Skew Circuits of Small Width

Nikhil Balaji[a,1], Andreas Krebs[b], Nutan Limaye[a]

[a] *Indian Institute of Technology, Bombay, India.*
[b] *University of Tübingen, Germany.*

Abstract

A celebrated result of Barrington (1985) proved that polynomial size, width-5 branching programs (BP) are equivalent in power to a restricted form of branching programs – polynomial sized width-5 permutation branching programs (PBP), which in turn capture all of NC^1. On the other hand it is known that width-3 PBPs require exponential size to compute the AND function. No such lower bound is known for width-4 PBPs, however it is widely conjectured that width-4 PBPs will not capture all of NC^1. In this work, we study the power of bounded width branching programs by comparing them with bounded width skew circuits.

It is well known that branching programs of bounded width have the same power as skew circuit of bounded width. The naive approach converts a BP of width w to a skew circuit of width w^2. We improve this bound and show that BP of width $w \geq 5$ can be converted to a skew circuit of width 7. This also implies that skew circuits of bounded width are equal in power to skew circuits of width 7. For the other way, we prove that for any $w \geq 2$, a skew circuit of width w can be converted into an equivalent branching program of width w. We prove that width-2 skew circuits are not universal while width-3 skew circuits are universal and that any polynomial sized CNF or DNF is computable by width 3 skew circuits of polynomial size. It is known that Parity does not have small CNFs or DNFs. It is easy to see that Parity has width-4 skew circuits.

We prove that a width-3 skew circuit computing Parity requires exponential size. This gives an exponential separation between the power of width-3

Email addresses: nbalaji@cse.iitb.ac.in (Nikhil Balaji), mail@krebs-net.de (Andreas Krebs), nutan@cse.iitb.ac.in (Nutan Limaye)

[1] This work was done while the author was a graduate student at the Chennai Mathematical Institute

skew circuits and width-4 skew circuits.

Keywords: Branching Programs, Barrington's Theorem, Skew Circuits, Lower bounds, Parity

---

1. Introduction

The class NC captures the hierarchy of decision problems that are solvable efficiently by Boolean circuit families of polylogarithmic depth and polynomial size. The question of whether NC is strictly contained in P is an important open question in Complexity theory. In fact, it is conjectured that the inclusion between the classes in this hierarchy namely, NC^i for $i \geq 1, i \in \mathbb{N}$ (problems solvable by Boolean circuit families of bounded fan-in, $O(\log^i n)$-depth and polynomial size) is strict.

In this paper, our focus is on NC^1. NC^1 occupies a central position in the study of small depth circuits. NC^1 consists of functions computable by circuit families built with bounded fan-in AND, OR and NOT gates of logarithmic depth and polynomial size. NC^1 itself is contained in Logspace [1] and many important integer arithmetic operations like iterated addition, multiplication, division and powering are in uniform NC^1 [2, 3]. In fact, all regular languages are contained in NC^1 and there is a regular language that is NC^1-hard [4]. However there are hardly any lower bounds known against NC^1.

Several useful characterizations of NC^1 has emerged over the years:

1. Any function computable by a polynomial size Boolean formula is in NC^1 [5].
2. Uniform NC^1 is exactly those class of languages recognized by Alternating Turing Machines in logarithmic time [6].
3. NC^1 contains exactly those functions that are computable by branching programs of bounded width [7].
4. NC^1 contains exactly those regular languages that are characterized by having a monoid containing a non-solvable group [8].

Our interest in NC^1 is motivated by the celebrated result of Barrington [7], that Branching Programs of width 5 are sufficient to capture NC^1 in its entirety. We take a brief detour to discuss the status of circuit and branching program lower bounds discovered so far, before stating our results.

1.1. Circuit Lower Bounds

In the 1980's a flurry of results gave superpolynomial lower bounds for explicit functions against even more restricted circuit classes namely, $\mathsf{AC}^0, \mathsf{ACC}^0$ and TC^0. Håstad [9], building on works of Furst, Saxe, Sipser [10] and Yao [11], showed that constant depth circuits over AND, OR and NOT gates require exponential size to compute the parity function. The main tool in proving these results was the method of random restrictions. Here one argues that a constant depth over AND, OR and NOT gates will simplify to a constant function with high probability when some of its variables are set to constants, and noting that parity is not a function that shares this property.

Razborov [12] and Smolensky [13] further strengthened these lower bounds to prove that even if we augment AC^0 circuits with MOD_p gates (a MOD_m gate outputs 1 if the number of 1's input to it is 0 modulo m), they cannot compute the MOD_q function (and also the Majority function) for distinct primes p, q. They derived these lower bounds via the *polynomial method* [14]. However, when m is composite, all the lower bound techniques devised so far grind to a halt, and we have no strong lower bounds for a function in NP against ACC^0, even though it is widely believed that such circuits cannot even compute the Majority function. Recently, drawing on decades of work in complexity theory, Williams proved that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ [15].

1.2. Branching Program Lower Bounds

Branching programs were originally introduced to understand space bounded computation. They were first defined in [16] and formally studied by Masek in his thesis [17]. Borodin et al.[18] proved that AC^0 is contained in the class of functions computed by bounded width branching programs and conjectured that Majority cannot be computed by them. In a surprising result, Barrington showed that in fact, width 5 branching programs can compute all of NC^1 and hence the Majority function.

