Electronic Colloquium on Computational Complexity, Report No. 152 (2020)

Variants of the Determinant Polynomial and VP-completeness

Prasad Chaugule*, Nutan Limaye†, and Shourya Pandey‡

Indian Institute of Technology, Bombay (IITB)

October 7, 2020

Abstract

The determinant is a canonical VBP-complete polynomial in the algebraic complexity setting. In this work, we introduce two variants of the determinant polynomial which we call $\texttt{StackDet}_n(X)$ and $\texttt{CountDet}_n(X)$ and show that they are VP and VNP complete respectively under p-projections. The definitions of the polynomials are inspired by a combinatorial characterisation of the determinant developed by Mahajan and Vinay (SODA 1997). We extend the combinatorial object in their work, namely *clow sequences*, by introducing additional edge labels on the edges of the underlying graph. The idea of using edge labels is inspired by the work of Mengel (MFCS 2013).

1 Introduction

In an influential paper of Valiant [Val79], a complexity theoretic view of algebraic computation was presented. This work led to a classification of polynomials based on the ease of computing them. Consequently, complexity classes such as VF, VBP, VP and VNP were defined and investigated in many follow-up papers. These algebraic classes were designed with the intention of mimicking Boolean complexity classes. It was believed that they would give rise to equally interesting, but potentially easier to resolve questions. For example, the question of separating the classes VP and VNP turned out to be very interesting, like its Boolean counterpart, namely the famous question of separating NP from P.

While there are many parallels between these two worlds, over the years, many crucial differences between them have also surfaced. Specifically, in the Boolean world, many naturally occurring problems have been found to be *complete* for the classes NP and P[1]. Although many naturally occurring polynomials are known to be complete[2] for VNP, until very recently no natural polynomial was known to be complete for VP.

The process of finding many complete problems for a complexity class is crucial in many ways. For one, each complete problem presents a potentially different way of understanding the class. It also makes the complexity class rich and robust. In this work, we contribute to the class of VP-complete polynomials.

*`prasad@cse.iitb.ac.in`

†`nutan@cse.iitb.ac.in` Funded by SERB project File no. MTR/2017/000909.

‡`shouryap@cse.iitb.ac.in`

[1]A problem P is said to be complete for a Boolean complexity class $\mathcal{C}$ if $P \in \mathcal{C}$ and any problem P' in $\mathcal{C}$ *reduces* to P in polynomial time.

[2]A polynomial $P_n(X)$ is said to be complete for an algebraic complexity class $\mathcal{A}$ if $P_n(X)$ can be computed in $\mathcal{A}$ and any polynomial $P'_m(Y)$ can be obtained from $P_n(X)$ by setting the variables in X to variables in Y or field constants. For formal definitions see Section 2

ISSN 1433-8092

Until as recently as 2014, hardly any natural VP-complete polynomials were known. In Durand et. al. [DMM+14] and Mahajan et al. [MS18], many interesting and fairly natural families of polynomials were shown to be VP-complete. In [CLV19], a few more polynomials complete for VP were presented. All these polynomials were based on counting graph homomorphisms[3].

In this work we define two fairly simple to state variants of the determinant polynomial and show that they are VP and VNP complete. As the determinant is known to be complete for the class VBP (a class known to be contained in VP), this gives a satisfactory way of using the same base polynomial, namely the determinant polynomial, whose generalisations capture the class VP and VNP.

The determinant polynomial is a central object of study in algebraic complexity theory. Classically, the determinant has been studied for many centuries by mathematicians, physicists, numerical analysts and computer scientists.

The determinant is known to be *easy to compute*. In this respect, it enjoys a rather rare place in computation; it is an extremely useful quantity which is also efficiently computable. The classical efficient algorithms for the determinant are typically variants of the Guassian elimination method. In last three to four decades, other approaches for computing the determinant have also been proposed. One such example is an innovative approach proposed by Mahajan and Vinay [MV97], which gave the first combinatorial characterization of the determinant that yielded an efficient algorithm.

In this work, we take our inspiration from this combinatorial characterization of the determinant polynomial and define two variants of the determinant which we call $\texttt{StackDet}_n$ and $\texttt{CountDet}_n$. We show that they are complete for the classes VP and VNP, respectively.

The main proof idea comes from a paper of Mengel [Men13], which introduces characterisations of VP and VNP using *Algebraic Branching Programs*[4] *(ABPs) with memory.* In that work, informally speaking, it is shown that when ABPs are appended with *stack-like* memory, then they capture the class VP, and when they are appended with *counter-like* memory, they characterise the class VNP. We use these ideas and combine them with the combinatorial characterisation of the determinant to define our polynomial families.

The proof that shows that the determinant polynomial is complete for VBP can be adapted in a very straightforward way along with the *ABP with memory characterisations* of VP and VNP from the work of [Men13], to obtain polynomial families that are hard for these classes. However, like many other classes of polynomials (see for instance polynomial families from [Raz08] and [Men11]), they are circuit-description dependent. From the work started by Durand et al. the quest has been to find circuit-description independent polynomial families complete for VP. We are able to achieve that here. The polynomial families we obtain here are circuit-description independent as desired and are variants of the determinant polynomial, which make them substantially different from the previous works [DMM+14], [MS18], [CLV19].

Combinatorial characterisation of the determinant. Let Y be an $m \times m$ matrix, with (i, j)th entry equal to $y_{i,j}$. It is known that the determinant of Y is sum of signed cycle covers of the directed graph represented by Y. This is one of the many combinatorial definitions of the determinant, but as is, it is not known to give rise to an efficient computational procedure. Mahajan and Vinay generalized cycle covers using a notion of *clow sequences* and proved that the sum of signed clow sequences also equals the determinant. They then proved that the signed sum of clow sequences is efficiently computable.

[3] See also [Eng16] for interesting variants of homomorphism polynomials.

[4] An algebraic branching program (ABP) is a directed layered acyclic graph with a source s and a sink t. The edges are labelled with formal variables or field constants. The weight of an s to t path π is the product of the weights on the edges of π. The polynomial computed by the ABP is the sum of weights of all the s to t paths. For more details see [SY10].

$\texttt{StackDet}_\mathbf{m}$ **and** $\texttt{CountDet}_\mathbf{m}$. We also use sum of signed clow sequences to define our polynomial. In our case, the graph has some additional edge labels. For $\texttt{StackDet}_m$ (for $\texttt{CountDet}_m$), the labels come from a *stack alphabet* (*counter alphabet*, resp.). Based on these labels, we get two types of clow sequences; those which are *stack-realizable* (*counter-realizable*) and those which are not. The polynomial sums only the prior clow sequences. We show that $\texttt{StackDet}_m$ is VP-complete and that $\texttt{CountDet}_m$ is VNP-complete. The VP upper bound (Section 3.1) comes from the observation that an ABP with stack-like memory, which was introduced by [Men13], can compute this polynomial efficiently. Similarly, VNP upper bound for $\texttt{CountDet}_m$ (Section 3.2) comes from the observation that an ABP with random-access memory, which was also introduced in [Men13], can compute this polynomial efficiently.

For the hardness proofs (Section 4, Section 6) we use ideas from [CLV19] about the block-tree structure of a universal circuit and ideas from [MV97] regarding cancellations of certain *bad clow sequences*.

2 Preliminaries

Let $G = (V, E)$ be a directed graph. A walk $(u_1, u_2, \ldots, u_{k+1})$ in G is called a closed walk, or a clow, if $u_1 = u_{k+1}$, u_1 is the least numbered vertex in the walk and for any $2 \leqslant i \leqslant k$, $u_i \neq u_1$. The vertex u_1 is called *the head of the clow*. We use $\deg(\mathcal{C})$ to denote the number of edges in $\mathcal{C}$ (counted with multiplicity), i.e. in this case k.

Definition 1 (A clow sequence [MV97]). *A clow sequence* $\widehat{\mathcal{C}} = \langle \mathcal{C}_1, \ldots \mathcal{C}_\ell \rangle$ *in a graph* $G = (V, E)$ *is an ordered tuple of clows such that* $\texttt{Head}(\mathcal{C}_1) < \texttt{Head}(\mathcal{C}_2) < \texttt{Head}(\mathcal{C}_3) < \ldots < \texttt{Head}(\mathcal{C}_\ell)$ *and* $deg(\widehat{\mathcal{C}}) = \sum_{i=1}^{\ell} deg(\mathcal{C}_i) = n$*, where* $n = |V|$*. The sign of a clow sequence,* $sign(\mathcal{C})$*, is* $(-1)^{n+\ell}$.

We define two types of directed graphs, namely Stack graphs and Counter graphs.

Definition 2 (Stack graphs and Counter graphs). *A stack graph is a directed graph* $G = (V, E, \Sigma, \phi)$*, where* V *is a set of vertices,* E *is a set of edges. The set* Σ *is a symbol set. The function* ϕ *labels every edge of the graph with either* $\texttt{Push}(s)$, $\texttt{Pop}(s)$ *for some* $s \in \Sigma$ *or with* $\texttt{No-op}$.

A counter graph is a directed graph $G = (V, E, \Sigma, \phi)$*, where* V *is a set of vertices,* E *is a set of edges. The set* Σ *is a symbol set. The function* ϕ *labels every edge of the graph with either* $\texttt{Read}(s)$, $\texttt{Write}(s)$ *for some* $s \in \Sigma$ *or with* $\texttt{No-op}$.

For any $s \in \Sigma$, $\texttt{Push}(s)$, $\texttt{Pop}(s)$ are *stack operations*. Similarly, $\texttt{Read}(s)$, $\texttt{Write}(s)$ are *counter operations*. $\texttt{No-op}$ is both a stack operation as well as a counter operation. Let $s_1 = [a_1, a_2, \ldots, a_m]$ and $s_2 = [b_1, b_2, \ldots, b_n]$ be two sequences of stack operations (or counter operations) then concatenation of s_1 followed by s_2 (denoted as $s_1 \circ s_2$) is the ordered sequence $[a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n]$. It is easy to extend this definition of concatenation of two sequences to any number of sequences.

Let $\mathcal{W} = (u_1, \ldots, u_{k+1})$ be a walk of length k in a stack graph (or counter graph) G. We define $Seq[\mathcal{W}]$ to be the sequence of stack operations (counter operations, respectively) along the edges in this walk, i.e. $[\phi(u_1, u_2), \phi(u_2, u_3), \ldots, \phi(u_k, u_{k+1})]$.

We now define stack-realizable sequences and counter-realizable sequences.

Definition 3 (Stack-realizable sequence). *A stack-realizable sequence of operations is a sequence of stack operations which can be inductively formed using the following rules :*

- *The empty sequence is a stack-realizable sequence.*
- *If* P *is a stack-realizable sequence then* $\texttt{Push}(s) \circ P \circ \texttt{Pop}(s)$ *is stack-realizable* $\forall s \in \Sigma$.

- *If P is a stack-realizable sequence then* $\texttt{No-op} \square P$ *and* $P \square \texttt{No-op}$ *are also stack-realizable.*
- *If P and Q are stack-realizable sequences then $P \square Q$ is a stack-realizable sequence.*

For example, $[\texttt{Push}(a), \texttt{Push}(b), \texttt{Pop}(b), \texttt{Push}(c), \texttt{No-op}, \texttt{Pop}(c), \texttt{Pop}(a), \texttt{No-op}]$ *is a stack-realizable sequence, whereas* $[\texttt{Push}(a), \texttt{Pop}(b)]$ *is not.*

Definition 4 (Counter-realizable sequence)**.** *A sequence of counter operations P is said to be counter-realizable if the following properties hold:*

- *For every $s \in \Sigma$,* $\texttt{Write}(s)$ *and* $\texttt{Read}(s)$ *occur equal number of times in P and*
- *for every prefix P' of P, the number of times* $\texttt{Write}(s)$ *occurs in P' is at least as much as the number of times* $\texttt{Read}(s)$ *appears in P'.*

A directed walk $\mathcal{W}$ in a stack graph (or counter graph) G is called *stack-realizable walk* (or counter-realizable walk, respectively) if and only if $Seq[\mathcal{W}]$ is stack-realizable (or counter-realizable, respectively).

Definition 5 (A realizable clow sequence)**.** *A clow sequence $\widehat{\mathcal{C}} = \langle \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell \rangle$ of a stack graph (or counter graph) G is called stack-realizable (or counter-realizable, respectively) if and only if $Seq[\mathcal{C}_1] \square Seq[\mathcal{C}_2] \square \ldots \square Seq[\mathcal{C}_\ell]$ is a stack-realizable sequence (or counter-realizable, respectively).*

Let X be a set of variables. The edges of the stack graph or counter graph may be labelled with variables from X as per a labeling function $\mathcal{L} : E \to X$. For a clow $\mathcal{C} = (u_1, u_2, \ldots, u_{k+1})$, $mon(\mathcal{C})$ denotes monomial formed by multiplying the labels of the edges in $\mathcal{C}$, i.e. $mon(\mathcal{C}) = \prod_{i=1}^{k} \mathcal{L}\left((u_i, u_{i+1})\right)$. Moreover, $mon(\widehat{\mathcal{C}}) = \prod_{i=1}^{\ell} mon(\mathcal{C}_i)$.

We are now ready to formally define the Stack Determinant polynomial.

Definition 6 (Stack Determinant polynomial)**.** *Let $G_n = (V, E, \Sigma, \Phi)$ be a stack graph such that $V = \{u_1, u_2, \ldots, u_{4n}\}$, $\Sigma = \{s_1, \ldots, s_n\}$, $E = \{(u_i, u_j) \mid 1 \leqslant i, j \leqslant 4n\}$, and $\mathcal{L}((u_i, u_j)) = x_{i,j}$. Let the function Φ be defined as follows.*

$$\Phi((u_p, u_q)) = \begin{cases} \texttt{Push}(s_i) & \text{if } q = p+1 \text{ and } p = 4 \times (i-1) + 1 \\ \texttt{Pop}(s_i) & \text{if } q = p+1 \text{ and } p = 4 \times (i-1) + 3 \\ \texttt{No-op} & \text{otherwise} \end{cases}$$

The Stack Determinant polynomial over the variable set X is defined as follows.

$$\texttt{StackDet}_{\texttt{n}}(X) = \sum_{\substack{All\ stack-realizable\ clow \\ sequences\ \widehat{\mathcal{C}}\ of\ degree\ |V|.}} sign(\widehat{\mathcal{C}}) \cdot mon(\widehat{\mathcal{C}})$$

Definition 7 (Count Determinant polynomial)**.** *Let $G_n = (V, E, \Sigma, \Phi)$ be a counter graph such that $V = \{u_1, u_2, \ldots, u_{4n}\}$, $\Sigma = \{s_1, \ldots, s_n\}$, $E = \{(u_i, u_j) \mid 1 \leqslant i, j \leqslant 4n\}$, and $\mathcal{L}((u_i, u_j)) = x_{i,j}$. Let the function Φ be defined as follows.*

$$\Phi((u_p, u_q)) = \begin{cases} \texttt{Write}(s_i) & \text{if } q = p+1 \text{ and } p = 4 \times (i-1) + 1 \\ \texttt{Read}(s_i) & \text{if } q = p+1 \text{ and } p = 4 \times (i-1) + 3 \\ \texttt{No-op} & \text{otherwise} \end{cases}$$

The Count Determinant polynomial over the variable set X is defined as follows.

$$\texttt{CountDet}_{\texttt{n}}(X) = \sum_{\substack{All\ counter-realizable\ clow \\ sequences\ \widehat{\mathcal{C}}\ of\ degree\ |V|.}} sign(\widehat{\mathcal{C}}) \cdot mon(\widehat{\mathcal{C}})$$

We now define other variants of stack determinant polynomial and the counter determinant polynomial where the stack symbol set Σ is of constant size, we fix $\Sigma = \{0, 1\}$ which is of size 2.

