees [Chow and Liu, 1968, Meila and Jordan, 2000], determinantal point processes [Kulesza and Taskar, 2012], and various families of probabilistic circuits (PCs) such as sum-product networks [Poon and Domingos, 2011, Peharz et al., 2018] and probabilistic sentential decision diagrams [Kisa et al., 2014].

As a unifying representation for all aforementioned models, probabilistic circuits (PCs) compactly represent polynomials encoding probability distributions (Fig. 2). The most commonly studied classes of PCs, for example, are compact representations of network polynomials [Darwiche, 2003], which are probability mass functions. A majority of prior works on PCs representing network polynomials, as well as the more recent PCs representing characteristic functions [Yu et al., 2023], assume that PCs need to satisfy a property called decomposability[1] [Darwiche and Marquis, 2002] for marginals to be tractable. However, for one class of PCs called probabilistic generating circuits (PGCs) [Zhang et al., 2021, Harviainen et al., 2023], there are no such structural assumptions, making them strictly more expressively efficient than decomposable PCs [Martens and Medabalimi, 2015]. PGCs are compact representations of probability generating functions (generating polynomials for short), and the only requirement for tractable marginals is that the generating polynomials being represented are multilinear.

From this perspective, we study the circuit representations for multilinear polynomials of different semantics, and find that they all are tractable for marginal probabilities regardless of the circuit structure. Moreover, we show that their circuit representations are all equally expressive-efficient, regardless of their choice of polynomial semantics.

In this work, in addition to the network polynomials $p(x, \bar{x})$ and generating polynomials $g(x)$, mentioned above, we also consider *likelihood* polynomials $p(x)$ [Roth and Samdani, 2009] and their Fourier transforms $\hat{p}(x)$, which are also known as characteristic functions [Yu et al., 2023]. We show that circuits computing these four classes of polynomials are all *equally* expressive-efficient (Sec. 3, 4, and 5). In particular, we show that given a circuit computing any of these polynomials, we can transform it to a circuit for any of the others in polynomial time with respect to the size of the original circuit. Figure 1 shows a diagram of the transforma-

[1]Also known as *syntactic multilinearity* in circuit complexity

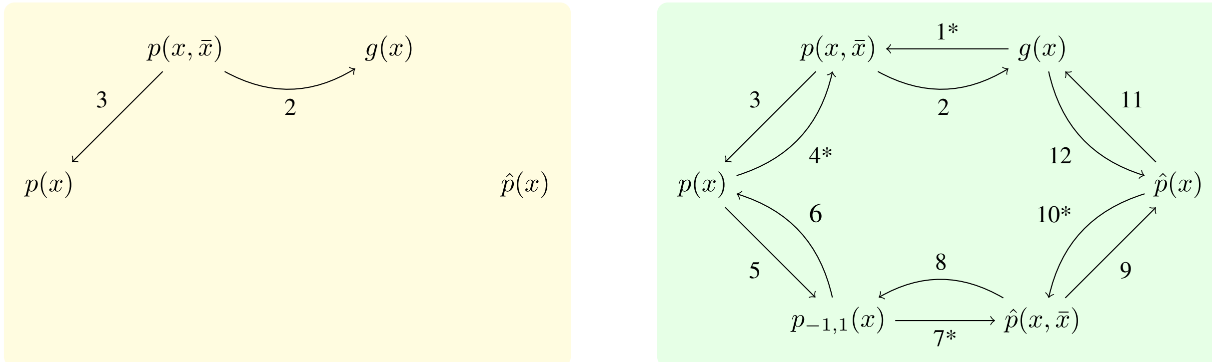


Figure 1: Polynomial time circuit transformations between polynomial semantics including: likelihood $p(x)$, network $p(x, \bar{x})$, generating $g(x)$, and Fourier $\hat{p}(x)$ polynomials. Previously known transformations are displayed on the left; (2) is given in Zhang et al. [2021], and (3) is implicit in Roth and Samdani [2009]. The results in this paper are shown on the right. Edges labeled by * correspond to transformations which map circuits of size s to circuits of size $O(n^2 s)$; other edges correspond to transformations which map circuits of size s to circuits of size $O(s)$.

tions we present. Notably, for the likelihood polynomials, we also propose the first tractable inference algorithm for their circuit representations. Our transformation assumes no structural properties of circuits and in Section 6, we show that some of the transformations can be simplified if we also assume decomposability.

In Section 7, we extend our discussion to non-multilinear polynomials. Specifically, PGCs represent multilinear generating polynomials for modeling binary random variables. We propose to generalize them to *categorical PGCs* such that the non-multilinear generating polynomials they represent have well-known categorical semantics. Unfortunately, we show that inference in a categorical PGC with n random variables and $k > 3$ categories is #P-hard, and they are therefore not tractable models.

2 BACKGROUND

We use $\Pr$ to denote a probability distribution on n binary random variables $\boldsymbol{X} = \{X_1, X_2, \ldots, X_n\}$, each taking values in $\{0, 1\}$. Let $[n] = \{1, 2, \ldots, n\}$. For any $S \subseteq [n]$, let $\boldsymbol{x}_S \in \{0,1\}^n$ denote the assignment $X_i = 1$ for $i \in S$ and $X_i = 0$ for $i \notin S$. We study polynomials in variables $x_1, \ldots, x_n$ which we often abbreviate to x. A polynomial is *multilinear* if it is linear in every variable.