After the strong lower bound results discussed above for AC^0 and ACC^0, the question of proving lower bounds for NC^1 gained a lot of attention. However this has turned out to be a notorious open problem, though in general the state of affairs in branching programs is marginally better than that of Boolean circuits (though, of course, no super polynomial lower bounds are known for any explicit function in NP). We refer the reader to two excellent sources – Razborov's survey [19] and Jukna's book [20] – for an overview of branching program lower bounds.

The branching program characterization of NC^1 has provided an avenue to understand the power of classes that reside inside NC^1. Though proving lower bounds for width 5 branching programs is equivalent to proving lower bounds for NC^1, it is conceivable that proving lower bounds for width 4 branching programs is easier. In this regard, it is known [21] that width 3 branching programs of a restricted type (permutation branching programs) require exponential size to compute the AND function. It is conjectured, but not known so far that width-4 permutation branching programs cannot compute the AND function. It is worthwhile to contrast this against the situation at width 5, where permutation branching programs are known to be as powerful as general branching programs of width 5 and hence NC^1 itself.

1.3. Skew Circuits

Skew circuits were originally introduced by Venkateswaran [22] to give uniform circuit characterizations of nondeterministic time and space classes. Using these, he turns questions about complexity classes like $\mathsf{P}, \mathsf{NP}, \mathsf{PSPACE}$ to questions about the relative power of Boolean circuits with structural restrictions. The Branching Program model lends itself to an easy skew circuit characterization and vice-versa. Arithmetic skew circuits have been extensively investigated in relation to the Determinant in the algebraic complexity setting and there are a host of interesting upper/lower bounds known for them [23, 24, 25, 26].

1.4. Our Results

Bounded width branching programs can be equivalently thought of as bounded width skew circuits (see for example [27]). In this paper, we take a closer look at this relationship. The folklore construction[2] converts a polynomial size branching program of width w into a polynomial size skew circuit of width w^2. We improve this construction and show that any bounded width branching program of width greater than or equal to 5 can be converted into an equivalent skew circuit of width 7. We also study the conversion of skew circuits into branching programs. Here, the known construction converts a skew circuit of width w into a branching program of width $w+1$ [27]. We improve this construction and prove that a polynomial size skew circuit of width w can be converted into a polynomial size branching program of

[2]Replace each wire by an AND gate and each node by an OR gate.

width w. These results prove that width 7 skew circuits of polynomial size characterize NC^1.

These structural results allow us to examine the set of languages in NC^1 by varying the width of skew circuits between 1 and 7. Like for permutation branching programs, some natural questions arise for bounded width skew circuits. We start by examining the power of width 2 skew circuits. We observe that they are not universal as they cannot compute parity of two bits.

We then study the power of width 3 skew circuits. Recall that a CNF (DNF) is an AND (OR) of ORs (ANDs) of variables, i.e. in a CNF the AND gate is (possibly) non-skew. We implement a CNF by a width 3 skew circuit. Formally, we prove that any k-CNF or any k-DNF of size s has width 3 skew circuits of length $O(sk)$. Given that any Boolean function on n variables has a CNF of exponential (in n) size, this also proves that width 3 skew circuits are universal.

We consider the problem of proving lower bound for width 3 skew circuits. A natural candidate is a function which has no polynomial sized CNF or DNF. It is known that Parity is one such function. We prove that Parity requires width 3 skew circuits of exponential size. We observe that Parity and Approximate Majority have respectively, linear and polynomial size width 4 skew circuits. This separates width 3 skew circuits from width 4 skew circuits.

1.5. Organization of the paper

The rest of the paper is organized as follows: In Section 2, we introduce the branching programs, permutation branching programs and skew circuits, and present some obvious containments. In Section 3, we present our improved results of simulation of branching programs by skew circuits and vice versa. In Section 4, we explore the skew circuit classes of width 1 to 7 and present some upper and lower bounds. In Section 5, we give an exponential separation between skew circuits of width-3 and width-4 via the Parity function. We discuss some questions that remain unanswered by our work in Section 6.

2. Preliminaries

In this section we introduce some notations and preliminaries which we will use in the rest of the paper. We refer the reader to a standard text like [20] for definitions of standard circuit complexity classes.

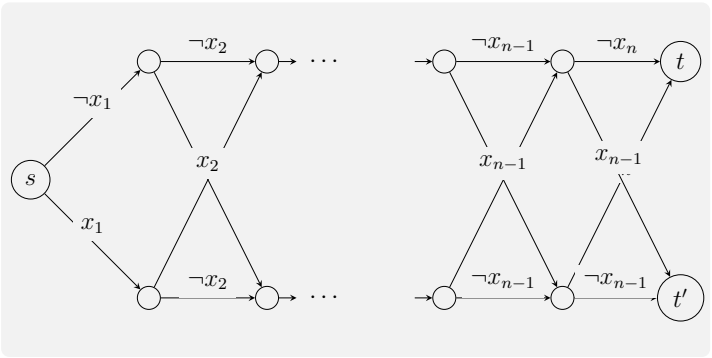

Figure 1: A width 2 BP computing Parity of n bits

A directed acyclic graph $G = (V, E)$ is called layered if the vertex set of the graph can be partitioned, $V = V_1 \cup \ldots \cup V_\ell$ in such a way that for each edge $e = (u, v)$ there exists $1 \leq i < \ell$ such that $u \in V_i$ and $v \in V_{i+1}$. Given a layered graph G the *length* of the graph is the number of layers in it and the *width* of the graph is the maximum over $i \in [\ell]$, $|V_i|$.