Definition 8 (Stack Determinant polynomial with $\Sigma = \{0,1\}$)**.** *Let $G_n = (V, E, \Sigma, \Phi)$ be a stack graph such that $V = \{u_1, u_2, \ldots, u_{8n}\}$, $\Sigma = \{0,1\}$, $E = \{(u_i, u_j) \mid 1 \leqslant i, j \leqslant 8n\}$, and $\mathcal{L}((u_i, u_j)) = x_{i,j}$. Let the function Φ be defined as follows.*

$$\Phi((u_p, u_q)) = \begin{cases} \texttt{Push}(0) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 1 \\ \texttt{Push}(1) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 3 \\ \texttt{Pop}(0) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 5 \\ \texttt{Pop}(1) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 7 \\ \texttt{No-op} & \textit{otherwise} \end{cases}$$

The Stack Determinant polynomial over the variable set X is defined as follows.

$$\texttt{StackDet}^{(2)}_{\texttt{n}}(X) = \sum_{\substack{\textit{All stack-realizable clow} \\ \textit{sequences } \widehat{\mathcal{C}} \textit{ of degree } |V|.}} sign(\widehat{\mathcal{C}}) \cdot mon(\widehat{\mathcal{C}})$$

Definition 9 (Count Determinant polynomial with $\Sigma = \{0,1\}$)**.** *Let $G_n = (V, E, \Sigma, \Phi)$ be a counter graph such that $V = \{u_1, u_2, \ldots, u_{8n}\}$, $\Sigma = \{0,1\}$, $E = \{(u_i, u_j) \mid 1 \leqslant i, j \leqslant 8n\}$, and $\mathcal{L}((u_i, u_j)) = x_{i,j}$. Let the function Φ be defined as follows.*

$$\Phi((u_p, u_q)) = \begin{cases} \texttt{Write}(0) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 1 \\ \texttt{Write}(1) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 3 \\ \texttt{Read}(0) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 5 \\ \texttt{Read}(1) & \textit{if } q = p+1 \textit{ and } p \mod 8 = 7 \\ \texttt{No-op} & \textit{otherwise} \end{cases}$$

The Count Determinant polynomial over the variable set X is defined as follows.

$$\texttt{CountDet}^{(2)}_{\texttt{n}}(X) = \sum_{\substack{\textit{All counter-realizable clow} \\ \textit{sequences } \widehat{\mathcal{C}} \textit{ of degree } |V|.}} sign(\widehat{\mathcal{C}}) \cdot mon(\widehat{\mathcal{C}})$$

We now define a notion of projections called p-projections and use them to define complete polynomials for algebraic complexity classes.

Definition 10. *A polynomial family $\{f_n\}$ is said to be a projection of a family $\{g_n\}$, denoted as $\{f_n\} \leqslant \{g_n\}$, if for every f_n (where $n \in \mathbb{N}$), there exist some $m \in \mathbb{N}$ where f_n can be computed by g_m by setting the variables of g_m to either the variables of f_n or the field constants. If m is polynomially bounded in n, it is said to be a p-projection, denoted by $\{f_n\} \leqslant_p \{g_n\}$.*

A p-bounded family $\{f_n\}$ is complete for class $\mathcal{C}$, if $f_n \in \mathcal{C}$ and for every $\{g_n\} \in \mathcal{C}$, $\{g_n\} \leqslant_p \{f_n\}$.

We now state our main theorems

Theorem 11. $\texttt{StackDet}_{\texttt{n}}(X)$ *and* $\texttt{StackDet}^{(2)}_{\texttt{n}}(X)$ *are* VP*-complete over any field under p-projections.*

Theorem 12. $\texttt{CountDet}_{\texttt{n}}(X)$ *and* $\texttt{CountDet}^{(2)}_{\texttt{n}}(X)$ *are* VNP*-complete over any field under p-projections.*

3 Upper bounds for variants of determinant family

In this section we prove that $\texttt{StackDet}_{\texttt{n}}(X)$ and $\texttt{StackDet}^{(2)}_{\texttt{n}}(X)$ are in VP while for the class VNP, we show that $\texttt{CountDet}_n$ and $\texttt{CountDet}^{(2)}_n$ are in VNP. We show this by giving a polynomial (in n) sized Stack Branching Program (SBP) for $\texttt{StackDet}_n$ and $\texttt{StackDet}^{(2)}_n$. We also give a polynomial sized Random Access Branching Program (RABP) for $\texttt{CountDet}_n$ and $\texttt{CountDet}^{(2)}_n$.

SBPs and RABPs were defined by Mengel in [Men13] to characterize the classes VP and VNP, respectively. We use this characterisation for our upper bound. We recall the definitions of SBP and RABP from [Men13].

Definition 13 (SBP [Men13])**.** *A stack branching program $G = (V, E)$ (over Σ) is an algebraic branching program with an additional function $\phi : E \longrightarrow \bigcup_{a\in\Sigma}\{\texttt{Push}(a), \texttt{Pop}(a)\} \cup \{\texttt{No-op}\}$. The polynomial computed by G is $f_G = \sum_{\mathcal{P}} mon(\mathcal{P})$, where the sum is over all the stack-realizable s-t paths in G. The size of a stack branching program G is the number of vertices in it, that is, $|V|$*

Definition 14 (RABP [Men13])**.** *A random access branching program $G = (V, E)$ (over Σ) is an algebraic branching program with an additional function $\phi : E \longrightarrow \bigcup_{a\in\Sigma}\{\texttt{Write}(a), \texttt{Read}(a)\} \cup \{\texttt{No-op}\}$. The polynomial computed by G is $f_G = \sum_{\mathcal{P}} mon(\mathcal{P})$, where the sum is over all the counter-realizable s-t paths in G. The size of a random access branching program G is the number of vertices in it.*

Lemma 15 ([Men13])**.** *A family $\{f_n\}$ is in VP if and only if there exist a stack branching program family $\mathcal{S}_n$ of size $\mathrm{poly}(n)$ to compute $\{f_n\}$. A family $\{f_n\}$ is in VNP if and only if there exist a random access branching program family $\mathcal{R}_n$ of size $\mathrm{poly}(n)$ to compute $\{f_n\}$.*

The upper bound proofs are motivated by the ABP upper bound for the Determinant polynomial proved by [MV97].

The determinant is known to be equal to the sum of signed clow sequences. This combinatorial definition of the determinant was used in [MV97] to obatin an ABP upper bound. Those familiar with the proof of [MV97] may notice that the definitions of $\texttt{StackDet}_\texttt{n}(X)$, $\texttt{StackDet}^{(2)}_\texttt{n}(X)$, $\texttt{CountDet}_\texttt{n}(X)$ and $\texttt{CountDet}^{(2)}_\texttt{n}(X)$ are inspired by this definition of the determinant. We observe that, just like the combinatorial definition of the determinant is used to obtain an ABP upper bound in [MV97], our definitions of $\texttt{StackDet}_\texttt{n}(X)/\texttt{StackDet}^{(2)}_\texttt{n}(X)$ and $\texttt{CountDet}_\texttt{n}(X)/\texttt{CountDet}^{(2)}_\texttt{n}(X)$ allow us to compute them using an SBP and RABP, respectively.

3.1 $\texttt{StackDet}_\texttt{n}(X)$ and $\texttt{StackDet}^{(2)}_\texttt{n}(X)$ are in VP

We show that $\texttt{StackDet}_\texttt{n}(X)$ is in VP. Let $G_n = (V, E, \Sigma, \Phi)$ and $\mathcal{L}$ be as in the definition of $\texttt{StackDet}_\texttt{n}(X)$. Consider the complete directed graph $G'_n = (V, E)$, i.e G_n without the stack symbols and labels. Let A_n denote the adjacency matrix of this graph under the labelling $\mathcal{L}$, i.e. $A_n[i, j] = x_{i,j}$. From the result of [MV97], we get an ABP, say $\mathcal{B}_n$, that computes the determinant of A_n.

From $\mathcal{B}_n$ we obtain an SBP $\mathcal{S}_n$, by simply defining the function ϕ. We inherit ϕ from the Φ defined in the stack graph G_n as follows. Let B_n be the graph underlying the ABP $\mathcal{B}_n$. In B_n some edges are labelled with X variables, while some other edges are labelled with field constants. The function ϕ for all edges which are labelled with field constants is set to No-op. Consider any edge (p, q) in B_n that is labelled with an X variable. Suppose the edge is labelled $x_{i,j}$, then we let $\phi((p, q)) = \Phi((u_i, u_j))$.

The following statement can now be proved in a straightforward way, which finishes the proof of the upper bound.

Claim 16. *Let $\widehat{\mathcal{C}}$ be any clow sequence in G_n. The SBP $\mathcal{S}_n$ has a stack-realizable path from s to t with weight $sign(\widehat{\mathcal{C}}) \cdot mon(\widehat{\mathcal{C}})$ if and only if $\widehat{\mathcal{C}}$ is a stack-realizable clow sequence of degree $|V|$.*

Proof. Let us start by recalling the construction of an ABP for the determinant polynomial from [MV97]. First recall that the determinant polynomial $\mathtt{Det_n}$ is defined as follows in [MV97].

$$\mathtt{Det_n}(G'_n) = \sum_{\mathcal{C} \text{ a clow sequence of degree } |V|} sign(\mathcal{C}) mon(\mathcal{C})$$

It was shown that there exist an algebraic branching program $\mathcal{B}_n$ (with s as the source vertex and t as the sink vertex and two special nodes t^+ and t^-) of size $\mathcal{O}(n^3)$ which computes $\mathtt{Det_n(G)}$. The ABP $\mathcal{B}_n$ has the following properties.

- For every clow sequence $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k \rangle$ of degree $|V|$ and positive signature, there exists a unique $s - t$ path $\mathcal{P}$ in $\mathcal{B}_n$ such that path $\mathcal{P}$ is obtained by unwinding the clows in the clow sequence $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k \rangle$ into paths, $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k$, respectively and then stitching these paths together in order $\mathcal{P}_1$ followed by $\mathcal{P}_2$ and so on upto $\mathcal{P}_k$ and then followed by a single edge $\hat{e}$ labelled by $+1$ from t^+ to t. For negative signature, it is similar; except the last edge is labelled -1 and is from t^- to t.
- The variable labels on the edges (except the last edge) in $s-t$ path $\mathcal{P}$ in $\mathcal{B}_n$ are consistent with the variable labels on the edges in the closed walks in the clow sequence $\mathcal{C}$ of G_n.
- There are no $s - t$ paths in $\mathcal{B}_n$ other than the kind of paths stated above.

As the SBP $\mathcal{S}_n$ has the same underlying graph as $\mathcal{B}_n$, i.e. B_n, ignoring Φ, we get a bijection between clow sequences of the stack graph G_n and s to t paths in B_n.

Stack graph S_n is obtained by specifying ϕ along with B_n. Note that the set of s to t paths in S_n and B_n continue to be the same. In S_n some paths become stack-realizable under the function ϕ. Consider a stack-realizable path $\mathcal{P}$ in S_n. It has a corresponding clow sequence $\widehat{\mathcal{C}}$ associated with it in G_n. As the labels of $\mathcal{P}$ are consistent with those on $\widehat{\mathcal{C}}$, we get that $\widehat{\mathcal{C}}$ is a stack-realizable clow sequence.

Conversely, if we start with a stack-realizable clow sequence of G_n, we will find an s to t stack-realizable path in S_n. This finishes the proof. $\square$

Remark 17. $\mathtt{StackDet_n^{(2)}}(X)$ *can be proved to be in VP using ideas similar to the ideas used to prove* $\mathtt{StackDet_n}(X)$ *in VP.*

3.2 $\mathtt{CountDet}_n$ and $\mathtt{CountDet^{(2)}}_n$ is in VNP

We first show that $\mathtt{CountDet_n}(X)$ is in VNP. Let $G_n = (V, E, \Sigma, \Phi)$ and $\mathcal{L}$ be as in the definition of $\mathtt{CountDet_n}(X)$. Consider the complete directed graph $G'_n = (V, E)$, i.e G_n without the counter symbols and labels. Let A_n denote the adjacency matrix of this graph under the labelling $\mathcal{L}$, i.e. $A_n[i, j] = x_{i,j}$. From the result of [MV97], we get an ABP, say $\mathcal{B}_n$, that computes the determinant of A_n.

From $\mathcal{B}_n$ we obtain an RABP $\mathcal{R}_n$, by simply defining the function ϕ. We inherit ϕ from the Φ defined in the counter graph G_n as follows. Let B_n be the graph underlying the ABP $\mathcal{B}_n$. In B_n some edges are labelled with X variables, while some other edges are labelled with field constants. The function ϕ for all edges which are labelled with field constants is set to `No-op`. Consider any edge (p, q) in B_n that is labelled with an X variable. Suppose the edge is labelled $x_{i,j}$, then we let $\phi((p, q)) = \Phi((u_i, u_j))$.

The following statement can now be proved in a straightforward way, which finishes the upper bound proof.

Claim 18. *Let $\widehat{\mathcal{C}}$ be any clow sequence in G_n. The* RABP *$\mathcal{R}_n$ has a counter-realizable path from s to t with weight $sign(\widehat{\mathcal{C}}) \cdot mon(\widehat{\mathcal{C}})$ if and only if $\widehat{\mathcal{C}}$ is a counter-realizable clow sequence of degree $|V|$.*

Proof. The proof of this claim is very similar to the proof of Claim 16. As in the proof of Claim 16, we consider the branching program $\mathcal{B}_n$ from [MV97] computing the determinant polynomial $\mathtt{Det_n}$.

The RABP $\mathcal{R}_n$ has the same underlying graph as $\mathcal{B}_n$, i.e. B_n, ignoring Φ, we get exactly the same correspondence between clow sequences of the stack graph G_n and s to t paths in B_n.