In this paper we consider multilinear polynomials as representations of probability distributions. To compactly represent polynomials, we use arithmetic circuits, a fundamental object of study in computer science [Shpilka and Yehudayoff, 2010] which have proven useful for representing tractable probabilistic models.

Definition 1. *An arithmetic circuit (AC) is a directed acyclic graph consisting of three types of nodes:*

1. *Sum nodes $\oplus$ with weighted edges to children;*
2. *Product nodes $\otimes$ with unweighted edges to children;*
3. *Leaf nodes, which are variables in $\{x_1, \ldots, x_n\}$ or constants in $\mathbb{R}$.*

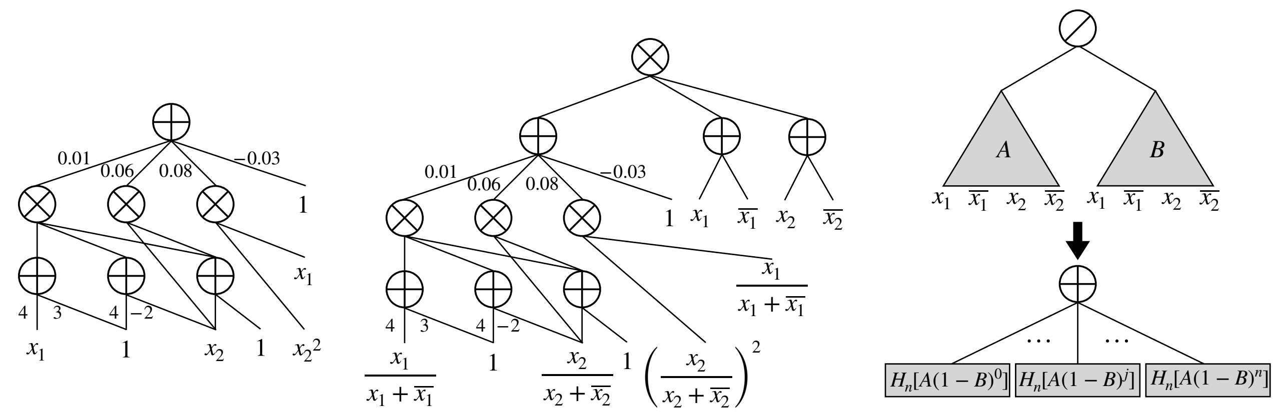
An AC has one node of in-degree 0*, and we refer to it as the* root*. The size of an AC is the number of edges in it.*

Each node in an AC represents a polynomial: (i) each leaf represents the polynomial x_i or a constant, (ii) each sum node represents the weighted sum of the polynomials represented by its children, and (iii) each product node represents the product of the polynomials represented by its children. The polynomial represented by an AC is the polynomial represented by its root. We note that the standard definition of AC in the circuit complexity literature uses unweighted sums, but the models are equivalent up to constant factors. For the remainder of this paper we use the term circuit to mean arithmetic circuit.

Note that when we say that two polynomials/circuits are the same, we *do not* mean that they agree on all inputs in $\{0,1\}^n$ but that they agree on all real inputs in $\mathbb{R}^n$; the polynomials are equivalent elements in the ring of polynomials $\mathbb{R}[x_1, \ldots, x_n]$.

3 NETWORK AND LIKELIHOOD POLYNOMIALS

There are various polynomials containing all the information of a binary distribution $\Pr$, in the sense that any value $\Pr(\boldsymbol{x})$ can be recovered from the polynomial alone. It is known that efficient circuit representations of some such polynomials still allow tractable marginal inference, but a unified analysis of the various polynomial representations is lacking. In this section, we begin with the most studied such polynomial, the network polynomial, and establish its connections to the



(a) PC for likelihood polynomial. (b) Leaf nodes replaced with division gadgets. (c) top: A single division node. bottom: Sum of homogeneous parts.

Figure 2: An example transforming a circuit representing a likelihood polynomial $p(x) = 0.08x_1x_2 + 0.16x_1 + 0.12x_2 + 0.09$ to a circuit representing a network polynomial. First, (b) gadgets using division nodes are introduced at the leaves (as well as a multiplying factor) to obtain a rational function equivalent to the network polynomial. Then (c:top) all divisions are pushed to a single division node at the root so $p(x, \bar{x}) = A/B$, and (c:bottom) a sum over necessary homogeneous parts of A and B is formed.

more natural – yet still, as we show, tractable – likelihood polynomial.

3.1 NETWORK POLYNOMIALS

Darwiche [2003] showed that Bayesian Networks can be compiled to circuits computing a certain polynomial representation of their distribution which he called the *network polynomial* (also see [Castillo et al., 1995]). The *network polynomial* of binary probability distribution $\Pr$ is

$$p(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n) = \sum_{S \subseteq [n]} \Pr(\boldsymbol{x}_S) \prod_{i \in S} x_i \prod_{i \notin S} \bar{x}_i. \tag{1}$$

Significant work towards learning and applying circuits computing this polynomial has since been developed [Poon and Domingos, 2011, Peharz et al., 2020, Liu et al., 2021]. In particular, this is the canonical polynomial computed by circuits in the growing literature on Probabilistic Circuits (PC) [Choi et al., 2020]. The key feature of circuits computing network polynomials is that they enable linear time (and very simple!) marginal inference. We note that while algorithms for marginalization are typically given for smooth and decomposable circuits, the following Proposition holds for circuits of any structure which compute a network polynomial.