Definition 1. *(Branching Programs) A Deterministic Branching Program (BP) is a layered directed acyclic graph G with the following properties:*

- *There is a designated source vertex s in the first layer (of in-degree 0) and a sink vertex t (of out-degree 0) in the last layer.*
- *The edges are labeled by an element of $X \cup \{0, 1\}$, where X is the set of input variables to the branching program.*

The branching program naturally computes a boolean function $f(X)$, where $f(X) = 1$ if and only if there is path from s to t in which each edge is labeled by a true literal or a constant 1 on input X. The length *(*width*) of the BP is the length (respectively, width) of the underlying layered DAG.*

We will denote the class of languages accepted by width-w BP by BP^w. Figure 1 shows a width-2 branching program that computes the Parity function on n inputs.

Barrington [21, 7] defined a restricted notion of branching programs called the Permutation Branching Program (PBP):

Definition 2. *(Permutation Branching Programs as a graph) A width-w PBP is a layered width w BP in which the following conditions hold:*

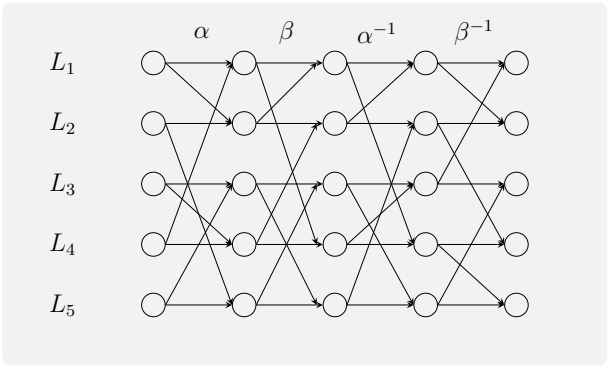

Figure 2: A Permutation branching program of width 5

- *There are designated source vertices $s_1, s_2, \ldots, s_w$ in the first layer, say layer 1 (of in-degree 0) and sink vertices $t_1, t_2, \ldots, t_w$ (of out-degree 0) in the last layer, layer ℓ.*
- *Each layer has exactly w vertices.*
- *In each layer $1 \leq i < \ell$, all the edges are labeled by a unique variable, say x_{j_i}.*
- *In each layer $1 \leq i \leq \ell$ and $b \in \{0, 1\}$, the edges activated when $x_{j_i} = b$ forms a permutation/matching, say $\theta_{i,b}$.*

The permutation branching program naturally computes a boolean function $f(X)$, where $f(X) = 1$ if and only if there is path from s_1 to t_1, s_2 to t_2, and so on till s_w to t_w, where in each path each edge is labeled by a true literal or a constant 1 on input X. We will refer to the class of languages accepted by polynomial sized width-w PBP by PBP^w.

The above definition of PBP can be rephrased as follows:

Definition 3. *(Permutation Branching Programs as a set of instructions) A width-w length-ℓ PBP is a program given by a set of ℓ instructions in which for any $1 \leq i \leq \ell$, the ith instruction is a three tuple $\langle j_i, \theta^i, \sigma^i \rangle$, where j_i is an index from $\{1, 2, \ldots, |X|\}$, θ^i, σ^i are permutations of $\{1, 2, \ldots, w\}$. The output of the instruction is θ^i if $x_{j_i} = 1$ and it is σ^i if $x_{j_i} = 0$. The output of the program on input x is the product of the output of each instruction of the program on x.*

We say that a permutation branching program computes a function f if there exists a fixed permutation $\pi \neq id$ such that for every x such that $f(x) = 1$ the program outputs π and for every x such that $f(x) = 0$ the program outputs id[3].

It is easy to see that the above two definitions of PBP are equivalent.

Definition 4. *(Skew Circuits) An AND gate is called* skew *if all but one of its children are input variables. A Boolean circuit in which all the AND gates are skew is called a* skew circuit*.*

We assume that the skew circuits are layered. The width of the circuit is the maximum number of gates in any layer. The layer may have AND, OR or input gates. Each type of gate contributes towards the width. We assume that the fan-in of the AND gates is bounded by 2 and there are no NOT gates (negations appear only for the input variables)[4]. We denote the class of languages decided by width-w skew circuits by SK^w. The following lemma summarizes some well known connections between BP^w, PBP^w, SK^w.