The counter graph $\mathcal{R}_n$ is obtained by specifying ϕ along with B_n. Note that the set of s to t paths in $\mathcal{R}_n$ and B_n continue to be the same. In $\mathcal{R}_n$ some paths become counter-realizable under the function ϕ. Consider a counter-realizable path $\mathcal{P}$ in $\mathcal{R}_n$. It has a corresponding clow sequence $\widehat{\mathcal{C}}$ associated with it in G_n. As the labels of $\mathcal{P}$ are consistent with those on $\widehat{\mathcal{C}}$, we get that $\widehat{\mathcal{C}}$ is a counter-realizable clow sequence.

Conversely, if we start with a counter-realizable clow sequence of G_n, we will find an s to t counter-realizable path in $\mathcal{R}_n$. This finishes the proof. □

Remark 19. $\mathtt{CountDet_n^{(2)}}(X)$ *can be proved to be in* VNP *using ideas similar to the ideas used to prove* $\mathtt{CountDet_n}(X)$ *in* VNP.

4 $\mathtt{StackDet_n}(X)$ is hard for VP

In this section we prove that $\mathtt{StackDet_n}(X)$ is VP-hard. We start by proposing two simple approaches for proving the hardness and discuss why they do not seem to work directly.

- The first way is to mimic the construction used to show that the determinant polynomial is VBP hard. Start with a stack branching program P computing f. P has designated nodes s and t. Add an extra vertex, say α, and add edges from t to α and from α to s. Also add self-loops on all the vertices of P other than s and t. Then do the following.

  (a) Firstly observe that the stack-realizable clow sequences of this graph can be partitioned into two sets, say $\mathcal{G}$ and $\mathcal{B}$. Prove that the clow sequences in $\mathcal{B}$ pairwise cancel each other and their weights add up to zero.

  (b) Moreover, show that the signed clow sequences in $\mathcal{G}$ are in one-to-one correspondence with the monomials of f.

  (c) Finally prove that the sum of signed clow sequences in $\mathcal{G}$ is equal to $\mathtt{StackDet}_n$.

  While (a) and (b) above can be proved, (c) does not seem to be true. This is because we do not have any control over the map ϕ used in P. Note that in the definition of $\mathtt{StackDet}_n$ Φ is a fixed map, whereas, in P, ϕ depends on the polynomial f. For instance, it is possible that stack symbols repeat themselves several times in ϕ, while in Φ they do not as per the definition. To obtain a graph *along with the* Φ as defined in $\mathtt{StackDet}_n$ does not seem feasible in this straightforward proof idea.

- A possible fix to the above problem is to update the definition of $\mathtt{StackDet}_n$ so that it allows for a ϕ that arises from the underlying stack branching program P that computes f. Unfortunately, that leads to polynomial families that are circuit-description dependent.

It turns out that the first approach above is what we plan to use. Our proof steps consist of the additional effort required to make this approach work.

The hardness proof proceeds in three stages. We begin with the following proof outline.

Step 1 Let $\mathcal{U}_m$ be a universal circuit ([Raz08, SY10, DMM$^+$14]) of size poly(m) computing an m-variate, degree poly(m) polynomial $f_m(Y) \in \mathsf{VP}$. We obtain *a universal block circuit* $\tilde{\mathcal{U}}_m$, which has some more structure than $\mathcal{U}_m$ and computes $f_m(Y)$.

Step 2 We take the directed graph underlying the circuit $\tilde{\mathcal{U}}_m$ and transform it into another graph G_N with N vertices, where $N = \text{poly}(m)$ and $N = 4n$ for some parameter n. The graph G_N has the following properties.

- All the cycle covers of G_N have the same sign (say +ve sign w.l.o.g.).
- All the cycle covers can be classified into two categories: *good cycle covers*, say $\mathcal{G}$, and *bad cycle covers*, say $\mathcal{B}$; and the sum of weights of the good cycle covers equals $f_m(Y)$. (We will define these notions formally below.)

Step 3 From G_N we obtain a stack graph H_N with the following properties.

- The sum of weights of stack-realizable cycle covers in H_N equals the sum of weights of cycle covers in $\mathcal{G}$, i.e. equal to $f_m(Y)$.
- Moreover, the set of stack-realizable clow sequences in H_N which are cycle covers, equals $\mathcal{G}$ and the sum of signed weights of stack-realizable clow sequences that are not cycle covers equals 0.
- Overall, the sum of signed weights of stack-realizable clow sequences of H_N equals $f_m(Y)$.

We can now interpret H_N as a complete graph, where $\mathcal{L}((u_i, u_j)) = 0$ if (u_i, u_j) is not an edge in H_N. We will show that the polynomial $\texttt{StackDet}_n$ defined with respect to H_N equals $f_m(Y)$.

The Step 1 and 2 above are obtained using the ideas of Block Trees from [CLV19]. Step 3 above uses the cancellation trick from [MV97], but now in the context of stack-realizable clow sequences (instead of clow sequences) and with respect to an SBP (instead of an ABP).

4.1 VP-hardness of $\texttt{StackDet}_\texttt{n}(X)$ Step 1

Recall that from the constructions in [Raz08, SY10, DMM$^+$14], we can assume the following properties about the universal circuit. The circuit $\mathcal{U}_m$ has m variables, size $s(m)$ and each even layer is a $+$ gate, while each odd layer is a $\times$ gate. The output gate is a $\times$ gate. The $\times$ gates are multiplicatively disjoint and have fan-in bounded by 2. The input gates have fanin 0, fanout 1. The total depth[5] of the circuit is $2c\lceil \log m \rceil + 1$, where c is some fixed constant. Say it computes a polynomial $f_m(Y)$ of degree poly(m)[6].

We now create a circuit $\tilde{\mathcal{U}}_m$, which will have the same depth, each even layer will again consist of $+$ gates and each odd layer of $\times$ gates. It will continue to be multiplicatively disjoint and its size will be poly$(s(m))$. It is created as follows:

- **Block structure.** In the jth layer of $\tilde{\mathcal{U}}_m$ we create $t(j) = 2^{\lfloor \frac{j}{2} \rfloor}$ many blocks. The blocks on the j layer are denoted by $B_1^{(j)}, B_2^{(j)}, \ldots, B_{t(j)}^{(j)}$.

- **Gates. If j is odd -** Let $g_1, \ldots, g_r$ be the $\times$ gates appearing in $\mathcal{U}_m$ in layer j. In $\tilde{\mathcal{U}}_m$, each block B has one copy of $g_1, \ldots g_r$.

  If j is even - Let $g_1, \ldots, g_r$ be the $+$ gates appearing in $\mathcal{U}_m$ in layer j. Each block B in jth layer in $\tilde{\mathcal{U}}_m$ has $s(m)$ sub-blocks. Each sub-block has one copy of $g_1, \ldots, g_r$. (That is,

[5] The depth of the circuit is the length of the longest input gate to output gate path.

[6] This description is slightly different as compared to the one in [DMM$^+$14], but it is easy to see that we can get this form for a universal circuit using ideas from [Raz08].

there are $s(m)$ copies of each gate in each block and there are $t(j)$ many blocks. So each gate is copied $t(j) \cdot s(m)$ times. Note that this is polynomially bounded in $\mathrm{poly}(m)$.)

- **Wires:** Let g be a $+$ gate in layer j with children $g_1, g_2, \ldots g_r$ in $\mathcal{U}_m$. Then the copy of g in $B_i^{(j)}$ has copies of $g_1, \ldots, g_r$ from block $B_i^{(j+1)}$ as its children for each $i \in t(j)$.

  Let g be a $\times$ gate in layer j with children $g_{\text{left}}, g_{\text{right}}$ in $\mathcal{U}_m$. Also among the different gates that use g_{left}, let g be the kth such gate. Then (the unique) copy of g in $B_i^{(j)}$ has kth copy of g_{left} from block $B_{2i-1}^{(j+1)}$ as its child. Similarly, among the gates that use g_{right}, let g be the k'th such gate. Then the copy of g in $B_i^{(j)}$ has k'th copy of g_{right} from block $B_{2i}^{(j+1)}$ as its child. Finally, we only keep the minimal circuit, i.e. we remove gates that eventually do not feed into the output gate.

This completes the description of $\tilde{\mathcal{U}}_m$. The construction is exactly the same as the construction of D'_n in [CLV19]. We call this the universal block circuit.

We state and prove the following claim which finishes step 1.

Claim 20. *The polynomial computed by $\tilde{\mathcal{U}}_m$ is $f_m(Y)$ and the size of the circuit is polynomial in $s(m)$, say $p(m)$, which in turn is polynomial in m.*

Proof. We start by noting that each gate in $\tilde{\mathcal{U}}_m$ is a copy of a gate in $\mathcal{U}_m$. We show that the polynomial computed at any gate $\tilde{g}$ in $\tilde{\mathcal{U}}_m$ is equal to the polynomial computed by the gate g in $\mathcal{U}_m$ of which $\tilde{g}$ is a copy. We do this layer-by-layer, starting with the lowermost layer.

The lowermost layer has input gates, and it is clear that the claim holds for these gates. Assuming that the claim is true for all gates of layer $j+1$, where j is some positive integer, consider now a gate $\tilde{g}$ in layer j of $\tilde{\mathcal{U}}_m$ such that $\tilde{g}$ is a copy of gate g in $\mathcal{U}_m$. We have two cases:

- g is a $+$ gate. Let $g_1, g_2, \cdots, g_r$ be its children in $\mathcal{U}_m$. By construction, $\tilde{g}$ has copies $\tilde{g}_1, \tilde{g}_2, \cdots, \tilde{g}_r$ of $g_1, g_2, \cdots, g_r$, respectively, appearing in layer $j+1$ as the children of $\tilde{g}$. By our hypothesis, $\tilde{g}_i$ computes the same polynomial in $\tilde{\mathcal{U}}_m$ that g_i computes in $\mathcal{U}_m$, for all $i \in [r]$. Therefore $\tilde{g}$ also computes the polynomial in $\tilde{\mathcal{U}}_m$ that is computed by g in $\mathcal{U}_m$.

- g is a $\times$ gate. Let g_{left} and g_{right} be the two children of g in $\mathcal{U}_m$. By construction, $\tilde{g}$ has copies $\tilde{g}_{\text{left}}$ and $\tilde{g}_{\text{right}}$ of g_{left} and g_{right} respectively appearing in layer $j+1$ as the children of $\tilde{g}$. By our hypothesis, $\tilde{g}_{\text{left}}$ and $\tilde{g}_{\text{right}}$ compute the same polynomial in $\tilde{\mathcal{U}}_m$ that g_{left} and g_{right} compute in $\mathcal{U}_m$, respectively. Therefore $\tilde{g}$ also computes the polynomial in $\tilde{\mathcal{U}}_m$ that is computed by g in $\mathcal{U}_m$.

Since the output gate of $\tilde{\mathcal{U}}_m$ is a copy of the output gate of $\mathcal{U}_m$, therefore the polynomial computed by $\tilde{\mathcal{U}}_m$ is indeed $f_m(Y)$.

Moreover, note that each gate g in $\mathcal{U}_m$ is copied in $\tilde{\mathcal{U}}_m$ at most $t(j) \cdot s(m)$ times, regardless of whether g is a $\times$ gate, a $+$ gate, or an input gate. Therefore, the number of gates in $\tilde{\mathcal{U}}_m$ is no more than $\sum_{j=1}^{2c\lceil \log m \rceil + 2} t(j) \cdot s(m) = s(m) \cdot \sum_{j=1}^{2c\lceil \log m \rceil + 2} 2^{\lfloor \frac{j}{2} \rfloor} = 3s(m)(2^{c\lceil \log m \rceil + 1} - 1) < 3s(m)(2^{c \cdot \log m + c + 1}) = 3 \cdot 2^{c+1} \cdot s(m) \cdot m^c$, which is polynomially bounded in $s(m)$, as $s(m) = \Theta(\mathrm{poly}(m))$.

□

4.2 VP-hardness of $\texttt{StackDet}_\texttt{n}(X)$ Step 2

We now consider the graph underlying the universal block circuit created in Step 1. We direct all the edges in this graph from top (i.e. from the output gate) to bottom (to the input gates).

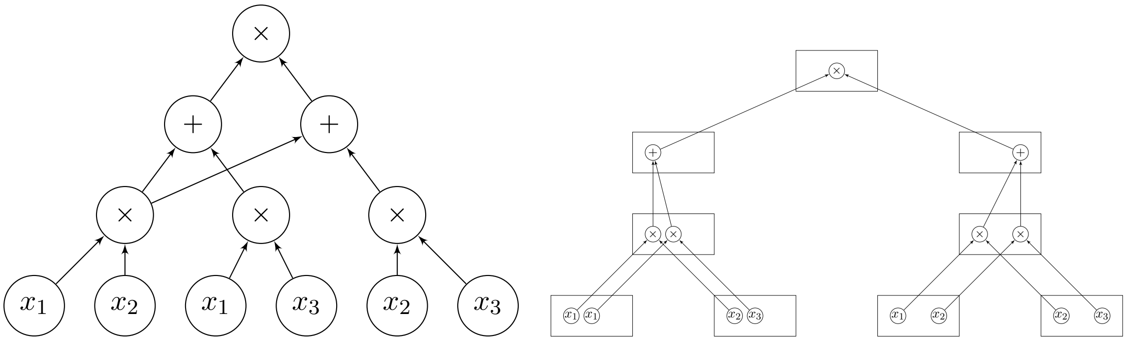

Figure 1: $\mathcal{U}_m$ computing $(x_1x_2 + x_1x_3)(x_1x_2 + x_2x_3)$ and the corresponding $\tilde{\mathcal{U}}_m$

The top-most layer has a single vertex, which is the output gate. Each layer j has $t(j)$-many blocks. We denote this directed graph by $V_{p(m)}$, where $p(m)$ is the number of vertices in this graph. We take two views of this underlying graph; *a coarse* view and *a fine* view. The fine view is simply the whole graph $V_{p(m)}$, while the coarse view is the graph formed by the block structure.

Block Tree. For the coarse view, we think of each block of $V_{p(m)}$ as a vertex. We call these *block vertices.* Two blocks vertices B, B' are said to be connected if and only if $\exists u \in B$ and $v \in B'$ such that there is an edge between u and v in $V_{p(m)}$. We refer to (B, B') as a *block edge.* By observing the connections in $V_{p(m)}$, it is easy to see that the coarse view results into a tree. We call this tree $T_{\Delta(m)}$, where $\Delta(m)$ denotes the number of leaf nodes in the tree. Let B be a vertex in T_Δ. If B is on an even layer, then it has only one child. We call these the *unary blocks.* If it is on an odd layer then it has two children. We call these blocks *binary blocks.* A path formed by block edges is called a *block path.*

When m is clear from the context, we use V_p and T_Δ to talk about these two graphs.

Construction of $\mathbf{G_N}$.

- For any binary block B and any vertex $u \in B$, we do the following. Let B_ℓ and B_r be the two children of B in T_Δ. Let $u_\ell \in B_\ell$ and $u_r \in B_r$ such that (u, u_ℓ) and (u, u_r) are edges in V_p. We sub-divide the edge (u, u_r) into (u, z_u) and (z_u, u_r). We delete the edge (u, z_u) from the graph, but retain the edge (z_u, u_r). For any node u in a binary block, we use $\texttt{Couple}(u)$ to denote the pair of edges $\{(u, u_\ell), (z_u, u_r)\}$. ($\texttt{Couple}(u)$ is not defined for a u in a unary block.)