Proposition 1. *Computing marginals on a circuit of size s representing a network polynomial takes $O(s)$ time. For the random variable assignment $X_i = 1$, set $x_i = 1$ and $\bar{x}_i = 0$; for $X_i = 0$, set $x_i = 0$ and $\bar{x}_i = 1$; marginalize X_i by setting $x_i = \bar{x}_i = 1$.*

The network polynomial is a polynomial with very specific structure. First, the network polynomial is multilinear. Second, every monomial with a nonzero coefficient contains x_i or $\bar{x}_i$ for *every* $i \in \{1, 2, \ldots, n\}$. However, we wonder whether the structure of the monomials with variables x_i and $\bar{x}_i$ is necessary for marginal inference. We next consider a definition which does not use these $\bar{x}_i$ variables but, as we show, remains tractable.

3.2 LIKELIHOOD POLYNOMIALS

Roth and Samdani [2009] considered perhaps the simplest polynomial representation of $\Pr$, that which directly computes $\Pr$ using variables $x_1, \ldots, x_n$. Such a polynomial is obtained from a network polynomial by substituting $\bar{x}_i = 1 - x_i$ (transformation 2 in Figure 1). We call this the *likelihood* polynomial:

$$p(x_1, \ldots, x_n) = \sum_{S \subseteq [n]} \Pr(\boldsymbol{x}_S) \prod_{i \in S} x_i \prod_{i \notin S} (1 - x_i). \tag{2}$$

While conceptually simple, it is not clear how or whether it is possible to efficiently compute marginals given a circuit representation of the likelihood polynomial. In particular, Roth and Samdani [2009] considered only "flat" representations of the likelihood polynomial, where all monomials with nonzero coefficients and their coefficients are stored explicitly. While marginal inference is linear in the size of the flat representation, there is an exponential gap in succinctness between circuits and flat representations.

We note that both network polynomials and likelihood polynomials are multilinear. Moreover, the standard structural property decomposability in the tractable circuits literature implies that the polynomial computed by a circuit is multilinear. Indeed, inference on circuits that agree with $\Pr$ on all inputs in $\{0,1\}^n$ becomes intractable without multilinearity. For example, if we just relax the restriction of multilinearity to circuits computing polynomials that are quadratic in each variable, marginal inference already becomes #P-hard (e.g. implicit in the proof of Theorem 2 in Khosravi et al. [2019]).

We show that given a circuit computing a likelihood polynomial, there is still a linear time marginal inference algorithm.

Proposition 2. *Marginal probabilities on a circuit of size s representing a likelihood polynomial can be computed in time $O(s)$.*

By definition, a circuit representing a likelihood polynomial computes

$$p(x_1,\ldots,x_n) = \sum_{S\subseteq[n]} \Pr(\boldsymbol{x}_S)\prod_{i\in S} x_i \prod_{i\notin S}(1-x_i). \quad (3)$$

We observe that setting $x_i = \bar{x}_i = 1$ in the following expression[2] is equivalent to marginalizing in a network polynomial as in Proposition 1.

$$\begin{aligned}
&\prod_{i=1}^{n}(x_i+\bar{x}_i)\cdot p\left(\frac{x_1}{x_1+\bar{x}_1},\ldots,\frac{x_n}{x_n+\bar{x}_n}\right)\\
&=\prod_{i=1}^{n}(x_i+\bar{x}_i)\sum_{S\subseteq[n]}\Pr(\boldsymbol{x}_s)\prod_{i\in S}\frac{x_i}{x_i+\bar{x}_i}\prod_{i\notin S}\left(1-\frac{x_i}{x_i+\bar{x}_i}\right)\\
&=\sum_{S\subseteq[n]}\Pr(\boldsymbol{x}_s)\prod_{i\in S}x_i\prod_{i\notin S}\bar{x}_i\\
&=p(x,\bar{x}).
\end{aligned}$$

In fact, this expression naturally corresponds to a circuit computing the network polynomial *using division nodes*; replace inputs x_i with $x_i/(x_i+\bar{x}_i)$ and multiply the whole circuit by $\prod_{i=1}^{n}(x_i+\bar{x}_i)$. However, the probabilistic circuits literature does not typically use division nodes, and available software libraries and known algorithms would need to be reconsidered to use division nodes, not to mention possible divide-by-zero problems – which we will in fact see arise in Section 4. This leads to the question, can we find a circuit computing an equivalent polynomial without use of division nodes? Classic work in the circuit complexity theory literature by Strassen [1973] provides a positive answer.

Theorem 1 (Strassen)**.** *If C is an arithmetic circuit with division nodes of size s, computing polynomial p of degree d over an infinite field, then there exists an arithmetic circuit C' of size poly(s,d,n) that computes p using only addition and multiplication nodes.*

[2] Readers familiar with the weighted model counting task on decomposable logic circuits might recognize a neutral labeling function in this expression [Kimmig et al., 2017].

In particular, we have the following Theorem, which corresponds to transformation 4 in Figure 1.