Lemma 5. *Let $w \in \mathbb{N}$. Then*

1. *For any w, $\mathsf{PBP}^w \subseteq \mathsf{BP}^w$.*
2. *For any $w \geq 5$, $\mathsf{BP}^w \subseteq \mathsf{PBP}^w$ [7].*
3. *For any w, BP^w is contained in SK^{w^2} (see for example [27]).*

Proof Sketch. The first part follows from the definitions of PBP^w and BP^w. The celebrated result of Barrington shows the converse for $w \geq 5$. The proof of 3 involves converting a layered DAG G into a circuit C_G such that on any input x there is a directed path from s to t in G if and only if C_G evaluates to 1 on x. This is done by converting every node of G into an OR gate and every edge of G into an AND gate. Let $e = (u, v)$ be an edge in G which connects vertex u to vertex v and has label x_e. Let OR_u and OR_v be the OR gates in C_G corresponding to u, v and let AND_e be the AND gate in C_G corresponding to e. In C_G, output of AND_e feeds into OR_v and AND_e receives OR_u and x_e as its inputs. It is easy to see that this construction

[3]This is often called the strong acceptance condition. Other notions of acceptance have been studied in the literature. See for example [28]

[4]This assumption is not without loss of generality. However, we will see that when a branching program is converted into a skew circuits, exactly this type of skew circuits arise.

ensures that on any input x there is a directed path from s to t in G if and only if C_G evaluated to 1 on x. Moreover, the circuit C_G thus constructed is a skew circuit. In a layered graph G if there are at most w nodes per layer, then there are at most w^2 edges between any two consecutive layers. Therefore, in C_G any layer can have at most w^2 gates.

□

Definition 6. *(Approximate Majority) Approximate Majority* $\mathsf{ApproxMaj}_{a,n} : \{0,1\}^n \to \{0,1\}$ *is the promise problem defined as:*

$$\mathsf{ApproxMaj}_{a,n}(x) := \begin{cases} 0, & \text{if } x \text{ has Hamming weight at most } a \\ 1, & \text{if } x \text{ has Hamming weight at least } n-a \end{cases}$$

3. Branching Programs and Skew Circuits

The Branching Program model lends itself to an easy skew circuit characterization and vice-versa. We sketch this correspondence in this section and improve upon the known conversion from skew circuits to branching programs.

3.1. Branching Programs to Skew Circuits

First we discuss the following folklore conversion of branching programs to skew circuits:

Lemma 7. *(Folklore) Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function computed by a width-w length-ℓ branching program with at most k edges between any two consecutive layers and with the additional property that each layer reads at most one variable or its negation. Then there is a skew circuit of width $max\{w+2, k\}$ and size $O((k+w)\ell)$ computing f.*

Proof. Given a branching program B of width w, we convert it to a skew circuit C as follows:

1. For every node g in B (except s), the circuit C has an OR gate $\vee_g$. The node corresponding to s in C is the gate that computes the constant function 1. The output gate of C is the node corresponding to t in B.
2. Suppose there are incoming edges $e_1, e_2, \ldots, e_k$ to the node g from gates $g_1, g_2, \ldots, g_k$ respectively. From our assumption they read the same input variable or its negation. For these wires we create AND gates

$\wedge_1, \wedge_2, \ldots, \wedge_k$ which feed into $\vee_g$ and each AND gate $\wedge_i$ receives two inputs: g_i and the variable (or its negation) labeling the edge e_i, respectively.

Every vertex in the BP gives rise to an OR gate in the skew circuit. And every wire in the BP gives rise to an AND gate in the skew circuit. As every wire in any layer reads the same variable or its negation, we need to add two vertices corresponding to this variable and its negation on the layer below the AND layer, i.e. in the OR layer just below it. Therefore, the width of the OR layer is at most $w + 2$, whereas the width of the AND layer is at most k. This immediately yields width and size bounds of $\max\{w + 2, k\}$ and $O((k + w)\ell)$ respectively. It is easy to see that for every $x \in \{0, 1\}^n$, $f(x) = B(x) = C(x)$. □

Figure 1 gives a width-2 branching program that computes the parity function on n inputs. A width-4 skew circuit computing the same function can be obtained by transforming the width-2 branching program via the method given above (See Figure 5).

3.2. Permutation Branching Programs to skew circuits

A permutation θ is called a *transposition* if either it is the identity permutation or there exists $i \neq j$ such that $\theta(i) = j$, $\theta(j) = i$ and for all $k \neq i \neq j$, $\theta(k) = k$. We call a transposition non-trivial if it is not the identity permutation, trivial otherwise.

Definition 8. *(Transposition Branching Programs, TBP) A width-w length-ℓ TBP is a program given by a set of ℓ instructions in which for any $1 \leq i \leq \ell$, the ith instruction is a three tuple $\langle j_i, \theta^i, \sigma^i \rangle$, where j_i is an index from $\{1, 2, \ldots, |X|\}$, θ_i, σ_i are transpositions of $\{1, 2, \ldots, w\}$. The output of the instruction is θ_i if $x_{j_i} = 1$ and it is σ_i if $x_{j_i} = 0$. The output of the program on input x is the product of the output of each instruction of the program on x.*

Lemma 9. *Given a width-w PBP of length ℓ there is an equivalent width-w TBP of length $O(w\ell)$.*

Proof. It is known (see for example [29]) that any permutation of $\{1, 2, \ldots, w\}$ can be written as a product of W transpositions of $\{1, 2, \ldots, w\}$, where $W = O(w)$. Let P be a width-w PBP of length ℓ. Consider the ith instruction in the program, say $\langle j_i, \theta_i, \sigma_i \rangle$. We know that we can write θ_i as a

product of W transpositions, i.e. $\theta_i = t_{i,1} \cdot t_{i,2} \dots t_{i,W}$, where for $1 \leq j \leq W$ $t_{i,j}$ is a transposition. Similarly, we have $\sigma_i = s_{i,1} \cdot s_{i,2} \dots s_{i,W}$, where $s_{i,j}$ is a transposition for $1 \leq j \leq W$.