  Note that this creates a new graph which is disconnected. If we look at the coarse view of this new graph then it is a collection of Δ block paths, let us call them $\mathcal{P}_1, \ldots, \mathcal{P}_\Delta$. Each block path contains exactly one leaf node of T_Δ. We will assume that the block paths are numbered such that the ith leaf node of T_Δ belongs to $\mathcal{P}_i$.

- We add two more vertices for each block path. We add a source vertex s_i and a sink vertex t_i for each $i \in [\Delta]$. We also add edges from s_i to all the vertices in the first block in the block path $\mathcal{P}_i$. The vertices in the last block in any block path are vertices corresponding to input gates in $\tilde{\mathcal{U}}_m$ and hence are labelled with input variables Y. Let u be a vertex in the leaf block of the path $\mathcal{P}_i$ labelled $y \in Y$. We add a directed edge (u, t_i) and label it with y. (We do this for each vertex in every leaf block of all block paths.) The graphs thus obtained are called $\mathcal{R}_1, \ldots, \mathcal{R}_\Delta$.

- We now identify t_i with s_{i+1} for $1 \leqslant i \leqslant \Delta - 1$. We use $\mathcal{R}$ to denote the graph thus formed and θ_i to denote the vertex formed by identifying t_i with s_{i+1} for $1 \leqslant i \leqslant \Delta - 1$.

Addtionally, we want to ensure that the number of vertices in the resultant graph is a multiple of 4 (This will help in defining a stack graph in the next step). To ensure this, we add three[7]additional vertices $\alpha_1, \alpha_2, \alpha_3$ and the following directed edges to obtain a graph D_N: $(t_\Delta, \alpha_3), (\alpha_3, \alpha_2), (\alpha_2, \alpha_1), (\alpha_1, s_1)$.

- We add self-loops on all the vertices except on α_1, α_2 and α_3. The edges which are not labelled with variables from Y are labelled 1.

The graph thus obtained is denoted by G_N, where N is the number of vertices in it. It is easy to note that $N = \text{poly}(p(m))$ which is $\text{poly}(m)$. We have also ensured that $N = 4n$ for some parameter n.

Definition 21. *We say that a cycle cover $C = \langle C_1, \ldots, C_k \rangle$ of G_N is a good cycle cover if for any vertex u appearing in C for which $\texttt{Couple}(u)$ is defined, either both the edges in $\texttt{Couple}(u)$ are present in C or neither is. All the other cycle covers are called bad cycle covers. Let $\mathcal{G}$ denote the set of all good cycle covers of G_N and $\mathcal{B}$ denote the set of all the bad cycle covers.*

Claim 22. *All the cycle covers of G_N have the same sign. Moreover, the sum of weights of good cycle covers equals $f_m(Y)$.*

Proof. Recall the graphs $\mathcal{R}_1, \ldots, \mathcal{R}_\Delta$ that we created from $\mathcal{P}_1, \ldots, \mathcal{P}_\Delta$. Consider any path π from s_i to t_i in $\mathcal{R}_i$. The first edge of π must be from s_i to a vertex belonging to the first block, and the last edge of π must be from a vertex belonging to the last block to the vertex t_i. All intermediate edges must connect adjacent blocks. So, the number of edges in π is one more than the number of blocks in $\mathcal{R}_i$. Therefore all paths from s_i to t_i in $\mathcal{R}_i$ have the same number of edges, say p_i.

Consider any path Π from s_1 to t_Δ. For any $2 \leqslant i \leqslant \Delta$, the vertex s_i must belong to Π (because deleting s_i disconnects the graph into two components, where s_1 and t_Δ belong to different components). This means Π can be viewed as a composition of the paths $\pi_1, \pi_2, \ldots, \pi_\Delta$, where π_i is a path from s_i to t_i for all $1 \leqslant i \leqslant \Delta$. This path π_i is also a path in $\mathcal{R}_i$, so it has length p_i. Therefore the path Π has length $p_1 + p_2 + \cdots + p_\Delta$, which we call q, say. In all, any path from s_1 to t_Δ has the same length q.

Let $C = \langle C_1, C_2, \cdots, C_k \rangle$ be a cycle cover of G_N, and consider a cycle of the cycle cover C that α_1 belongs to, say C_1. The only incoming edge to α_1 is via t_Δ, and the only edge outgoing from α_1 is to α_2. This means the edges (t_Δ, α_1) and (α_1, α_2) belong to C_1. The only outgoing edge from α_2 is to α_3, and the only outgoing edge from α_3 is to s_1. Therefore, the edges (α_2, α_3) and (α_3, s_1) also belong to C_1. So, C_1 contains a path from t_Δ to s_1 via α_1, α_2 and α_3. The remaining part of C_1 is a path from s_1 to t_Δ. This path does not use the vertices α_1, α_2, and α_3, so it is also a path in $\mathcal{R}$. As shown before, any such path from s_1 to t_Δ has length $q = p_1 + p_2 + \cdots + p_\Delta$. Therefore C_1 is a cycle of length $q + 4$.

Consider a cycle $C_j \neq C_1$ in the cycle cover C. This cycle cannot use the vertices α_1, α_2 and α_3. Furthermore, if C_j is not a loop, then it is a cycle in $\mathcal{R}$, which contradicts the fact that $\mathcal{R}$ is a DAG. Therefore C_j is a loop. In all, the cycle cover C has exactly one cycle C_1 of length $q + 4$ passing through α_1, α_2, and α_3, and $N - q - 4$ loops covering the vertices not present in the cycle C_1. Either way, the sign of any cycle cover C is fixed. It is also easy to see from the above discussion that there is a one-to-one correspondence between a path Π from s_1 to t_Δ in $\mathcal{R}$ and cycles covers of G_N.

[7]Recall that the Definition of $\texttt{StackDet}_n(X)$ requires that the total number of vertices of underlying graph is a multiple of 4. As Δ is a power of 2, it is easy to note that adding three new vertices will always make the total number of vertices of graph G_N a multiple of 4.

We will now show that the good cycle covers of G_N have a one-to-one correspondence with the proof trees of $\tilde{\mathcal{U}}_m$. Let $\mathcal{T}$ be any proof tree of $\tilde{\mathcal{U}}_m$. For any vertex u corresponding to a $\times$ gate of $\tilde{\mathcal{U}}_m$, such that u_l is the left child and u_r is the right child of u in $\mathcal{T}$, split the edge (u, u_r) into (u, z_u) and (z_u, u_r) and delete edge (u, z_u). This splits $\mathcal{T}$ into Δ paths $Q_1, Q_2, \cdots, Q_\Delta$, where Q_i belongs to $\mathcal{R}_i$ for each $i \in [\Delta]$. These Δ paths (when concatenated appropriately) trace out a path Π in D_N. This path can be completed into a cycle C_1 in G_N. This cycle C_1 along with self-loops on all the other vertices outside of C_1, forms a cycle cover C of G_N. Note that, the way this cycle cover was created, for each u in a binary block of V_p, either both edges of $\texttt{Couple}(u)$ are present in C or neither edge of $\texttt{Couple}(u)$ is present in C. Therefore C is a good cycle cover. It is easy to see that the cycle cover has weight equal to the monomial computed by $\mathcal{T}$ in $\tilde{\mathcal{U}}_m$.

For the converse, we show that a good cycle cover of G_N can be traced back to a unique parse tree of $\tilde{\mathcal{U}}_m$. Let $C = \langle C_1, C_2, \cdots, C_k \rangle$ be a good cycle cover of G_N. Let C_1 be the big cycle and the rest of the cycles in the cover be self-loops. Let E_1 denote the edges that C_1 shares with graphs $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_\Delta$. As this is a good cycle cover, for each vertex u in C_1 for which $\texttt{Couple}(u)$ is defined, edges (u, u_ℓ) and (z_u, u_r) are both present in C_1. We will identify z_u with u for all such vertices. This will give rise to a unique parse tree of $\tilde{\mathcal{U}}_m$. □

Remark 23. *To be able to sum over only the good cycle covers, we need a mechanism to ensure that the edges in* $\texttt{Couple}(u)$ *are either both activated or both deactivated. In Valiant's work [Val79] for instance, this is ensured by using an* iff graph gadget. *If we can come up with such a gadget then we will be able to show that* $\texttt{Det}_\texttt{n}$ *is VP-complete, thereby showing VP= VBP. Unfortunately, we are not able to do that. We ensure* coupling *using the stack symbols.*

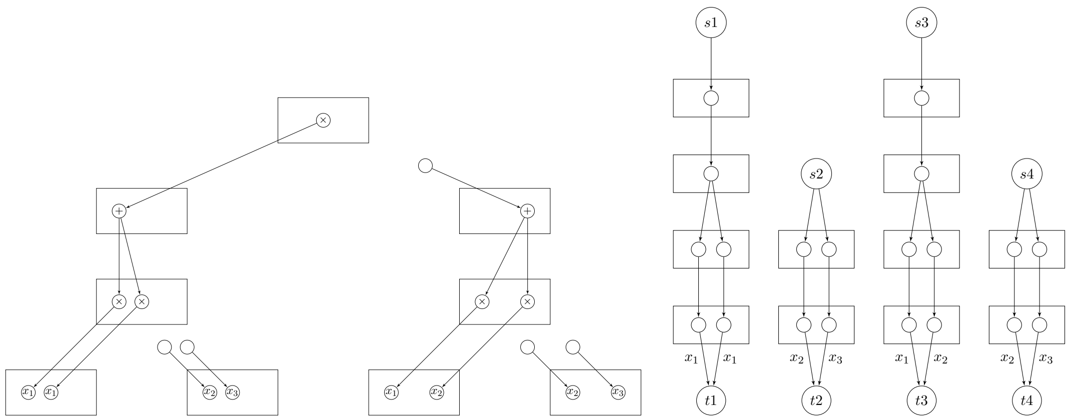

Figure 2: Graphs $\mathcal{P}_1, \ldots, \mathcal{P}_\Delta$ and $\mathcal{R}_1, \ldots, \mathcal{R}_\Delta$.

4.3 VP-hardness of $\texttt{StackDet}_\texttt{n}(X)$ Step 3

We would like to modify the graph G_N so that we filter out good cycle covers, while killing all the bad cycle covers. That is, we would like to simultaneously activate both the coupled edges of a vertex u or simultaneously de-activate both the coupled edges, in any cycle cover. We achieve this using stack symbols. Specifically, we create a stack graph H_N from G_N to achieve this.

Construction of H_N. For a vertex u for which $\texttt{Couple}(u)$ is defined, we set $\phi((u, u_\ell)) = \texttt{Push}(s_u)$ and $\phi((z_u, u_r)) = \texttt{Pop}(s_u)$. For all the other edges, ϕ is set to $\texttt{No-op}$.

Claim 24. *Consider the stack graph H_N constructed as above.*

- *The sum of weights of stack-realizable cycle covers in H_N equals the sum of weights of cycle covers in $\mathcal{G}$, i.e. equal to $f_m(Y)$.*

- *Moreover, the set of stack-realizable clow sequences in H_N which are cycle covers, equals $\mathcal{G}$ and the sum of signed weights of stack-realizable clow sequences that are not cycle covers equals 0.*

Proof. **Part 1.** From the proof of Claim 22, we have that there is a bijection between parse trees of $\tilde{\mathcal{U}}_m$ and good cycle covers of G_N. To prove the first part of the claim, we will show that there is a bijective map from a good cycle covers of G_N to stack-realizable cycle covers of H_N.

We start with some notations. Let C be a good cycle cover in G_N. Let $\mathcal{T}_C$ be the unique parse tree corresponding to C. Let $C = \langle C_1, \ldots, C_k\rangle$ and C_1 be the long cycle, while all other C_is be self-loops. (Any good cycle cover has this structure as we established in the proof of Claim 22.) Let $U_C = \{u_1, \ldots, u_\tau\}$ be the subset of vertices in C_1 for which `Couple` is defined. Note that the output gate, let us call it u^*, of $\mathcal{T}_C$ belongs to U_C.

We say that a vertex $u \in U_C$ has rank k, denoted as $\texttt{rank}(u)$, if it appears at distance $2k-1$ from the leaves in $\mathcal{T}_C$. (Note that, vertices in U_C appear at only odd distance from the leaves in $\mathcal{T}_C$.)

For $u \in U_C$ such that $\texttt{rank}(u) = 1$, u_ℓ and u_r are leaves, i.e. nodes corresponding to input gates. For a vertex u in U_C such that $\texttt{rank}(u) > 1$, let u_ℓ and u_r be its two children in $\mathcal{T}_C$. Let u' be u_ℓ's unique child in $\mathcal{T}_C$ and let u'' be the unique child of u_r in $\mathcal{T}_C$. Note that $u', u'' \in U_C$ and $\texttt{rank}(u') = \texttt{rank}(u'') = \texttt{rank}(u) - 1$.

Let Π_C be the unique path traced out by C_1 in $\mathcal{R}$. (Recall, $\mathcal{R}$ is the graph obtained by concatenating $\mathcal{R}_i$ for $i \in [\Delta]$ as described in the construction.)

For a vertex $u \in U_C$, such that $\texttt{rank}(u) = 1$, we use $\Pi_{[u]}$ to denote the subpath of Π_C from u to u_r. Given the structure of the subtree rooted at u in T_C, and assuming that u_r appears in $\mathcal{R}_{i+1}$ for some $i \in [\Delta - 1]$, we get that $\Pi_{[u]} = (u, u_\ell) \cdot (u_\ell, \theta_i) \cdot (\theta_i, z_u) \cdot (z_u, u_r)$. (Recall that θ_i was the vertex obtained by identifying t_i of $\mathcal{R}_i$ with s_{i+1} of $\mathcal{R}_{i+1}$ for $i \in [\Delta - 1]$.)

On the other hand, for $u \in U_C$ and $\texttt{rank}(u) > 1$ such that u_r appears in $\mathcal{R}_{i+1}$ for some $i \in [\Delta - 1]$, we use $\Pi_{[u]}$ to denote the subpath of Π corresponding to the entire subtree rooted at u in $\mathcal{T}_C$. Specifically, for the given the structure of the subtree rooted at u in $\mathcal{T}_C$, $\Pi_{[u]} = (u, u_\ell) \cdot (u_\ell, u') \cdot \Pi_{[u']} \cdot (\theta_i, z_u) \cdot (z_u, u_r) \cdot (u_r, u'') \cdot \Pi_{[u'']}$. We will now prove the following statement.

$$\text{For any } u \in U_C,\ Seq[\Pi_{[u]}] \text{ is stack-realizable in } H_N. \tag{1}$$

If we are able to show this, then in particular for $u^* \in U_C$ we will get that $\Pi_{[u*]}$ is stack-realizable. This will then imply that $(s_1, u^*) \cdot \Pi_{[u*]} \cdot (\theta_\Delta, t_\Delta)$ is also stack-realizable, because both (s_1, u^*) and $(\theta_\Delta, t_\Delta)$ are `No-op` edges.