Theorem 2. *Let $\Pr$ be a probability distribution on n binary random variables. Then a circuit of size s computing the likelihood polynomial for $\Pr$ can be transformed to a circuit of size $O(n^2 s)$ computing the network polynomial for $\Pr$.*

To illustrate the algorithm, we consider the running example in Figure 2. Figure 2a shows the initial circuit that represents the likelihood polynomial. Figure 2b shows the circuit computing the expression with division nodes. To remove division nodes, the first observation is that all division nodes can be moved 'up' to a single division at the output node using the identities $(a/b)\times(c/d)=(ac)/(bd)$ and $(a/b)+(c/d)=(ad+bc)/(bd)$, as visualized in Figure 2c. At this point we have the network polynomial written as a ratio of two polynomials, $p(x,\bar{x}) = A(x,\bar{x})/B(x,\bar{x})$. Without loss of generality we assume B has constant term one, i.e. $B(0,0,\ldots,0)=1$.[3]

One additional result from the circuit complexity theory literature is needed at this point; for any circuit f of size s and degree d, a circuit of size $O(d^2 s)$ can be constructed (with $d+1$ outputs) computing $H_0[f], H_1[f],\ldots,H_d[f]$ where $H_i[f]$ has degree i and $f=\sum_i H_i[f]$ [Shpilka and Yehudayoff, 2010]. This process is called *homogenization*, and the $H_i[f]$'s the *homogeneous parts* of f.

The final division node can now be eliminated by use of the common polynomial identity $\frac{a}{1-r}=\sum_{j=0}^{\infty} ar^j$. We have

$$p(x,\bar{x}) = \frac{A}{B} = \frac{A}{1-(1-B)} = \sum_{j=0}^{\infty} A(1-B)^j. \quad (4)$$

In particular, these equalities hold for the homogeneous parts of $p(x,\bar{x})$. And, because $B(0,\ldots,0)=1$, we know that $1-B$ has constant term zero, and so all monomials in $(1-B)^j$ have degree at least j. Since we know that the network polynomial $p(x,\bar{x})$ has all terms of degree exactly n, we only need to compute

$$H_n[p(x,\bar{x})] = \sum_{j=0}^{n} H_n[A(1-B)^j],$$

as illustrated by Figure 2c. In particular, a single circuit computing $(1-B)^j$ for $j\in\{0,1\ldots,n\}$ can be homogenized in addition to homogenizing A, to compute $p(x,\bar{x})$ with size $O(n^2 s)$.

[3] By a standard argument, if B does not already have constant term 1, then its inputs can be translated and the whole function scaled accordingly to achieve this property.

4 GENERATING POLYNOMIALS

So far we have considered circuits that directly compute a distribution. However, there are other well known polynomial representations of probability distributions which have been shown as promising representations for tractable probabilistic modeling. Zhang et al. [2021] consider circuits computing the Probability Generating Function of a distribution. Generating Functions are well studied in mathematics as theoretical objects [Wilf, 2005], but have recently been identified as useful data structures [Zhang et al., 2021, Klinkenberg et al., 2023, Zaiser et al., 2023]. The *generating polynomial* for probability distribution $\Pr$ is

$$g(x_1, \dots, x_n) = \sum_{S \subseteq [n]} \Pr(\boldsymbol{x}_s) \prod_{i \in S} x_i. \tag{5}$$

Zhang et al. [2021] call circuits computing generating polynomials Probabilistic Generating Circuits (PGCs) and show that marginal inference on PGCs is tractable. For a PGC of size s in n variables, they provide an $O(sn \log n \log \log n)$ marginal inference algorithm which has been improved by Harviainen et al. [2023] to $O(ns)$. It is also noted by Zhang et al. [2021] that circuits computing network polynomials can be transformed to PGCs simply by replacing $\bar{x}_i$'s by 1, and so any distribution with a polynomial-size circuit computing its network polynomial also has a polynomial-size PGC; this is transformation 2 in Figure 1. On the other hand, they show that there are distributions with polynomial-size PGCs but for which any decomposable circuit computing the network polynomial using only positive weights has exponential size (and additional PGC lower bounds are known Bläser [2023]). It is left as an open question whether this separation still holds for circuits with unrestricted weights. We settle this question with a negative answer. Using a method similar to that in Section 3, we show that given a PGC, one can find a circuit computing the network polynomial with a polynomial increase in size; this is transformation 1 in Figure 1.

Theorem 3. *Let* $\Pr$ *be a probability distribution on* n *binary random variables. Then a circuit of size* s *computing the probability generating function for* $\Pr$ *can be transformed to a circuit of size* $O(n^2 s)$ *computing the network polynomial for* $\Pr$.

Proof. We obtain the desired circuit by first constructing a circuit that computes $p(x, \bar{x})$ using division nodes. Observe

$$\begin{aligned}
&\left(\prod_{i=1}^{n} \bar{x}_i\right) g\left(\frac{x_1}{\bar{x}_1}, \frac{x_2}{\bar{x}_2}, \dots, \frac{x_n}{\bar{x}_n}\right) \\
&= \left(\prod_{i=1}^{n} \bar{x}_i\right) \sum_{S \subseteq [n]} \Pr(\boldsymbol{x}_S) \prod_{i \in S} \frac{x_i}{\bar{x}_i} \\
&= \sum_{S \subseteq [n]} \Pr(\boldsymbol{x}_S) \prod_{i \in S} x_i \prod_{i \notin S} \bar{x}_i \\
&= p(x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n)
\end{aligned}$$

The degree of $p(x, \bar{x})$ is n, and so using Theorem 1 as in Section 3, there is a circuit computing $p(x, \bar{x})$ without division nodes of size $O(n^2 s)$. ☐

We note the similarity of the proof of Theorem 2 to that of Theorem 3. They both involve constructing a circuit to represent $p(x, \bar{x})$ initially using division nodes and then removing the division nodes. We also note the crucial difference between the proofs; in the construction for Theorem 3, the circuit with division nodes *can not be used to evaluate* $p(x, \bar{x})$ *directly because it would require division by zero whenever* $\bar{x}_i = 0$ *for any* $i \in [n]$. Therefore the ability to remove divisions while maintaining equivalence of the polynomial computed is essential for this transformation to be meaningful. As one immediate consequence, this implies the existence of polynomial size PCs computing network polynomials for DPPs since Zhang et al. [2021] showed the existence of polynomial size PGCs for DPPs. Another practical benefit is that rather than using a bespoke polynomial-interpolation algorithm for inference in PGCs, there is a simple feedforward (and easily implemented, on a GPU for example) method of inference for PGCs after the transformation has been performed.