To give a TBP equivalent to P, we replace every instruction $\langle j_i, \theta_i, \sigma_i \rangle$ in P by the following: $\langle j_i, t_{i,1}, id \rangle \cdot \langle j_i, t_{i,2}, id \rangle \dots \langle j_i, t_{i,W}, id \rangle \cdot \langle j_i, id, s_{i,1} \rangle \cdot \langle j_i, id, s_{i,2} \rangle \dots \langle j_i, id, s_{i,W} \rangle$. By a simple inductive argument we can prove that the the transposition branching program thus obtained is equivalent to P. As $W = O(w)$, the upper bound on the length of the resulting branching program follows. □

We defined TBPs as a set of instructions. Like in the case of PBPs, the definition of TBPs can be rephrased in terms of the underlying DAG. We observe the following about the DAG resulting from TBPs.

A width-w TBP is a layered width w PBP in which the following conditions hold:

- There are designated source vertices $s_1, s_2, \dots, s_w$ in the first layer, say layer 1 (of in-degree 0) and sink vertices $t_1, t_2, \dots, t_w$ (of out-degree 0) in the last layer, layer ℓ.
- Each layer has exactly w vertices.
- In each layer $1 \leq i < \ell$, all the edges are labeled by a unique variable, say x_{j_i}.
- In each layer $1 \leq i \leq \ell$, one of the following holds:
  - either the edges corresponding to $x_{j_i} = 1$ form a non-trivial transposition and the edges corresponding to $x_{j_i} = 0$ form the identity permutation
  - or the edges corresponding to $x_{j_i} = 0$ form a non-trivial transposition and the edges corresponding to $x_{j_i} = 1$ form the identity permutation

Remark 1. *As a result of the above properties of the TBP the total number of distinct edges between any two layers in a width-w TBP is at most $w+2$: there are w edges corresponding to the identity permutation, 2 edges corresponding to the transposition of two elements, and $w-2$ edges corresponding to the identity maps for all but the two transposed elements. The $w-2$ last edges overlap with the w edges corresponding to the identity permutation.*

Lemma 10. $\mathsf{PBP}^w \subseteq \mathsf{SK}^{w+2}$

Proof. Given a PBP^w for a language L of size s, by Lemma 9 we know that L also has a width-w TBP. By Remark 1 the underlying DAG for the TBP has at most $w+2$ edges between any two consecutive layers. Using Lemma 7 we get a skew circuit of width $w+2$ for L. Note that the size of such a circuit is $O(ws)$ □

Using Barrington's characterization of NC^1 and Lemma 10 we get the following: $\mathsf{NC}^1 = \mathsf{BP}^5 = \mathsf{PBP}^5 \subseteq \mathsf{SK}^7$

3.3. Skew Circuits to Branching Programs

In this section we start from a skew circuit of bounded width and convert it into a branching program of bounded width. Formally, we prove the following:

Theorem 11. *If C is a skew circuit of width w and length ℓ then there is a branching program P of width w and size $O(w\ell)$ computing the same function as C.*

Proof. Recall that in a skew circuit C, AND gates have fan-in 2 and at least one child is an input variable whereas OR gates have arbitrary fan-in and arbitrary predecessors.

Given a skew circuit C of width w and length ℓ_0 we will construct a branching program P of width w that will recognize the same language. Let $G_{i1}, \ldots, G_{iw}$ be the gates of C on layer i for $i = 1, \ldots, \ell_0$.

Let $X = \{\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_L}\}$ ($|X| = L$) be the set of layers on which there is at least one input gate. Without loss of generality we assume that in each $j \in [L]$ the gate $G_{\ell_{i_j} w}$ is an input gate. (There may be other input gates as well.)

We will construct a branching program of length $s = L + 2$ and width w. The nodes in the branching program in layer $\ell_{i_j} \in X$ will be called $N_{j0}, \ldots, N_{j(w-1)}$. The node N_{00} is the initial node and the node $N_{(s-1)(w-1)}$ will be the target node.

The nodes $N_{11}, \ldots, N_{(s-1)(w-1)}$ will by our construction compute the value of the nodes in a layer in X. More formally, for every input x, the gate $G_{\ell_j c}$ in layer ℓ_j of the circuit (and layer j in X) evaluated to 1 iff the node N_{jc} can be reached from the initial node. Since the gate G_{iw} in X is an input

gate we will not add corresponding gate in the branching program. We have completely specified the vertex set of the branching program P.

We now describe the edge set of P. We add an edge from $N_{(j-1)0}$ to N_{j0} labeled by 1 for every $1 \leq j \leq s-1$. This ensures that all nodes N_{j0} are always reachable from the initial node.

Suppose that the layer ℓ_j and $\ell_j + 1$ are both in X, i.e. $\ell_{j+1} = \ell_j + 1$, then the edges between the nodes in the layer ℓ_j and ℓ_{j+1} in the branching program are easy to state. A node N_{j+1c} is connected to N_{jd} if there is an edge between the corresponding gates $G_{\ell_{j+1}c}$ and $G_{\ell_j d}$. Also the edge in the branching program is labeled by 1 if the gate $G_{\ell_j d}$ is an OR gate, and labeled by the variable x_i (or its negation $\neg x_i$) if $G_{\ell_j d}$ is an AND gate querying x_i, resp. $\neg x_i$. If an OR gate in ℓ_j is connected to an input gate, we generate an edge to N_{j0} labeled by the literal queried by the input gate.