We prove (1) by induction on $\texttt{rank}(u)$. Suppose $\texttt{rank}(u) = 1$ and say $u_r \in \mathcal{R}_{i+1}$, then as noted above, $\Pi_{[u]} = (u, u_\ell) \cdot (u_\ell, \theta_i) \cdot (\theta_i, z_u) \cdot (z_u, u_r)$. From our function ϕ defined for H_N, we see that $Seq[\Pi_{[u]}] = \texttt{Push}(s_u) \circ \texttt{No-op} \circ \texttt{No-op} \circ \texttt{Pop}(s_u)$. Therefore it is stack-realizable.

Suppose $\texttt{rank}(u) = k > 1$ and say that $u_r \in \mathcal{R}_{i+1}$. In this case, as noted above, we have $\Pi_{[u]} = (u, u_\ell) \cdot (u_\ell, u') \cdot \Pi_{[u']} \cdot (\theta_i, z_u) \cdot (z_u, u_r) \cdot (u_r, u'') \cdot \Pi_{[u'']}$. From this, we see that $Seq[\Pi_{[u]}] = \texttt{Push}(s_u) \circ \texttt{No-op} \circ Seq[\Pi_{[u']}] \circ \texttt{No-op} \circ \texttt{Pop}(s_u) \circ \texttt{No-op} \circ Seq[\Pi_{[u'']}]$. As $\texttt{rank}(u'), \texttt{rank}(u'') < k$, by induction hypothesis we have that $Seq[\Pi_{[u']}]$ and $Seq[\Pi_{[u'']}]$ are stack-realizable. Therefore, we get that $Seq[\Pi_{[u]}]$ is also stack-realizable.

It is not hard to argue that bad cycle covers of G_N get mapped to cycle covers of H_N, which are not stack-realizable.

Part 2. Recall that in the proof of Claim 22 we showed that any cycle cover of G_N consists of one big cycle and a collection of self-loops. Similarly, it is easy to see that in H_N any clow sequence has a certain structure: except for one clow, which will be of length $\geqslant p+4$, all other clows in the clow sequence are self-loops.

We first note that this unique long clow will contain the vertex α_1. Suppose it does not contain α_1, then no other clow in the clow sequence can cover α_1 (as all other clows are self-loops and α_1 does not have a self-loop on it). But suppose α_1 is not covered by any clow in the clow sequence, then the degree of such a clow sequence is strictly less than $|V|$.

Under the ordering in which vertex α_1 gets the lowest number, say 1, the long clow will be the first clow in the sequence, say $\mathcal{C}_1$ and α_1 will be its head.

We will adopt ideas from [MV97] in order to argue that the sum of weights of stack-realizable clow sequences which are not cycle covers is 0 in H_N. Like in [MV97], we define an involution on the signed clow sequences. (Recall that an involution is a bijective map ψ such that ψ^2 is identity.) The map ψ will have the property that any stack-realizable clow sequence $\widehat{\mathcal{C}}$ which is not a cycle cover, is paired off with another stack-realizable clow sequence $\widehat{\mathcal{C}'}$ which is again not a cycle cover and the monomials corresponding to $\widehat{\mathcal{C}}$ and $\widehat{\mathcal{C}'}$ are the same, but $sign(\widehat{\mathcal{C}'}) = -sign(\widehat{\mathcal{C}})$. For clow sequence $\widehat{\mathcal{C}}$ which is a cycle cover, the map ψ maps it to itself, i.e. it is identity for cycle covers.

Let $\widehat{\mathcal{C}}$ be a stack-realizable clow sequence, which is not a cycle cover. We start walking along the edges of C_1 starting from the head. One of the following two cases will happen first.

- **Case 1.** Either we will encounter a vertex v in C_1 such that there exists a $C_i \in \widehat{\mathcal{C}}$ for $i > 1$, such that C_i is a self-loop at vertex v.
- **Case 2.** Or we will encounter a vertex u that has $\beta \geqslant 1$ self-loops in C_1.

First note that, if $\widehat{\mathcal{C}}$ is not a cycle cover then one of the two cases must occur.

Suppose Case 1 occurs. In this case, consider $\widehat{\mathcal{C}'}$ obtained from $\widehat{\mathcal{C}}$ by merging cycle C_i with C_1, by attaching it at v in C_1. We will define ψ of $\widehat{\mathcal{C}}$ to be this $\widehat{\mathcal{C}'}$. It is easy to see that if $\widehat{\mathcal{C}}$ is a stack-realizable clow sequence, then so is $\widehat{\mathcal{C}'}$. Both have the same set of edges. And $\widehat{\mathcal{C}'}$ has one less component than $\widehat{\mathcal{C}}$, i.e. their signs are opposite.

On the other hand, suppose Case 2 occurs. In this case, consider $\widehat{\mathcal{C}'}$ obtained from $\widehat{\mathcal{C}}$ by detaching one of the β-many self-loops from u and adding that as a separate cycle in $\widehat{\mathcal{C}'}$. To observe that $\widehat{\mathcal{C}'}$ thus obtained is a stack-realizable clow sequence, we first note that there is no other clow in $\widehat{\mathcal{C}'}$ with the same head as this newly added self-loop. This is easy to see, because if say there was already a clow in $\widehat{\mathcal{C}'}$ with u as its head, then we would have been in Case 1 above. We also observe that if $\widehat{\mathcal{C}}$ is stack-realizable, then detaching a self-loop, which is a `No-op` edge, will ensure that $\widehat{\mathcal{C}'}$ is also stack-realizable. Here again, $\widehat{\mathcal{C}}$ and $\widehat{\mathcal{C}'}$ have the same set of edges and $\widehat{\mathcal{C}}$ has one less component than $\widehat{\mathcal{C}'}$, i.e. they have opposite signs.

Note that in both the cases above, if $\psi(\widehat{\mathcal{C}}) = \widehat{\mathcal{C}'}$ then $\psi(\widehat{\mathcal{C}'}) = \widehat{\mathcal{C}}$. Hence, we have the desired involution. □

With this claim we are now almost done. We will now show that there is an ordering of the vertices of H_N, which gives a graph $G_n = (V, E, \Sigma, \Phi)$ as defined in Definition 6 and a labelling function $\mathcal{L}$ as defined in Definition 6, such that $f_m(Y)$ can be obtained as a projection of $\mathtt{StackDet_n}(X)$ defined with respect to G_n, which finishes the proof.

We now come up with such an ordering. We start by ordering vertices $\theta_1, \ldots, \theta_{\Delta-1}$ and α_1, α_2 and α_3. Note that these vertices must appear in any cycle, which is not a self-loop. If we start traversing any such cycle from α_1, then we will visit these vertices in the following order $\langle \alpha_1, s_1, \theta_1, \ldots, \theta_{\Delta-1}, t_\Delta, \alpha_3, \alpha_2, \alpha_1 \rangle$. We number these vertices in the reverse order, i.e. α_1 gets numbered 1, α_2 gets 2, α_3 is numbered 3, t_Δ is numbered 4 and so on till s_1 is numbered $\Delta + 4$. This numbering ensures that all the edges that appear between these vertices get `No-op` label on them.

Now, let $u_1, u_2 \ldots, u_\tau$ be the vertices for which $\texttt{Couple}$ is defined. Let $\texttt{Couple}(u_i) = \{(u_i, u_{i\ell}), (z_{u_i}, u_{ir})\}$. For every $i \in [\tau]$, let the four vertices $u_i, u_{i\ell}, z_{u_i}, u_{ir}$ be numbered as $4(i-1)+1+(\Delta+4)$, $4(i-1)+2+(\Delta+4)$, $4(i-1)+3+(\Delta+4)$ and $4(i-1)+4+(\Delta+4)$ respectively[8]. It is easy to check that such an ordering always gives distinct numbers to all the vertices of the graph and this ordering is consistent with Φ from Definition 6.

The labelling function $\mathcal{L}$ retains the labels of all the edges of H_N as they are. For any two vertices u, v in H_N, such that there is no edge in H_N between u and v, we add such an edge in G_n, but set $\mathcal{L}((u,v)) = 0$. This labelling function now ensures that when we consider the $\texttt{StackDet}$ polynomial with respect to G_n we obtain $f_m(Y)$.

5 $\texttt{StackDet}_\texttt{n}^{(2)}(X)$ is hard for VP

In this section, we give steps for the construction of a graph H'_M where $M = \text{poly}(m)$ and $M = 8n$ (for some n) such that $\texttt{StackDet}_\texttt{n}^{(2)}(X)$ of H'_M under our projection is equal to $f_m(Y)$. We consider the stack graph H_N constructed from the universal circuit $\mathcal{U}_m$ as discussed in Section 4 and convert it into another stack graph H'_M where the stack symbol set is of a constant size, that is, 2. The idea here is to encode the symbols in the stack symbol set $\Sigma = \{s_1, s_2, \ldots, s_n\}$ using binary alphabet $\Sigma_{(2)} = \{0,1\}$ such that in every encoding the number of occurrences of zeroes is equal to the number of occurences of ones. Let $\kappa : \Sigma \longrightarrow \Sigma^*_{(2)}$ be a variable length encoding where for $1 \leqslant i \leqslant n$, $\kappa(s_i) = 0^i 1^i$. We fix some notations, let s is encoded as a binary string $b = b_1 b_2 b_3 \ldots b_j$ then j is called the length of the encoding denoted as $\ell(s) = j$ and $\kappa_i(s) = b_i$. Let b^R denote the reverse of string b, that is, $b^R = b_j b_{j-1} \ldots b_2 b_1$.

Steps for converting H_N to H'_M Consider the stack graph H_N constructed from the universal circuit $\mathcal{U}_m$ as discussed in Section 4. We delete the α_1, α_2 and α_3 vertices (and the edges incident on them) from graph H_N and we add 7[9] new vertices $\alpha'_1, \alpha'_2, \ldots, \alpha'_7$. We also add the following directed edges: $(t_\Delta, \alpha_7), (\alpha_7, \alpha_6), \ldots, (\alpha_3, \alpha_1), (\alpha, s_1)$.

For every symbol $s_i \in \Sigma$, there exist two directed edges (u_1, v_1) and (u_2, v_2) in stack graph H_N such that $\phi((u_1, v_1)) = \texttt{Push}(\texttt{s}_\texttt{i})$ and $\phi((u_2, v_2)) = \texttt{Pop}(\texttt{s}_\texttt{i})$. For every symbol $s_i \in \Sigma$, we make the following modifications to graph H_N. It is easy to note that the length of encoding of s_i, that is $\ell(s_i) = 2i$. For the sake of clarity, we assume $\ell(s_i) = j$.

1. We delete the edge (u_1, v_1). We add $2j-2$ new vertices, say $d_1, d_2, d_3, \ldots, d_{2j-2}$. We add the edges $\{(u_1, d_1)\} \cup \{(d_i, d_{i+1}) | 1 \leqslant i \leqslant (2j-3)\} \cup \{(d_{2j-2}, v_1)\} \cup \{(d_i, d_i) | 1 \leqslant i \leqslant 2j-2\}$. We set the labels on newly added edges to constant 1. We set $\phi(u_1, d_1) = \texttt{Push}(\kappa_\texttt{1}(\texttt{s}_\texttt{i}))$ and $\phi(d_{2k-2}, v_1) = \texttt{Push}(\kappa_\texttt{k}(\texttt{s}_\texttt{i}))$. For every even $t \in [2j-2]$, we set $\phi((d_t, d_{t+1})) = \texttt{Push}(\kappa_{\frac{t}{2}+1}(s_i))$. (see Figure 3)

2. We delete the edge (u_2, v_2). We add $2j-2$ new vertices, say $d'_1, d'_2, d'_3, \ldots, d'_{2j-2}$. We add the edges $\{(u_2, d'_1)\} \cup \{(d'_i, d'_{i+1}) | 1 \leqslant i \leqslant (2j-3)\} \cup \{(d'_{2j-2}, v_2)\} \cup \{(d'_i, d'_i) | 1 \leqslant i \leqslant 2j-2\}$. We set the labels on newly added edges to constant 1. We set $\phi(u_1, d'_1) = \texttt{Pop}(\kappa_\texttt{1}(\texttt{s}^\texttt{R}_\texttt{i}))$ and $\phi(d'_{2k-2}, v_1) = \texttt{Pop}(\kappa_\texttt{k}(\texttt{s}^\texttt{R}_\texttt{i}))$. For every even $t \in [2j-2]$, we set $\phi((d'_t, d'_{t+1})) = \texttt{Pop}(\kappa_{\frac{\texttt{t}}{2}+\texttt{1}}(\texttt{s}^\texttt{R}_\texttt{i}))$. (see Figure 4).

Finally, the labels of all the self-looped vertices of graph H'_M are changed from constant 1 to -1. We now state our main Lemma.

[8] It is not too hard to see that $\Delta + 4$ is a multiple of 4. (As Δ is a power of 2.)

[9] It is easy to see that adding 7 new vertices will make the total number of vertices a multiple of 8.

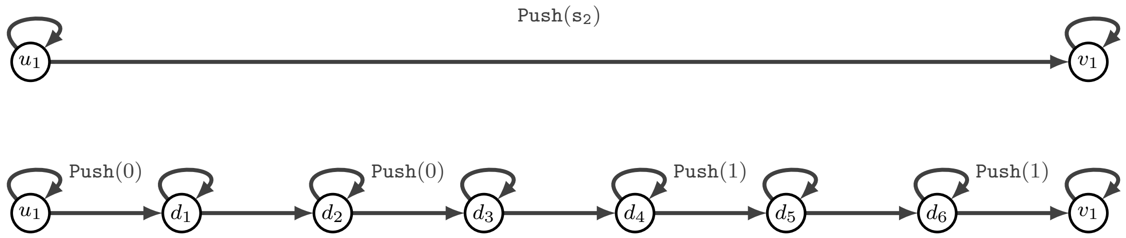

Figure 3: Edge (u_1, v_1) of graph H_N (upper figure) is transformed to a directed path from u_1 to v_1 in graph H'_M (lower figure)

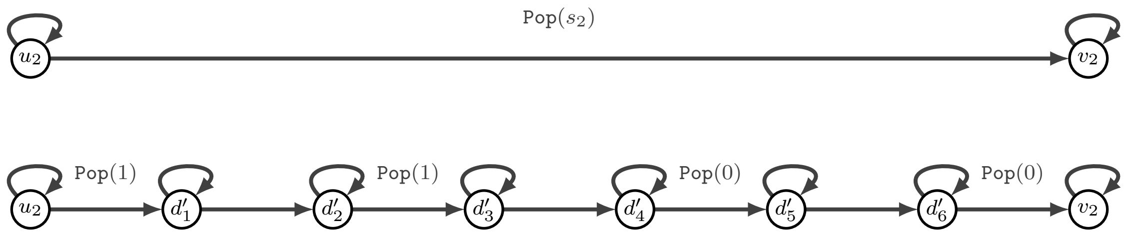

Figure 4: Edge (u_2, v_2) of graph H_N (upper figure) is transformed to a directed path from u_2 to v_2 in graph H'_M (lower figure)

Lemma 25. *Consider the stack graph H'_M constructed above*

1. *The signature of every stack-realizable cycle cover in H'_M is same, W.L.O.G, we assume it to be positive. Moreover, the sum of weights of stack-realizable cycle covers in H'_M is equal to $f_m(Y)$.*

2. *The sum of signed weights of stack-realizable clow-sequences which are not cycle covers equals* 0*.*

It is easy to see that the proof ideas of Lemma 24 can be used to prove Lemma 25. We therefore skip the details of the proof of Lemma 25.