5 FOURIER TRANSFORMS

Fourier analysis involves representing functions in the frequency domain and is ubiquitous across math and computer science. Yu et al. [2023] show that circuits representing Fourier transforms (called Characteristic Functions in Probability Theory) can improve learning in a mixed discrete-continuous setting while still supporting marginal inference when the circuit is smooth and decomposable (see Section 6 for discussion of these properties). Xue et al. [2016] show that Fourier representations can improve approximate inference in the binary setting too. The *Fourier transform* [O'Donnell, 2014] of pseudoboolean function $p : \{0, 1\}^n \to \mathbb{R}$ is

$$\hat{p}(\boldsymbol{t}) = 2^{-n} \sum_{\boldsymbol{x} \in \{0,1\}^n} p(x)(-1)^{\langle \boldsymbol{t}, \boldsymbol{x} \rangle}. \tag{6}$$

It is convenient that in this binary case, $\hat{p}$ can also be simply written as a multilinear polynomial (note that the equality holds on its domain $\{0,1\}^n$):

$$\hat{p}(t_1, \ldots, t_n) = 2^{-n} \sum_{S \subseteq [n]} p(x_S) \prod_{i \in S} (1 - 2t_i). \quad (7)$$

For the rest of the paper we use $\hat{p}$ to refer to this multilinear polynomial. We note that Fourier analysis of binary functions is a rich subject in its own right and refer the reader to O'Donnell [2014].

While there is no obvious connection between network polynomials, generating functions, and Fourier transforms, we show that in fact they are closely related. This relation hinges on switching between the domains $\{0,1\}^n$ and $\{-1,1\}^n$. In particular, we define for any polynomial f its counterpart $f_{-1,1}$ as follows:

$$f_{-1,1}(x_1, \ldots, x_n) = f\left(\frac{1-x_1}{2}, \ldots, \frac{1-x_n}{2}\right), \quad (8)$$

also a multilinear polynomial. Similarly, observe that we can write

$$f(x_1, \ldots, x_n) = f_{-1,1}\left(1 - 2x_1, \ldots, 1 - 2x_n\right). \quad (9)$$

Note that f and $f_{-1,1}$ compute *the same function* on the respective domains $\{0,1\}^n$ and $\{-1,1\}^n$ up to the bijection $\phi : \{0,1\} \to \{-1,1\}$ given by $\phi(b) = (-1)^b$ applied bitwise. In particular, Equations 8 and 9 can be applied to circuits with modifications at only the leaves, giving the following lemma.

Lemma 1. *A circuit of size s computing polynomial f (resp. $f_{-1,1}$) can be transformed to a circuit of size $O(s)$ computing $f_{-1,1}$ (resp. f).*

We now make a simple observation that connects Fourier transforms with generating polynomials; up to a constant factor, generating polynomials *are* Fourier transforms, written on the domain $\{-1,1\}^n$.

Proposition 3. *Let $\Pr : \{0,1\}^n \to \mathbb{R}$ be a probability distribution with generating polynomial g and Fourier polynomial $\hat{p}_{-1,1}(x)$ on the domain $\{-1,1\}$. Then $g(x) = 2^n \hat{p}_{-1,1}(x)$.*

Proof.

$$\begin{aligned} g(x) &= \sum_{S \subseteq [n]} \Pr(\boldsymbol{X}_s) \prod_{i \in S} x_i \\ &= \sum_{S \subseteq [n]} \Pr(\boldsymbol{X}_S) \prod_{i \in S} \left(1 - 2\left(\frac{1-x_1}{2}\right)\right) \\ &= 2^n \hat{p}_{-1,1}(x). \end{aligned}$$

□

Using only the ability to switch between the domains $\{0,1\}^n$ and $\{-1,1\}^n$ and Proposition 3, we now have transformations 11 and 12 in Figure 1.

Theorem 4. *Let $\Pr$ be a probability distribution on n binary random variables. Then a circuit of size s computing the generating polynomial g for $\Pr$ (resp. $\hat{p}$) can be transformed to a circuit of size $O(s)$ representing the Fourier transform $\hat{p}$ for $\Pr$ (resp. g).*

Proof. Proposition 3 and Lemma 1. □

Having observed this connection between generating polynomials and Fourier polynomials, we have completed a path between p and $\hat{p}$ in Figure 1, i.e. a polynomial time transformation between circuits computing them. However, we point out that this path more naturally corresponds to computing the inverse Fourier transform, and there is a symmetric set of transformations that compute $\hat{p}$ from p in a more natural way. In particular, it is more common to define the binary Fourier transform $\hat{p}$ of p in terms of its Fourier expansion:

$$\begin{aligned} p(x) &= \sum_{S \subseteq [n]} \hat{p}(x_S)(-1)^{\sum_{i \in S} x_i} \\ &= \sum_{S \subseteq [n]} \hat{p}(x_S) \prod_{i \in S} (1 - 2x_i) \end{aligned}$$