Now assume that the layer ℓ_j is in X and ℓ_{j+1} is the next layer in X and $\ell_{j+1} > \ell_j + 1$. Then in the skew circuit, no input gates occur strictly between the layers ℓ_j and ℓ_{j+1}. This implies that there are no AND gates in the layers $\ell_j + 2, \ldots, \ell_{j+1}$. Hence the functions computed by the gates in layer ℓ_{j+1} are ORs of some gates in layer $\ell_j + 1$. In layer gates in layer $\ell_j + 1$ are ORs of either ANDs of gates in layer $\ell_j + 1$ and an input variable or ORs of directly gates in layer $\ell_j + 1$. Therefore, we add the following edges in the branching program: a node $N_{(j+1)c}$ is connected to N_{jd} if the OR function computed by $G_{\ell_{j+1}c}$ has $G_{\ell_j d}$ as one of the inputs. This edge in the branching program is labeled by 1 if this was a direct OR, it is labeled by the variable x_i (or its negation $\neg x_i$) it it was an 'or' of an 'and' querying x_i (resp. $\neg x_i$).

It is easy to verify by induction on the layers that N_{jc} is reachable from the initial gate if the corresponding gate evaluates true. Finally we add an edge from the node corresponding to the output gate to $N_{(s-1)(w-1)}$. □

Putting together Lemma 10 and Theorem 11 we get the following corollary:

Corollary 12. $\mathsf{NC}^1 = \mathsf{BP}^5 = \mathsf{PBP}^5 = \mathsf{SK}^7$

4. Width ≤ 7 skew circuits

In this section we study the structure of the languages in NC^1 by investigating properties of skew circuits of width 7 or less. By definition $\mathsf{SK}^i \subseteq \mathsf{SK}^{i+1}$ for $1 \leq i \leq 6$.

We start by proving that width 2 skew circuits are not universal.

Lemma 13. *A width* 2 *skew circuit of arbitrary size cannot compute Parity of* 2 *bits.*

Proof. Let $f = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$. To show that $f \notin \mathsf{SK}^2$, firstly notice that fixing the values of both the variables is necessary and sufficient to make f a constant function. Let C be a SK^2 circuit of minimal length computing f. The output gate of C cannot have a constant as its input: suppose the output is an AND(OR) gate and receives 0 (resp. 1) as input then the circuit computes a constant function. On the other hand, if the output is an AND(OR) gate and receives 1 (resp. 0) as input then such an output gate is redundant, contradicting the minimality of the circuit.

Now, the output cannot be an AND gate because being skew, it can be fixed by fixing one of its inputs, which contradicts the fact that C computes f. Therefore the output gate of C, say g_0, is an OR gate. Let g_1, g_2 be its two inputs. The following cases arise:

a) Both g_1, g_2 cannot be OR since we can collapse such a layer and still compute the same function, contradicting minimality.
b) Both g_1, g_2 cannot be AND: suppose they are both AND gates, the next layer must have at least one input variable, say x_i. Let the other gate on that layer be h_1. If both query the variable, then by setting that variable to 0, we will make the output zero, which will contradict the assumption that C computes f. Suppose one of them, say g_1, does not query x_1 then the output of the circuit is $g_0 = (h_1 \wedge x_i) \vee h_1 = h_1$. This contradicts the minimality of the circuit.
c) Even one of g_1, g_2 cannot be a variable, else by setting that variable we can fix the output of the circuit.
d) Due to cases (a), (b) and (c) we are only left with a case in where one of the inputs to g_0 is an AND and the other is OR (say g_1 is AND and g_2 is OR). In the layer just below this layer, there will have to be an input gate in a minimal circuit and this will have to be queried by both g_1, g_2. This variable can now be fixed (to make $g_2 = 1$ and therefore $g_0 = 1$) so that the output of the circuit is fixed.

Therefore, no width 2 skew circuit computes f. This proves that width 2 circuits are not universal. □

Recall that a *k-DNF of size s on n variables* is an OR of s terms, where each term is an AND of at most k literals from $\{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\}$.

Similarly, *k-CNF of size s on n variables* is an AND of s clauses, where each clause is an OR of at most k literals from $\{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\}$.