Ordering of the vertices of H'_M To finish the proof of the VP-hardness of $\texttt{StackDet}_{\texttt{n}}^{(2)}(X)$, it is sufficient to describe the ordering of the vertices of graph H'_M such that edge label function ϕ in graph H'_M is consistent with the edge label function ϕ described in the Definition 8. We first order the vertices $\theta_1, \theta_2, \ldots, \theta_{\Delta-1}$ and $\alpha_1, \alpha_2, \ldots, \alpha_7$. It is easy to note that all of these vertices must appear in any cycle which is not a self-loop. If we traverse such a cycle starting from vertex α_1, we visit these vertices in a particular order, the order is $\langle \alpha_1, s_1, \theta_1, \theta_2, \ldots, \theta_{\Delta-1}, t_\Delta, \alpha_7, \alpha_6, \ldots, \alpha_1 \rangle$. We number these vertices in the reverse order, that is, α_1 gets numbered 1, α_2 gets numbered 2, α_3 gets numbered 3 and so on till α_7 gets numbered 7 and then t_Δ gets numbered 8 and so on till s_1 is numbered $\Delta + 8$. This numbering will ensure that all the edges which appears between these vertices gets labelled by `No-op`. It is easy to note that for large enough m, Δ is always some power of 2 and therefore a multiple of 8 and therefore, $\Delta + 8$ is a multiple of 8. Since, the total number of occurrences of zeroes and ones in every encoding is same, we know that the total number of occurrences of $\texttt{Push}(0), \texttt{Push}(1), \texttt{Pop}(0), \texttt{Push}(1)$ as edge labels is same in graph H'_M. Let δ be the total number of occurences of each of $\texttt{Push}(0), \texttt{Push}(1), \texttt{Pop}(0), \texttt{Push}(1)$. We now partition the set of all the push-pop-labelled edges (that is, edges which are not labelled by No-op) of graph H'_M into δ number of sets where each set consists of four edges, say, $(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4)$ such that $\Phi((u_1, v_1)) = \texttt{Push}(0), \Phi((u_2, v_2)) = \texttt{Push}(1), \Phi((u_3, v_3)) = \texttt{Pop}(0), \Phi((u_4, v_4)) = \texttt{Pop}(1)$. We label the elements of set δ as $\delta_1, \delta_2, \ldots, \delta_j$. For every δ_i, the tail of the edge (u_1, v_1) is numbered as $\Delta+8+8(i-1)+1$, the head of the edge (u_1, v_1) is numbered as $\Delta+8+8(i-1)+2$, the tail of the edge (u_2, v_2) is numbered as $\Delta+8+8(i-1)+3$, the head of the edge (u_2, v_2) is numbered as $\Delta+8+8(i-1)+4$, the tail of the edge (u_3, v_3) is numbered as $\Delta+8+8(i-1)+5$, the head of the edge (u_3, v_3) is numbered as $\Delta+8+8(i-1)+6$, the tail of the edge (u_4, v_4) is numbered as $\Delta+8+8(i-1)+7$ and the head of the edge (u_4, v_4) is numbered as $\Delta+8+8(i-1)+8$. It is easy to note that such an ordering will exhaust all the vertices of graph H'_M and it will also ensure that every vertex gets a distinct number and is consistent with Φ from Definition 8.

6 VNP-hardness of $\texttt{CountDet}_\texttt{n}(X)$ and $\texttt{CountDet}^{(2)}_\texttt{n}(X)$

In this section we first show that $\texttt{CountDet}_\texttt{n}(X)$ is hard for VNP. We will first show that the Permanent polynomial[10] can be computed as a projection of $\texttt{CountDet}_\texttt{n}(X)$. This will prove that $\texttt{CountDet}_\texttt{n}(X)$ is VNP-hard over fields of characteristic $\neq 2$. To show its hardness over fields of characteristic 2, we will show that it can compute another polynomial, namely EC^*_m, as a projection, where $n = \mathrm{poly}(m)$. This polynomial was shown to be VNP-complete over fields of characteristic 2 in [Hru15].

6.1 VNP-hardness of $\texttt{CountDet}_\texttt{n}(X)$ over fields of characteristic $\neq 2$

Let $Y = \{y_{1,1}, y_{1,2}, \ldots, y_{m,m}\}$. We will show that $\texttt{Perm}_m(Y)$ can be obtained as a projection of $\texttt{CountDet}_\texttt{n}(X)$, where $n = \mathrm{poly}(m)$. To prove this, we create a counter graph H_N, such that $N = \mathrm{poly}(m)$ and the following properties hold.

- All the counter-realizable cycle covers in H_N have the same sign.
- Moreover, the sum of the weights of the counter-realizable clow sequences which are cycle covers, equals $\texttt{Perm}_m$ and the sum of the signed weights of the counter-realizable clow sequences which are not cycle covers = 0.

Then by simple re-ordering of the vertices of H_N and adding edges to make it a complete graph G_n, as in the definition of $\texttt{CountDet}_\texttt{n}(X)$, we get that $\texttt{Perm}_m(Y)$ can be obtained as a projection of $\texttt{CountDet}_\texttt{n}(X)$.

In order to describe the construction of H_N, we first create $2m$ smaller counter graphs, $W_1, \ldots, W_m$ and $R_1, \ldots, R_m$. For each $i \in [m]$, $W_i = (V_i^w, E_i^w, \Sigma_i^w, \phi_i^w)$ is as follows.

- $V_i^w = \{s_i^w, t_i^w\} \cup \{u_{i,1}, \ldots, u_{i,m}\} \cup \{v_{i,1}, \ldots, v_{i,m}\}$. $E_i^w = \bigcup_{j \in [m]} \{(s_i^w, u_{i,j})\} \cup \bigcup_{j \in [m]} \{(v_{i,j}, t_i^w)\} \cup \bigcup_{j \in [m]} \{(u_{i,j}, v_{i,j})\}$. $\Sigma_i^w = \{\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,m}\}$.
- For each $j \in [m]$, $\phi_i^w((u_{i,j}, v_{i,j})) = \texttt{Write}(\alpha_{i,j})$. ϕ_i^w is No-op for all other edges in E_i^w.

Similarly, for each $i \in [m]$, $R_i = (V_i^r, E_i^r, \Sigma_i^r, \phi_i^r)$ can be described as follows.

- $V_i^r = \{s_i^r, t_i^r\} \cup \{a_{i,1}, \ldots, a_{i,m}\} \cup \{b_{i,1}, \ldots, b_{i,m}\}$. $E_i^r = \bigcup_{j \in [m]} \{(s_i^r, a_{i,j})\} \cup \bigcup_{j \in [m]} \{(b_{i,j}, t_i^r)\} \cup \bigcup_{j \in [m]} \{(a_{i,j}, b_{i,j})\}$. $\Sigma_i^r = \{\alpha_{1,i}, \alpha_{2,i}, \ldots, \alpha_{m,i}\}$. For each $j \in [m]$, $\phi_i^r((a_{i,j}, b_{i,j})) = \texttt{Read}(\alpha_{j,i})$. ϕ_i^r is No-op for all other edges in E_i^r.

[10]Recall that $\texttt{Perm}_m(Y) = \sum_{\sigma:\text{ permutation of [m]}} \prod_{i \in [m]} y_{i,\sigma(i)}$

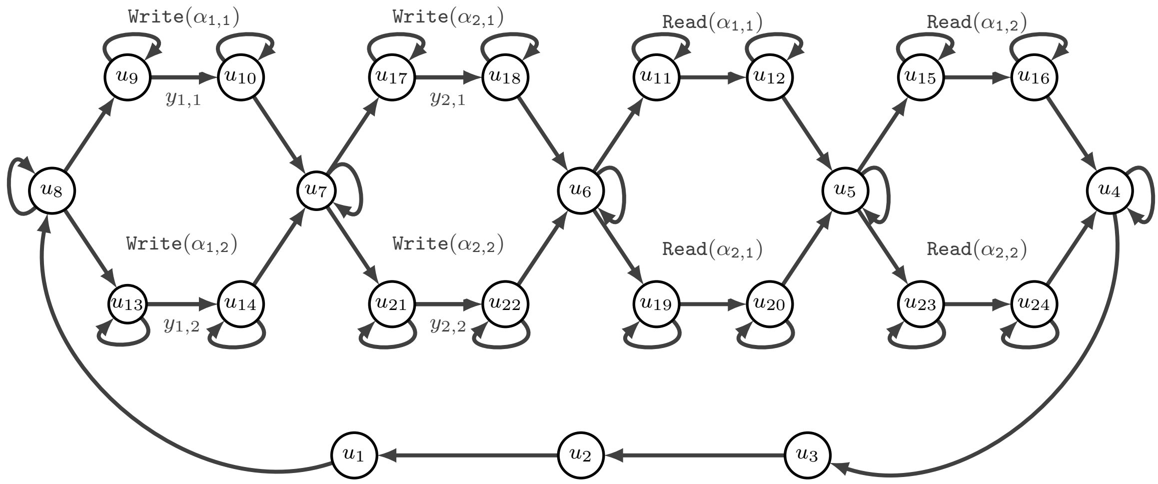

Figure 5: H_N for $m = 2$, all edges are labelled with constant 1

Let H'_N be the graph formed by identifying t^w_i with s^w_{i+1} for $1 \leqslant i \leqslant m-1$ and by identifying t^w_m with s^r_1 and also identifying t^r_i with s^r_{i+1} for $1 \leqslant i \leqslant m-1$. We also add labels on the edges of H'_N. We define $\mathcal{L}((u_{i,j}, v_{i,j})) = y_{i,j}$ for $i, j \in [m]$. For all other edges, $\mathcal{L}$ is set to 1. We first make the following observation about H'_N.

Claim 26. *For each monomial in $\mathcal{M}$ in $\texttt{Perm}_m(Y)$, there is a unique counter-realizable path π from s^w_1 to t^r_m in H'_N such that $\prod_{e \in \pi} \mathcal{L}(e) = \mathcal{M}$.*
For any counter-realizable path π from s^w_1 to t^r_m in H'_N, $\prod_{e \in \pi} \mathcal{L}(e)$ corresponds to a unique monomial of $\texttt{Perm}_m(Y)$.

Proof. From our construction of H'_N, it is easy to see that the vertices $t^w_1, t^w_2, \ldots, t^w_m$ and the vertices $t^r_1, t^r_2, \ldots, t^r_{m-1}$ are all cut-vertices in H'_N, and deleting any one of them disconnects the vertices s^w_1 and t^r_m. This means any path π from s^w_1 to t^r_m passes through the vertices t^w_i for $1 \leqslant i \leqslant m$ and t^r_i for $1 \leqslant i \leqslant m-1$. Therefore, π can be viewed as a composition of the $2m$ paths $\pi^w_1, \pi^w_2, \ldots, \pi^w_m, \pi^r_1, \pi^r_2, \ldots, \pi^r_m$ in that order, where π^w_i is the subpath of π between s^w_i and t^w_i for $1 \leqslant i \leqslant m$, and π^r_i is the subpath of π between s^r_i and t^r_i for $1 \leqslant i \leqslant m$. In fact, for any such $2m$ paths, their composition (in that order) is a path from s^w_1 to t^r_m in H'_N.

We now proceed with the proof of the claim. Any monomial $\mathcal{M}$ in $\texttt{Perm}_m(Y)$ is of the form $\prod_{i=1}^m y_{i,\sigma(i)}$, where σ is a permutation of $[m]$. The path π is constructed as follows: take π^w_i to be the path $s^w_i, u^w_{i,\sigma(i)}, v^w_{i,\sigma(i)}, t^w_i$, and π^r_i to be the path $s^r_i, u^r_{i,\sigma^{-1}(i)}, v^r_{i,\sigma^{-1}(i)}, t^r_i$, both for $1 \leqslant i \leqslant m$. Consider the sequence of counter operations along π, other than the $\texttt{No-op}$ operations. The only edges that have such operations are the edges $(u^w_{i,\sigma(i)}, v^w_{i,\sigma(i)})$ for $1 \leqslant i \leqslant m$ and $(u^r_{i,\sigma^{-1}(i)}, v^w_{i,\sigma^{-1}(i)})$ for $1 \leqslant i \leqslant m$. This implies that the counter operations encountered in π are $\texttt{Write}(\alpha_{1,\sigma(1)}), \texttt{Write}(\alpha_{2,\sigma(2)}), \ldots, \texttt{Write}(\alpha_{m,\sigma(m)}), \texttt{Read}(\alpha_{\sigma^{-1}(1),1}), \texttt{Read}(\alpha_{\sigma^{-1}(2),2}), \ldots,$ $\texttt{Read}(\alpha_{\sigma^{-1}(m),m})$ in that order. Now, σ is a permutation of $[m]$, so the pairs $(\sigma^{-1}(j), j)$ for $1 \leqslant j \leqslant m$ are a permutation of the pairs $(j, \sigma(j))$ for $1 \leqslant j \leqslant m$. Therefore, the m symbols read are exactly the m symbols written, possibly in a different order. Since the write operations all come before the read operations, this sequence of counter operations is indeed a counter-realizable sequence. Moreover, the only edges that have labels other than 1 are the edges $(u^w_{i,\sigma(i)}, v^w_{i,\sigma(i)})$ for $1 \leqslant i \leqslant m$, and these edges have labels $y_{i,\sigma(i)}$. Therefore, π is a counter-realizable path from s^w_1 to t^r_m in H'_N computing the monomial $\prod_{e \in \pi} \mathcal{L}(e) = \prod_{i=1}^m y_{i,\sigma(i)} = \mathcal{M}$.