where the last equality holds for inputs in $\{0,1\}^n$. When written in this form, it becomes clear that $\hat{p}$ computes the *coefficients* of p when written as a linear combination of parity functions (specifically, $\hat{p}(x_S)$ computes the coefficient of the parity function $\prod_{i \in S}(1 - 2x_i)$). Note the equivalence of the functions $\prod_{i \in S}(1 - 2x_i)$ to the monomials $\prod_{i \in S} x_i$ on the respective domains $\{0,1\}^n$ and $\{-1,1\}^n$, and then we have that $\hat{p}(x_S)$ simply computes the coefficient of the monomial $\prod_{i \in S} x_i$ in $p_{-1,1}$. Thus, we can find $\hat{p}$ from p by first transforming p to $p_{-1,1}$ using Lemma 1 (transformations 5 and 7 in Figure 1). Then to obtain a polynomial which computes the coefficients of $p_{-1,1}$ we use the equivalent transformation from generating polynomials to network polynomials to obtain polynomial $\hat{p}_{-1,1}(x, \bar{x})$, and finally substituting $\bar{x} = 1 - x$ we obtain $\hat{p}(x)$; transformations 7 and 9 in Figure 1. Moreover, the reverse transforms can be obtained by the same methods in the 'upper half' of Figure 1 as well.

Having now completed the transformations presented in Figure 1, we ask how they simplify in the presence of structural constraints common in the tractable circuits literature.

6 DECOMPOSABILITY

So far we make no assumptions on the structural properties of PCs; in this section, we consider the special case where the PC is *decomposable* [Darwiche and Marquis, 2002], which is a common assumption that guarantees tractable marginals, and we show that in this case some of the transformations described before can be simplified. We use the *scope* of a node to refer to the set of all i such that variables x_i or $\bar{x}_i$ appear as inputs among its descendants and itself.

Definition 2 (Decomposability)**.** *A product node is decomposable if its children have disjoint scopes. A circuit is decomposable if all its product nodes are decomposable.*

Definition 3 (Smoothness)**.** *A sum node in indeterminates x and $\bar{x}$ is smooth if its children have equal scope. A circuit is smooth if all of its sum nodes are smooth.*

Decomposability is a very common property because it guarantees multilinearity and, when paired with smoothness, guarantees tractable marginal inference by computing a network polynomial. In particular, it is well known that if a circuit is smooth and decomposable, it computes a network polynomial [Poon and Domingos, 2011, Choi et al., 2020]. We note that if a circuit is decomposable, then it can be made smooth efficiently (increasing the size at most by a linear factor Choi et al. [2020], and less for certain decomposable structures Shih et al. [2019]).

We now show how the transformations used for Theorems 2,3,4 can be simplified and improved for decomposable circuits. First, we show that in decomposable circuits Fourier transforms correspond to trivial modifications *at only the leaves*.

Theorem 5. *A decomposable circuit of size s representing a likelihood polynomial p can be transformed to a decomposable circuit of size $O(s)$ representing a Fourier transform $\hat{p}$ by only modifications to the leaves.*

Sketch. A circuit representing $\hat{p}$ can be constructed with modifications pushed entirely to the leaves inductively. Essentially, decomposability allows Fourier transforms to be pushed to the children of each product node; transforms are also straightforwardly pushed to children of sum nodes. Finally, leaf nodes are univariate and so can be transformed trivially. □

Transformations 1 and 4 in Figure 1 can be simplified when the initial circuits are decomposable; the decomposability is preserved during the transformation, and the worst-case increase in size is lowered to $O(ns)$. First, a decomposable circuit of size s computing a likelihood polynomial p can be transformed to decomposable circuit of size $O(ns)$ computing $p(x, \bar{x})$. We note that this problem is exactly that of smoothing [Shih et al., 2019, Choi et al., 2020] and so the following lemma is included for completeness but is already known. In particular, this shows how Theorem 2 can be viewed as a generalization of smoothing to circuits without decomposability.

Lemma 2. *A decomposable circuit of size s computing likelihood polynomial $p(x)$ can be transformed to a decomposable circuit of size $O(ns)$ computing network polynomial $p(x, \bar{x})$.*

Next, we also have that a decomposable circuit of size s computing a generating polynomial g can be transformed to decomposable circuit of size $O(ns)$ computing $p(x, \bar{x})$. This problem, while not smoothing, can be solved by a similar approach; rather than smoothing with gadgets computing $x_i + \bar{x}_i$, simply use $\bar{x}_i$.

Lemma 3. *A decomposable circuit of size s computing generating polynomial $g(x)$ can be transformed to a decomposable circuit of size $O(ns)$ computing network polynomial $p(x, \bar{x})$.*

We note that Lemmas 2 and 3 hold for the symmetric transformations as described in Section 5; in particular, for decomposable versions of transformations 7 and 10 in Figure 1.