Lemma 14. *Let f be a k-DNF of size s on n variables. Then f has a width-3 skew circuit of length $O(sk)$.*

Proof. A term is an AND of at most k literals. It is easy to see that any such term can be computed by a skew circuit of width 2 and length $O(k)$. Suppose $C_1, C_2, \ldots, C_s$ are width-2 skew circuits of length $\ell_1, \ell_2, \ldots, \ell_s$, respectively computing the s terms in the k-DNF of size s. We need to compute an OR of these. This can be done using the one extra width: stagger circuits $C_1, C_2, \ldots, C_s$ one after the other. This requires width 2 and length $L := k \cdot \sum_{i=1}^{s} \ell_i = O(sk)$. Now add one OR gate per layer alongside these C_is. Let us call them $g_1, g_2, \ldots, g_L$. Feed output of g_i to g_{i+1} for all $1 \leq i \leq L$ and feed output of C_i to g_{ℓ_i+1}. It is easy to see that g_{L+1} computes f. (See Figure 3.) □

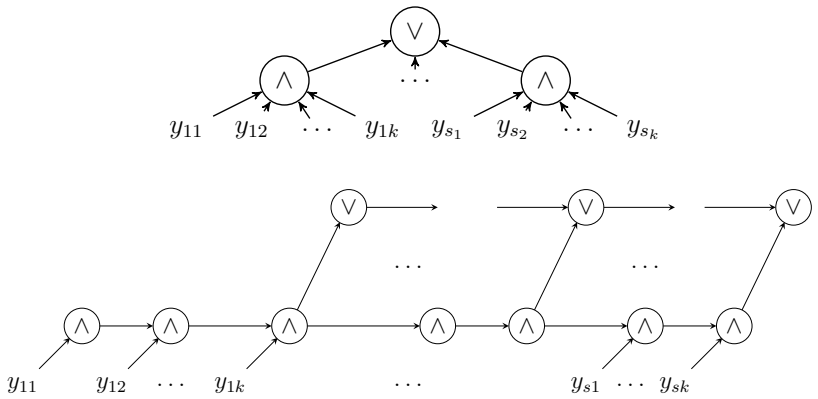

Figure 3: Width 3 skew circuits for DNFs

As any Boolean function on n variables has an n-DNF of size at most 2^n, we get the following corollary.

Corollary 15. *Let $f : \{0,1\}^n \to \{0,1\}$. Then f can be computed by width-3 skew circuit of length $O(n2^n)$, i.e. width-3 skew circuits are universal.*

Lemma 16. *Let f be a k-CNF of size s on n variables. Then f has a width-3 skew circuit of length $O(sk)$.*

Proof. Note that in a CNF, the top AND gate gets clauses as inputs. That is, the AND gate is not skew. However, it is still possible to get a skew circuit for CNFs. We prove this by induction on the number of clauses, i.e. s. The base case is $s = 1$. This is just an OR of literals, which is computable by width-2 skew circuit of length $O(k)$. Let $f_i(x_1, \ldots, x_n) = C_1 \wedge \ldots \wedge C_i$ be computable by width-3 skew circuit of length $O(ik)$. Now $f_{i+1} = f_i \wedge C_{i+1}$ (where $C_{i+1} = x_{j_1} \vee x_{j_2} \ldots \vee x_{j_k}$) is computed as follows: $f_{i+1} = (\ldots((f_i \wedge x_{j_1}) \vee (f_i \wedge x_{j_2}) \ldots (f_i \wedge x_{j_k}))\ldots)$. Note that we need width 1 each for the f_i, the AND gate and the input variable. (Even though we require width 3 to compute f_i, after the computation, it just requires width 1 to carry around the value of the function to the next stage). (See Figure 4.) □

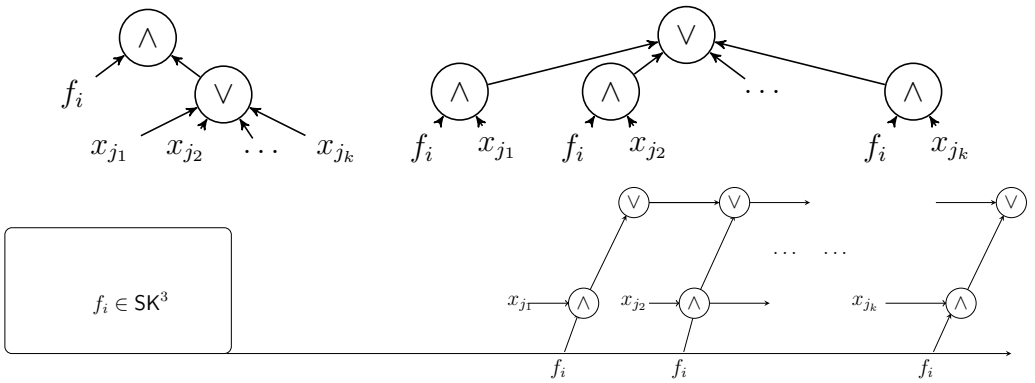

Figure 4: Width 3 skew circuits for CNFs

Definition 17. *(Approximate Majority) Approximate Majority* $\mathsf{ApproxMaj}_{a,n} : \{0,1\}^n \to \{0,1\}$ *is the promise problem defined as:*

$$\mathsf{ApproxMaj}_{a,n}(x) := \begin{cases} 0, & \text{if } x \text{ has Hamming weight at most } a \\ 1, & \text{if } x \text{ has Hamming weight at least } n - a \end{cases}$$

By a result of Viola [30], it is known that Approximate Majority is computable by P-uniform depth-3 circuits of polynomial size with alternating AND/OR layers with the output gate being an OR gate. This along with Lemma 16 above yields:

Corollary 18. $\mathsf{ApproxMaj}_{a,n}$ *has skew circuits of width* 4 *and polynomial length.*

Razborov[12] and Smolensky[13] show that Parity does not have constant depth circuit of polynomial size. It also implies that Parity does not have polynomial size CNFs or DNFs . However, we now show that Parity has skew circuits of width 4 and polynomial size.