Conversely, let π be a counter-realizable path from s_1^w to t_m^r in H'_N. For each $1 \leqslant i \leqslant m$, the path π_i^w is a path from s_i^w to t_i^w. Any such path clearly is of the form $s_i^w, u_{i,f_i}^w, v_{i,f_i}^w, t_i^w$ for some $1 \leqslant f_i \leqslant m$. Similarly, for each $1 \leqslant i \leqslant m$, the path π_i^r is a path from s_i^r to t_i^r of the form $s_i^r, u_{g_i,i}^r, v_{g_i,i}^r, t_i^r$ for some $1 \leqslant g_i \leqslant m$. We represent the j_is and k_is using two functions $f, g : [m] \to [m]$ defined as $f(i) = j_i$ for all $i \in [m]$ and $g(i) = k_i$ for all $i \in [m]$. From the previous paragraph, the sequence of counter operations along π other than `No-op` is $\texttt{Write}(\alpha_{1,f(1)}), \texttt{Write}(\alpha_{2,f(2)}), \ldots, \texttt{Write}(\alpha_{m,f(m)}), \texttt{Read}(\alpha_{g(1),1}), \texttt{Read}(\alpha_{g(2),2}), \ldots, \texttt{Read}(\alpha_{g(m),m})$ in that order. This sequence is counter-realizable, because π is a counter-realizable path.

For each $1 \leqslant j \leqslant m$, the operation $\texttt{Read}(\alpha_{g(j),j})$ appears in the sequence of counter operations. This means $\texttt{Write}(\alpha_{g(j),j})$ is an operation earlier in the sequence. The only such write operation appearing in the sequence is $\texttt{Write}(\alpha_{g(j),f(g(j))})$, so $f(g(j)) = j$. Similarly, for each $1 \leqslant j \leqslant m$, the operation $\texttt{Write}(\alpha_{j,f(j)})$ appears in the sequence of counter operations, which means $\texttt{Read}(\alpha_{j,f(j)})$ appears later in the sequence. The only such read operation appearing in the sequence is $\texttt{Read}(\alpha_{g(f(j)),f(j)})$, so $g(f(j)) = j$. Therefore $f(g(j)) = g(f(j)) = j$ for all $j \in [m]$, so f and g are both permutations of $[m]$ and are inverses of each other. We rewrite f as σ and g as σ^{-1}. The monomial computed by π is, therefore, $\prod_{e \in \pi} \mathcal{L}(e) = \prod_{i=1}^m y_{i,\sigma(i)}$, which is a monomial of $\texttt{Perm}_\texttt{m}(\texttt{Y})$. $\square$

We now construct the graph H_N from graph H'_N. If m is odd, then we add a vertex α and edges (α, s_1^w) and (t_m^r, α) and if m is even, then we add three vertices $\alpha_1, \alpha_2, \alpha_3$ and edges (α_1, s_1^w) (α_2, α_1), (α_3, α_2) and (t_m^r, α_3) (see figure 5). This ensures that, $N = 4n$ for some parameter n, where N is the number of vertices in H_N. We set the weights of all the extra added edges as 1 and label it with `No-op`. We now add self-loops on all the vertices with weight 1 except the α vertices. All the self-loop edges have the label of `No-op` on it. Consider the counter graph H_N constructed as above, we will argue that the sum of weights of counter-realizable cycle covers in H_N equals the $\texttt{Perm}_m(Y)$ and the sum of signed weights of counter-realizable clow sequences that are not cycle covers equals 0. Without loss of generality, we assume that m is odd. Similarly, we can extend our arguments for even m.

We already know from Claim 26 that there exists a bijection between the set of monomials in $\texttt{Perm}_m(Y)$ and the set of counter-realizable paths between s_1^w to t_m^r in graph H'_N. It is therefore sufficient to show a bijection between the set of counter-realizable paths between s_1^w to t_m^r in graph H'_N and the set of all counter-realizable cycle covers of graph H_N. We also argue that the sign of every cycle cover of graph H_N is same (w.l.o.g., we assume it to be positive). Since, α is a vertex in H_N without any self loop, any cycle cover $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_k \rangle$ of H_N must cover α with some cycle, w.l.o.g., we call it $\mathcal{C}_1$ which have both the edges (α, s_1^w) and (t_m^r, α), and all other cycles in $\mathcal{C}$ are self-loops on all the vertices which are not covered in cycle $\mathcal{C}_1$. It is easy to observe that the length of any cycle in H_N which uses vertex α is always equal to $6m + 2$. Therefore, the total number of vertices of graph H_N which are not covered in this long cycle and which will get covered by self loops in any cycle cover is $N - 6m - 2$. It immediately follows that the sign of every cycle cover of graph H_N is same.

We now show a bijection between the set of all counter-realizable cycle covers of graph H_N and the set of counter-realizable paths between s_1^w to t_m^r in graph H'_N. It is easy to see that a cycle cover $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_k \rangle$ of H_N is counter-realizable iff the long cycle which uses the vertex α is counter-realizable, w.l.o.g., we call the long cycle $\mathcal{C}_1$. It is easy to note that, for any counter-realizable cycle cover $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_k \rangle$, the cycle $\mathcal{C}_1$ must be formed by an edge (α, s_1^w), followed by a unique counter-realizable directed path $\mathcal{P}$ between s_1^w to t_m^r (of graph H'_N), followed by an edge (t_m^r, α). Also, for every counter-realizable directed path $\mathcal{P}$ from s_1^w to t_m^r (of graph H'_N), one can form a unique counter-realizable long cycle (and therefore a counter-realizable cycle cover) in H_N where the long cycle is (α, s_1^w), followed by directed path

$\mathcal{P}$ between s_1^w to t_m^r, followed by (t_m^r, α). This finishes the first part of our argument.

We now argue that the sum of signed weights of all counter-realizable clow sequences of graph H_N which are not cycle covers is equal to 0. We first argue that there exist no clow sequence in graph H_N which does not contain the vertex α in any of its clow. Suppose there exist some clow which does not contain α then we consider the graph formed by deleting the vertex α in H_N, in such a graph, the only closed walks possible are single-loops on each vertex of such a graph. But the total degree of such a clow sequence can never be equal to N, therefore, such a clow sequence is not a valid clow sequence. Let us assume that α is the least numbered vertex in H_N, say, numbered with 1. Since, α is a vertex without a self-loop, any clow involving α must have edges (α, s_1^w) and (t_m^r, α). It is easy to see that in H_N, any counter-realizable clow sequence, say, $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k \rangle$ satisfies the property that except the first clow $\mathcal{C}_1$ (which involves the vertex α), all other clows in the clow sequence $\mathcal{C}$ are self-loops. α_1 will be the head of $\mathcal{C}_1$. It is crucial to note that since all self–loops in graph H_N are labelled with `No-op`, the clow sequence $\mathcal{C}$ is counter-realizable iff the first clow $\mathcal{C}_1$ is counter-realizable.

We can now use similar ideas discussed in part 2 of the proof of Claim 24 to show that there always exists an involution ψ on the set of counter-realizable clow sequences of graph H_N such that ψ will map any counter-realizable cycle cover to itself and for any counter-realizable clow sequence which is not a cycle cover, say $\mathcal{C}$, there exist another counter-realizable clow sequence which is not a cycle cover, $\mathcal{C}'$ such that $\psi(\mathcal{C}) = \mathcal{C}'$ and $\psi(\mathcal{C}') = \mathcal{C}$ and the monomials associated with both $\mathcal{C}$ and $\mathcal{C}'$ are same but their signatures are opposite.

Obtaining G_n from H_N. To obtain G_n from this H_N, we need to give an ordering on the vertices that is consistent with Definition 7 and ensure that G_n is a complete graph. Ensuring the latter is easy. We add all the missing edges and set $\mathcal{L}$ value for them to 0.

To describe the ordering, let us first assume that m is odd. When m is even, the ordering can be worked out similarly. We first introduce some notation. For $1 \leqslant i \leqslant m-1$ let us denote the vertex obtained by fusing t_i^w with s_{i+1}^w by θ_i. Let us denote the vertex obtained by fusing t_m^w with s_1^r by θ_m. Also for $1 \leqslant i \leqslant m-1$, let us denote the vertex obtained by fusing t_i^r with s_{i+1}^r by θ'_i.

The ordering can now be described as follows. Vertex α is set to 1 and the vertex t_m^r is set to 2. The vertices θ'_1 to θ'_{m-1} are numbered in reverse order, i.e. θ'_{m-1} is set to 3, θ'_{m-2} to 4 and so on up to θ'_1 is set to $m+1$. We also number the vertices θ_1 to θ_m in reverse order starting from $2m+1$ down to $m+2$. We number the vertex s_1^w as $2m+2$[11]

Now, let us assume that the symbols in Σ are ordered in some arbitrary order, say $a_1, \ldots, a_{m^2}$. In H_N, let $\mathcal{E}$ be defined as $\{e \mid \phi(e) \neq \texttt{No-op}\}$. From our construction of H_N, no two edges in $\mathcal{E}$ share any endpoints. Also for each $a_i \in \Sigma$, there is a unique edge with $\texttt{Write}(a_i)$ on it and a unique edge with $\texttt{Read}(a_i)$ on it. We now fix the following ordering: the tail of the edge with $\texttt{Write}(a_i)$ on it is assigned $4(i-1)+1+(2m+2)$ and its head is assigned $4(i-1)+2+(2m+2)$, the tail of the edge with $\texttt{Read}(a_i)$ on it is assigned $4(i-1)+3+(2m+2)$ and finally, its head is assigned $4(i-1)+4+(2m+2)$.

6.2 VNP-hardness of $\texttt{CountDet}_\texttt{n}(X)$ over fields of characteristic $= 2$

Over characteristic 2, $\texttt{Perm}_m$ is known to be easy. Therefore, in order to prove VNP-hardness over characteristic 2 fields, we use a different VNP-hard polynomial. This polynomial was shown to be VNP-hard over characteristic 2 fields in a work of Hrubes [Hru15]. The polynomial is based on the algebraic variant of the well-known Edge Cover problem. We start by defining the polynomial.

[11]For an odd m, note that $2m+2$ is always a multiple of 4.

Definition 27. *Let $m = \binom{\tau}{2}$ for some parameter τ. Let $G = (V, E)$ be a complete undirected graph on τ vertices, i.e. $V = [\tau]$ and $E = \{(i,j) \mid 1 \leqslant i < j \leqslant \tau\}$ and let edge $e = (i,j)$ be labelled with $y_{i,j}$.* $\mathtt{EC}^*_m(Y) = \sum_{E' \subseteq E, E' \text{ is an edge cover}} \prod_{(i,j)\in E', i<j} y_{i,j}$

In [Hru15], the above polynomial was shown to be VNP-hard. We will show that we can write $\mathtt{EC}^*_m(Y)$ as a projection of $\mathtt{CountDet_n}(X)$, where $n = \text{poly}(m)$.

For this, we will define $m+1$ counter graphs, $W, R_1, \ldots, R_m$, which when interconnected appropriately will give us another counter graph H_N, where $N = \text{poly}(m)$ and it has the following two properties.

- All the counter-realizable clow sequences in H_N have the same sign.
- Moreover, the sum of the weights of the counter-realizable clow sequences which are cycle covers equals $\mathtt{EC}^*_m$ and the sum of the signed weights of the counter-realizable clow sequences which are not cycle covers $= 0$.

Construction of W. For each edge $(i,j) \in E$ such that $1 \leqslant i < j \leqslant \tau$, we add a directed path $\rho_{i,j} = \langle (s_{i,j}, i), (i,j), (j, t_{i,j}) \rangle$ in W. We will call $s_{i,j}$ the source of $\rho_{i,j}$ and $t_{i,j}$ the sink of $\rho_{i,j}$. We arrange these paths in a linear order $\rho_{1,2}, \ldots, \rho_{1,\tau}, \rho_{2,3}, \ldots, \rho_{2,\tau}, \ldots \rho_{\tau-1,\tau}$ one after the other. Addtionally, we do the following. We rename $s_{1,2}$ as s_1 and $t_{\tau-1,\tau}$ as t_τ

- Add edges from s_1 to all the other sources, i.e. $\forall 1 \leqslant i < j \leqslant \tau$, add edge $(s_1, s_{i,j})$.
- Add edges from the sink of all the paths to t_τ, i.e. $\forall 1 \leqslant i < j \leqslant \tau$, add $(t_{i,j}, t_\tau)$.
- Also add edges from sink of a path to the sources of all the paths that come after it in the above order.
- We define $\phi((s_{i,j}, i)) = \mathtt{Write}(\alpha_{i,j})$ and $\phi((j, t_{i,j})) = \mathtt{Write}(\alpha_{j,i})$ for all $i \leqslant 1 < j \leqslant \tau$. We also define $\phi((s_1, 1)) = \mathtt{Write}(\alpha_{1,2})$ and $\phi((\tau, t_\tau)) = \mathtt{Write}(\alpha_{\tau,\tau-1})$ For all the other edges we set ϕ to be $\mathtt{No\text{-}op}$. We also assign $\mathcal{L}((i,j)) = y_{i,j}$ (See Figure 6).

Let π be any path from s_1 to t_τ in W. We will say that an edge (i,j) of G is traversed in π if $\rho_{i,j}$ is in π, i.e. all the three edges in $\rho_{i,j}$ are traversed in π. It is easy to see the following property holds.

Observation 28. *Let $S \subseteq E$, then there is a path π_S in W such that it traverses exactly the set of edges in S. Moreover, if S is an edge cover then for each vertex $i \in [\tau]$ we would have at least one edge $(i,j) \in S$ such that upon traversing the path π_S we would have done* $\mathtt{Write}(\alpha_{i,j})$ *and* $\mathtt{Write}(\alpha_{j,i})$ *for that edge.*

Construction of $R_1, \ldots, R_\tau$. We have one graph R_i for each vertex $i \in [\tau]$. This graph will allow for reading the symbols $\alpha_{i,j}$ for all $j \neq i$. For each $i \in [\tau]$, we describe $R_i = (V_i, E_i, \Sigma_i, \phi_i)$.

- $V_i = \bigcup_{j \in [\tau]-\{i\}} \{a_{i,j}\} \cup \bigcup_{j \in [\tau]-\{i\}} \{b_{i,j}\}$. Let *min* and *max* denotes the minimum and maximum number in $[\tau] - \{i\}$. $E_i = \{(a_{i,j}, b_{i,j}) \mid j \in [\tau] - \{i\}\} \cup \{(a_{i,min}, a_{i,j'}) \mid j' \in [\tau] - \{i\} \text{ and } j' > min\} \cup \{(b_{i,j}, b_{i,max}) \mid j \in [\tau] - \{i\} \text{ and } j < max\} \cup \bigcup_{k \in [\tau]-\{i\}} \{(b_{i,k}, a_{i,j}) | j > k\}$. $\phi((a_{i,j}, b_{i,j})) = \mathtt{Read}(\alpha_{i,j})$, where $j \neq i$ and ϕ for all the other edges is $\mathtt{No\text{-}op}$ (See Figure 7).

We relabel $a_{i,min}$ as a^*_i and we relabel $b_{i,max}$ as b^*_i. We observe the following properties about R_i.