7 CATEGORICAL DISTRIBUTIONS

So far we have considered binary probability distributions, functions of the form $\Pr : \{0,1\}^n \to \mathbb{R}$. Of course, categorical distributions of the form $\Pr : S^n \to \mathbb{R}$ for arbitrary finite set S are also of interest. In the PC literature, categorical distributions are typically encoded as binary distributions using binary indicator variables [Darwiche, 2003, Poon and Domingos, 2011, Choi et al., 2020]. Indeed, the polynomials in this paper have no other straightforward and potentially tractable extension to the categorical setting, with one exception: generating polynomials. In fact, the generating polynomials considered in Zhang et al. [2021] are a restriction to the binary case of the following more general and standard definition. Let $\Pr : K^n \to \mathbb{R}$ be a probability distribution for $K = \{0, 1, 2, \dots, k-1\}$ where we call the elements of $\{0, 1, 2, \dots, k-1\}$ categories. Then the probability generating polynomial of Pr is

$$g(x) = \sum_{(d_1, d_2, \dots, d_n) \in K^n} \Pr(d_1, \dots, d_n) x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n}. \tag{10}$$

It is then natural to consider a categorical PGC as a circuit computing the generating function of a categorical distribution with more than two categories. This begs the question, are categorical PGCs a tractable model? To this, we give a negative answer. In fact, not only are marginals hard, but even likelihoods.

Theorem 6. *Computing likelihoods on a categorical PGC is #P-hard for $k > 3$ categories.*

We prove Theorem 6 by a reduction from the $\{0,1\}$-Permanent to categorical PGC inference. The classic work of Valiant [1979b] shows that computing the permanent of matrices with entries in $\{0,1\}$ is #P-hard. The permanent of a matrix M is

$$\text{per}\, M = \sum_{\sigma \in S_n} \prod_{i=1}^{n} M_{i,\sigma(i)}$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \underline{1} & \underline{1} & 0 & 0 \\ 0 & 0 & \underline{\underline{1}} & 0 & 0 \\ 0 & 0 & \underline{\underline{1}} & 0 & 0 \end{pmatrix}$$

Figure 3: An example of the permanent-preserving operation used to make M sparse. The new row and column are shaded in blue. The newly-added nonzero entries are singly-underlined. The nonzero entries that moved from their original column to the new one are doubly-underlined.

where S_n is the symmetric group of order n.

Our reduction proceeds in two steps. Let $M \in \{0,1\}^{n \times n}$. We first find a slightly larger but sparse matrix M' such that $\text{per}\, M = \text{per}\, M'$. Then, we use a polynomial construction from Valiant [1979a], Koiran and Perifel [2007] to obtain a categorical PGC for which computing a single likelihood would equivallently compute $\text{per}\, M$.

1. Let $M \in \{0,1\}^{n \times n}$. Suppose the ith column of M contains more than three nonzero entries. Insert a new row and column between the original i and $(i+1)$th rows and columns respectively, setting their $(i+1)$th entries (i.e. their shared value on the main diagonal) to 1. Also set the ith entry of the new row to 1. Now select any two of the original nonzero entries of the ith column and move them to the new $(i+1)$th column (i.e. if they have index j and j' in column i, set $M_{j,i} = M_{j',i} = 0$ and $M_{j,i+1} = M_{j',i+1} = 1$). Call the resulting matrix M' and observe that $\text{per}\, M = \text{per}\, M'$. Figure 3 gives an example of this transformation. Repeat this permanent-preserving operation until all columns contain at most three nonzero entries, which requires at most n^2 repetitions.

2. Let n' be the new size of M' (i.e. $M' \in \{0,1\}^{n' \times n'}$). We now simply construct a circuit $g(x)$ computing

$$\prod_{i=1}^{n'} \sum_{j=1}^{n'} M'_{i,j} x_j. \tag{11}$$

   Observe that the coefficient of the the monomial $\prod_{i=1}^{n'} x_i$ in $g(x)$ is exactly $\text{per}\, M' = \text{per}\, M$. Thus with $g(x)$ interpreted as a categorical PGC, the likelihood query with $X_1 = X_2 = \ldots = X_{n'} = 1$ computes $\text{per}\, M$.

This motivates the need to research tractable categorical distributions, e.g. possibly in the direction suggested by Cao et al. [2023]. In particular, this calls for careful consideration of the use of generating functions over categorical variables, which are *not* tractable models in general.

8 CONCLUSION

We studied tractable probabilistic circuits computing various polynomial representations of probability distributions. For binary probability distributions we show that a number of previously studied polynomials have equivalently expressive-efficient circuit representations. Among circuits computing network, likelihood, generating, and Fourier polynomials, all support tractable marginal inference, and, given a circuit computing any one polynomial, a circuit computing any other can be obtained with at most a polynomial increase in size. This establishes a relationship between several previously-independent marginal inference algorithms, and establishes one novel marginal inference algorithm, namely for circuits computing likelihood polynomials. These results connect well-studied mathematical objects like generating functions and Fourier transforms in their forms as tractable probabilistic circuits, opening up potential future research, for example leveraging theory developed in one semantics and translating it to another, or learning in one representation space and transforming to another.

Acknowledgements

We thank Benjie Wang for his insightful comments. This work was funded in part by the DARPA PTG Program under award HR00112220005, the DARPA ANSR program under award FA8750-23-2-0004, and NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129.

References

Markus Bläser. Not all strongly rayleigh distributions have small probabilistic generating circuits. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 2592–2602. PMLR, 2023. URL `https://proceedings.mlr.press/v202/blaser23a.html`.

William X Cao, Poorva Garg, Ryan Tjoa, Steven Holtzen, Todd Millstein, and Guy Van den Broeck. Scaling integer arithmetic in probabilistic programs. In *Uncertainty in Artificial Intelligence*, pages 260–270. PMLR, 2023.

Enrique Castillo, José Manuel Gutiérrez, and Ali S Hadi. Parametric structure of probabilities in bayesian networks. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 89–98. Springer, 1995.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. oct 2020. URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

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

Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, may 2003. ISSN 0004-5411. doi: 10.1145/765568.765570. URL https://doi.org/10.1145/765568.765570.