Observation 29. *1. Let π be any path from a^*_i to b^*_i. There exists at least one $j \in [\tau], j \neq i$ such that we encounter* $\mathtt{Read}(\alpha_{i,j})$ *along π.*

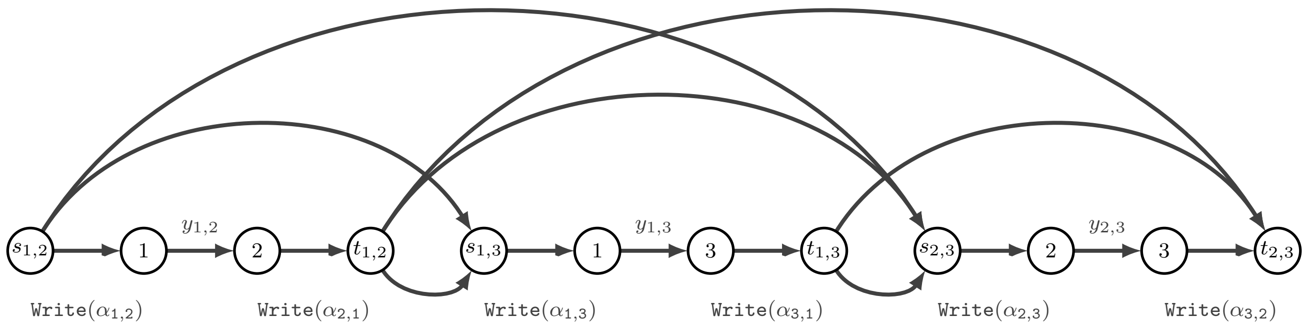

Figure 6: Construction of W for $\tau = 3$ and $m = 3$. For all $1 \leqslant i < j \leqslant 3$, we set $\phi(s_{i,j}, i) = \texttt{Write}(\alpha_{i,j})$ and $\phi(j, t_{i,j}) = \texttt{Write}(\alpha_{j,i})$ and $\mathcal{L}((i,j)) = y_{i,j}$

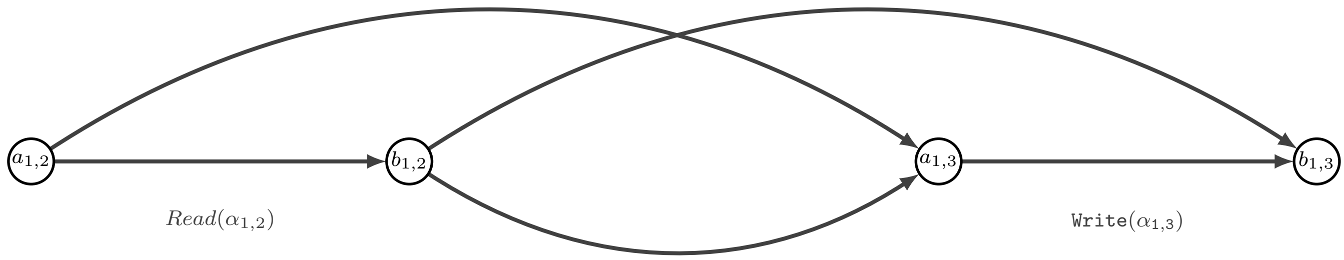

Figure 7: Construction of R_1 when $m = 3$. All edges are labelled with constant 1

2. *Let $S \subseteq [\tau]\backslash\{i\}$, there exists a path from a_i^* to b_i^*, say π_S, that encounters exactly the set $\{\texttt{Read}(\alpha_{i,j}) \mid j \in S\}$ along it.*

Construction of H_N. We now interconnect W and $R_1, \ldots, R_\tau$ to create the graph H_N as follows. We add an edge from t_τ to a_1^*. We also add edges from b_i^* to a_{i+1}^* for $1 \leqslant i \leqslant \tau - 1$. Finally, we add an edge from b_τ^* to s_1 and self-loops on all nodes other than b_τ^* and s_1. We first observe the following properties about H_N.

Claim 30. *Let Π be any counter-realizable path from s_1 to b_τ^* in H_N. The product of the Y variables along Π, corresponds to a unique monomial in $\texttt{EC}_m^*(Y)$.*

Conversely, if $\mathcal{M}$ is a monomial in $\texttt{EC}_m^*$ then there is a unique counter-realizable path Π in H_N from s_1 to b_τ^* such that the product of Y variables along Π equals $\mathcal{M}$.

Proof. Let Π be a counter-realizable path from s_1 to b_τ^* in H_N. Clearly, Π is obtained by concatenating the following paths and edges in this order: $\pi_0 \cdot (t_\tau, a_1^*) \cdot \pi_1 \cdot (b_1^*, a_2^*) \pi_2 \ldots \cdot (b_{\tau-1}^*, a_\tau^*) \cdot \pi_\tau$, where π_0 is a directed path from s_1 to t_τ in W and π_i is a directed path from a_i^* to b_i^* in R_i for $1 \leqslant i \leqslant \tau$.

By part 1 of Observation 29, we know that for each $i \in [\tau]$, π_i must encounter $\texttt{Read}(\alpha_{i,j})$ for at least one $j \neq i$. As the path is counter-realizable, there will not be any read operation that does not have a corresponding write operation before it. Let S be a subset of edges of G that are traversed in π_0. As all reads must find a corresponding write along π_0, we have that for each vertex i, there must be at least one edge (i, j) in S, which results into $\texttt{Write}(\alpha_{i,j})$ and

$\mathtt{Write}(\alpha_{j,i})$ along π_0. Therefore, S must be an edge cover. Hence the monomial obtained by taking product of the Y variables along the path gives rise to a monomial $\mathtt{EC}^*_m$.

Conversely, let $\mathcal{M}$ be a monomial in $\mathtt{EC}^*_m$. Then $\mathcal{M}$ corresponds to a subset of edges S of E that forms an edge cover of G. By Observation 28 we know that there exists a unique path in W that traverses exactly the set of edges in S. Let us call this path π_S. As S is an edge cover, we know that for each i in the vertex set of G, there exists at least one $j \neq i$ such that $\mathtt{Write}(\alpha_{i,j})$ occurs in π_S. For $i \in [\tau]$, let $U_i = \{\alpha_{i,j} \mid j \in [\tau]\backslash\{i\}$ and $\mathtt{Write}(\alpha_{i,j})$ occured along $\pi_S\}$. We know that $|U_i| \geqslant 1$ for $i \in [\tau]$. From the second part of Observation 29 we know that we can uniquely append π_S with paths $\pi_{U_1}, \pi_{U_2}, \ldots \pi_{U_\tau}$, where π_{U_i} is the unique path between a^*_i and b^*_i that traverses the set U_i. Therefore the path $\pi_S \cdot (t_\tau, a^*_1) \cdot \pi_{U_1} \cdot (b^*_1, a^*_2)\pi_{U_2} \ldots \cdot (b^*_{\tau-1}, a^*_\tau) \cdot \pi_{U_\tau}$ is a counter-realizable path and the product of the Y variables along it gives rise to the monomial $\mathcal{M}$. $\square$

Consider the counter graph H_N as defined in Section 6.2. We will argue that the sum of weights of counter-realizable cycle covers in H_N equals the $\mathtt{EC}^*_m(Y)$ and the sum of signed weights of counter-realizable clow sequences which are not cycle covers equals 0. We already know from Claim 30, that there exists a bijection between the set of monomials in $\mathtt{EC}^*_m(Y)$ and the set of counter-realizable paths between s_1 to b^*_τ in graph H_N. It is therefore sufficient to show that there exists a bijection between the set of counter-realizable paths between s_1 to b^*_τ in graph H_N and the set consisting of all counter-realizable cycle covers of graph H_N. Since, we are working on fields of characteristic 2, sign of every cycle cover of graph H_N is positive.

We now show a bijection between the set of all counter-realizable cycle covers of graph H_N and the set of counter-realizable paths between s_1 to b^*_τ in graph H_N. It is easy to note that any cycle cover $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_k \rangle$ of H_N must use the edge (b^*_τ, s_1), this is because, if it does not use this edge, then there is no way to cover vertices s_1 and b^*_τ by any other cycle in cycle cover $\mathcal{C}$ in graph H_N. We assume that $\mathcal{C}_1$ is the cycle in cycle cover $\mathcal{C}$ which uses the edge (b^*_τ, s_1). It is also easy to note that all the vertices which are not covered in $\mathcal{C}_1$ must be covered by self-loops on each of them in cycle cover $\mathcal{C}$, that is, in other words, all cycles in the cycle cover $\mathcal{C}$, except $\mathcal{C}_1$ are all self-loops. This is because, after deleting vertices b^*_τ and s_1, the only cycles left in graph H_N are self-loops. It is easy to see that for any counter-realizable clow sequence $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_k \rangle$, the cycle $\mathcal{C}_1$ must be formed by an edge (b^*_τ, s_1), followed by a unique counter-realizable directed path $\mathcal{P}$ between s_1 to b^*_τ of graph H_N. Also, for every counter-realizable directed path $\mathcal{P}$ between s_1 to b^*_τ, one can form a unique counter-realizable long cycle (and therefore a counter-realizable cycle cover) in H_N where the long cycle is formed by an edge (b^*_τ, s_1), followed by a unique counter-realizable directed path $\mathcal{P}$ between s_1 to b^*_τ of graph H_N. This finishes the first part of our argument.

We now prove that the sum of all signed weights of counter-realizable clow sequences of graph H_N which are not cycle covers is equal to 0. We first show that there exist no clow sequence in graph H_N without using the vertex s_1 in any of its clow. For the sake of contradiction, let us assume that there exists a clow sequence $\widehat{\mathcal{C}} = \langle \widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}'_2, \widehat{\mathcal{C}}'_3, \ldots, \widehat{\mathcal{C}}'_k \rangle$ which does not use vertex s_1. We now consider the graph formed by deleting the vertex s_1 in H_N, in such a graph, the only closed walks possible are single-loops on each vertex of such a graph. It is easy to see that the total degree of $\widehat{\mathcal{C}}$ is always less than N, therefore, $\widehat{\mathcal{C}}$ is not a valid clow sequence. Let us assume that s_1 is the least numbered vertex in H_N, say, numbered with 1.

Since, s_1 is a vertex without a self-loop, any clow which uses s_1 must have the edge (b^*_τ, s_1). It is easy to see that any counter-realizable clow sequence, say, $\mathcal{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k \rangle$ consists of the big cycle $\mathcal{C}_1$ (which uses the vertex s_1) and all other clows in $\mathcal{C}$ are self-loops. s_1 will be the head of $\mathcal{C}_1$. It is crucial to note that since all self–loops in graph H_N are labelled with $\mathtt{No\text{-}op}$, the clow sequence $\mathcal{C}$ is counter-realizable iff the clow $\mathcal{C}_1$ is counter-realizable.

Using similar ideas discussed in part 2 of the proof of Claim 24, it can be shown that there exists an involution ψ defined on the set of all counter-realizable clow sequences of graph H_N, such that the function ψ maps every counter-realizable clow sequence which is also a cycle cover to itself and for any counter-realizable clow sequence $\mathcal{C}$, which is not a cycle cover, there exists a counter-realizable clow sequence $\mathcal{C}'$ which is also not a cycle cover such that the monomials associated with both $\mathcal{C}$ and $\mathcal{C}'$ are same but with opposite signatures, also, $\psi(\mathcal{C}) = \mathcal{C}'$ and $\psi(\mathcal{C}') = \mathcal{C}$.

Ordering of the vertices of H_N To finish the proof we also show that we can give a complete ordering of the vertices of H_N consistent with Definition 7 and turn it into a complete graph to obtain G_n to fit the dscription of G_n as in Definition 7. To make it a complete graph, simply add all the missing edges and assign $\mathcal{L}$ for the newly added edges to 0. To get the ordering, fix any arbitrary ordering for the set Σ, the stack alphabet of H_N, say $a_1, a_2, \ldots, a_{|\Sigma|}$. In H_N, let $\mathcal{E}$ be defined as $\{e \mid \phi(e) \neq \texttt{No-op}\}$. From our construction of H_N, no two edges in $\mathcal{E}$ share any endpoints. Also for each $a_i \in \Sigma$, there is a unique edge with $\texttt{Write}(a_i)$ on it and a unique edge with $\texttt{Read}(a_i)$ on it. We now fix the following ordering: the tail of the edge with $\texttt{Write}(a_i)$ on it is assigned $4(i-1)+1$ and its head is assigned $4(i-1)+2$, the tail of the edge with $\texttt{Read}(a_i)$ on it is assigned $4(i-1)+3$ and finally, its head is assigned $4(i-1)+4$.

Remark 31. VNP-*hardness of* $\texttt{CountDet}_{\texttt{n}}^{(2)}(X)$*: The ideas used in Section 5 to prove the VP-hardness of* $\texttt{StackDet}_{\texttt{n}}^{(2)}(X)$ *can be adopted to prove the VNP-hardness of* $\texttt{CountDet}_{\texttt{n}}^{(2)}(X)$*. The proof will broadly consist of three parts:*

1. *We encode the symbol set Σ into binary strings using binary alphabet $\Sigma^{(2)}$. This encoding is exactly same as we discussed in Section 5*

2. *In the second step, we convert the graph H_N (H_N is the graph from Section 6.1 in case of VNP-hardness for fields of char≠2 and from Section 6.2 in case of VNP-hardness for fields of char=2) and modify it to another graph H'_M such that $M = 8n$, for some n. This is done on similar lines as in Section 5.*

3. *Finally, we order and number the vertices of graph H'_M such that the ordering will ensure that every vertex in H'_M gets a distinct number and is consistent with Φ from Definition 9. This is again on similar lines as in Section 5.*

We skip the other details of this proof.

Acknowledgement : We would like to thank Nitin Saurabh for his useful comments on the initial draft of this paper

References

[CLV19] Prasad Chaugule, Nutan Limaye, and Aditya Varre. Variants of homomorphism polynomials complete for algebraic complexity classes. In *Computing and Combinatorics - 25th International Conference, COCOON*, volume 11653 of *LNCS*, pages 90–102. Springer, 2019.

[DMM+14] Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. Homomorphism polynomials complete for vp. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 29, 2014.

[Eng16] Christian Engels. Dichotomy theorems for homomorphism polynomials of graph classes. *Journal of Graph Algorithms and Applications*, 20(1):3–22, 2016.

[Hru15] Pavel Hrubes. On hardness of multilinearization, and VNP completeness in characteristics two. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:67, 2015.

[Men11] Stefan Mengel. Characterizing arithmetic circuit classes by constraint satisfaction problems. In *ICALP*, pages 700–711. Springer, 2011.

[Men13] Stefan Mengel. Arithmetic branching programs with memory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 667–678. Springer, 2013.

[MS18] Meena Mahajan and Nitin Saurabh. Some complete and intermediate polynomials in algebraic complexity theory. *Theory of Computing Systems*, 62(3):622–652, 2018.

[MV97] Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. Technical report, 1997.

[Raz08] Ran Raz. Elusive functions and lower bounds for arithmetic circuits. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 711–720. ACM, 2008.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010.

[Val79] L. G. Valiant. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 249–261, 1979.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il