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

Juha Harviainen, Vaidyanathan Peruvemba Ramaswamy, and Mikko Koivisto. On inference and learning with probabilistic generating circuits. In *The 39th Conference on Uncertainty in Artificial Intelligence*, 2023.

Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. *On tractable computation of expected predictions*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22: 46–62, 2017. ISSN 1570-8683. SI:Uncertain Reasoning.

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)*, July 2014. URL http://starai.cs.ucla.edu/papers/KisaKR14.pdf.

Lutz Klinkenberg, Tobias Winkler, Mingshuai Chen, and Joost-Pieter Katoen. Exact probabilistic inference using generating functions, 2023.

Pascal Koiran and Sylvain Perifel. The complexity of two problems on arithmetic circuits. *Theoretical Computer Science*, 389(1):172–181, 2007. ISSN 0304-3975.

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012. ISSN 1935-8237. doi: 10.1561/2200000044. URL http://dx.doi.org/10.1561/2200000044.

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. *arXiv preprint arXiv:2111.11632*, 2021.

James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks, 2015.

Marina Meila and Michael I Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1(Oct): 1–48, 2000.

Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Probabilistic deep learning using random sum-product networks. *arXiv preprint arXiv:1806.01910*, 2018.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020.

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, 2011. doi: 10.1109/ICCVW.2011.6130310.

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

Dan Roth and Rajhans Samdani. Learning multi-linear representations of distributions for efficient inference. *Machine Learning*, 76(2):195–209, 2009. doi: 10.1007/s10994-009-5130-x. URL https://doi.org/10.1007/s10994-009-5130-x.

Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. *Smoothing structured decomposable circuits*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, 01 2010. doi: 10.1561/0400000039.

Volker Strassen. Vermeidung von divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973. URL http://eudml.org/doc/151394.

L. G. Valiant. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, page 249–261, New York, NY, USA, 1979a. Association for Computing Machinery. ISBN 9781450374385. doi: 10.1145/800135.804419. URL https://doi.org/10.1145/800135.804419.

L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979b. ISSN 0304-3975.

Herbert S Wilf. *generatingfunctionology*. CRC press, 2005.

Yexiang Xue, Stefano Ermon, Ronan Le Bras, Carla P. Gomes, and Bart Selman. Variable elimination in the fourier domain. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 285–294, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/xue16.html`.

Zhongjie Yu, Martin Trapp, and Kristian Kersting. Characteristic circuit. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Fabian Zaiser, Andrzej S. Murawski, and Luke Ong. Exact bayesian inference on discrete models via probability generating functions: A probabilistic programming approach, 2023.

Honghua Zhang, Brendan Juba, and Guy Van den Broeck. Probabilistic generating circuits. In *Proceedings of the UAI Workshop on Tractable Probabilistic Modeling (TPM)*, jul 2021. URL `http://starai.cs.ucla.edu/papers/ZhangICML21.pdf`.

A PROOFS

Proof of Lemma 5.

Proof. We construct $\hat{p}$ inductively as follows. For a product node, we have $\hat{p}(t) = \hat{p}_0(t_0)\hat{p}_1(t_1)$, and so

$$\begin{aligned} p(x) &= \sum_{t \in \{0,1\}^n} \hat{p}(t)(-1)^{\langle t,x \rangle} \\ &= \sum_{t \in \{0,1\}^n} \hat{p}_0(t_0)\hat{p}_1(t_1)(-1)^{\langle t,x \rangle} \\ &= \left(\sum_{t_0 \in \{0,1\}^n} \hat{p}_0(t_0)(-1)^{\langle t_0,x_0 \rangle} \right) \cdot \left(\sum_{t_1 \in \{0,1\}^n} \hat{p}_1(t_1)(-1)^{\langle t_1,x_1 \rangle} \right) \\ &= p_0(x_0)p_1(x_1) \end{aligned}$$

where the first equality follows from definition, the second from the hypothesis, the third from algebra, and the final from definition. For a sum node, we have $\hat{p}(t) = \sum_i w_i \hat{p}_i(t)$, and so

$$\begin{aligned} p(x) &= \sum_{t \in \{0,1\}^n} \hat{p}(t)(-1)^{\langle t,x \rangle} \\ &= \sum_{t \in \{0,1\}^n} \left(\sum_i w_i \hat{p}_i(t) \right) (-1)^{\langle t,x \rangle} \\ &= \sum_i w_i \sum_{t \in \{0,1\}^n} \hat{p}_i(t)(-1)^{\langle t,x \rangle} \\ &= \sum_i w_i p_i(x) \end{aligned}$$

where the equalities follow, respectively, from definition, assumption, commutativity of addition, and definition.

For leaf nodes, it suffices to consider only univariate leaves that are children of sums; for any leaf a child of a product node, add a sum node with weight 1 between them. Then, for a univariate child of a sum node with scope the singleton $\{i\}$, we have either $p(x_i) = c$, and so

$$\begin{aligned} \hat{p}(t_i) &= \sum_{S \subseteq [n]} p(x_S) \prod_{i \in S} (1 - 2t_i) \\ &= 2^{-n}(c + c(1 - 2x_i)) = 2^{-n+1} c(1 - x_i) \end{aligned}$$

or $p(x_i) = x_i$, in which case

$$\hat{p}(t_i) = \sum_{S \subseteq [n]} p(x_S) \prod_{i \in S} (1 - 2t_i) = 2^{-n}(1 - 2x_i). \tag{12}$$

□