Lemma 19. *Parity on n variables has a skew circuit of width 4 and length $O(n)$.*

Proof. This is an easy observation which comes from the fact that Parity has a branching program of width 2 and length $O(n)$. This fact along with part 3 of Lemma 5 proves the result. (Figure 5 shows the width 4 skew circuit for Parity.) □

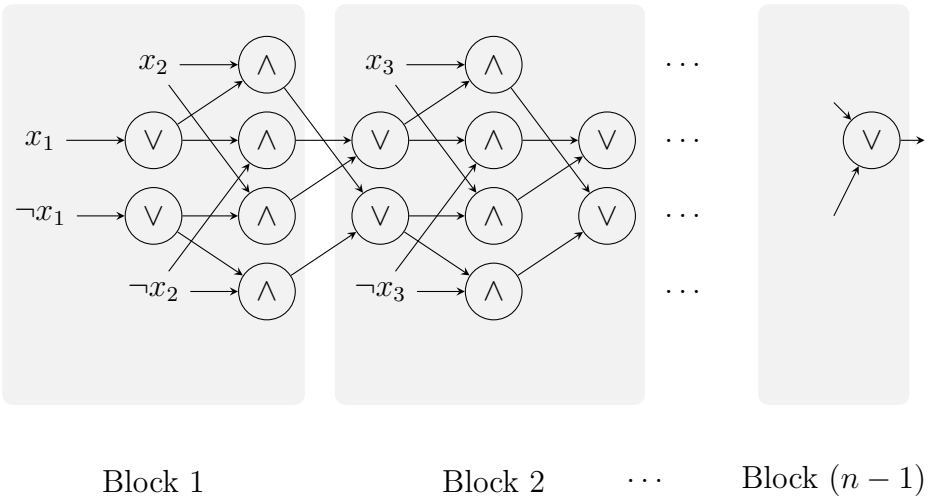

Figure 5: Width-4 skew circuit for Parity

5. Parity and SK^3

In this section we prove that Parity does not have polynomial length width-3 skew circuits. As Parity has width 4 skew circuits of linear length (Lemma 19), this separates SK^3 from SK^4.

Theorem 20. $\mathsf{SK}^3 \subsetneq \mathsf{SK}^4$.

In order to prove this we first show that any width three skew circuit computing Parity can be converted into a normal form. We then show that any polynomial sized circuit of that normal form cannot compute Parity.

Lemma 21. *Let C be a Boolean width 3 skew circuit with size $s(n)$ computing* $\textsc{Parity}_n$. *The circuit C can be converted into another circuit D such that D computes Parity of at least $n-2$ bits and has the following structure:*

1. *The top gate of D is an OR of at most $3s(n)^2$ disjoint skew circuits, say $C_1, C_2, \ldots, C_m$, where $m \leq 3s(n)^2$.*
2. *The sum of sizes of all C_is is at most $O(s(n)^3)$*
3. *At most two of these circuits have width 3 and all the other have width at most 2.*

Lemma 22. *Let D be a circuit satisfying properties 1,2,3 from Lemma 21 and computing* $\textsc{Parity}_n$. *Then there exists a constant c such that the size of D is at least $2^n/n^c$.*

Using Lemma 21 and Lemma 22 it is easy to see that the lower bound for Parity follows.

5.1. Proof of Lemma 21

Let C be a width three circuit computing Parity of n bits of size $s(n)$. The top gate of C cannot be AND. This is because, by fixing the input wires of the AND gate, we could fix the output of the circuit, however, Parity of n bits cannot be fixed by fixing $< n$ bits. Therefore, we can assume that the top gate is an OR gate.

Proposition 23. *Let C be any width three skew circuit computing Parity of n bits. Let k be the highest layer in C consisting of only AND gates, say g_k^1, g_k^2, g_k^3. We can convert this into another width 3 skew circuit D computing Parity of at least $n-2$ bits such that no layer of D contains three AND gates.*

Proof. Let k be the highest layer (closest to the output gate) with all three AND gates, say g_k^1, g_k^2, g_k^3. Note that, due to skewness of the circuit, the layer $k+1$ cannot have any AND gates. If any one of the gates at layer k has fan-in 1, then we can replace that gate by an OR gate. Therefore, we will assume that each gate has fan-in 2. Let gates at layer below, i.e. $k-1$, be $g_{k-1}^1, g_{k-1}^2, g_{k-1}^3$. As we are dealing with skew circuits, at least one of $g_{k-1}^1, g_{k-1}^2, g_{k-1}^3$ is an input gate. Suppose there is only one input gate, then

all gates at layer k must read this input bit. And therefore, the three gates at layer k can be fixed by fixing this input bit. If there are two input gates, then either both read the same literal (a variable and its negation) or they read two distinct input bits. In the latter case, by fixing the two distinct bits, all the three gates can be fixed. Finally, if the two variables being read are x and $\neg x$ then fixing x may not fix all three gates. For example, suppose $g^1_{k-1} = x, g^2_{k-1} = \neg x, g^3_{k-1} = g$, and $g^1_k = (g \text{ AND } x)$ and $g^2_k = (g \text{ AND } \neg x)$. Then fixing x to any value will not fix both g^1_k and g^2_k. However, in this case, note that gates at layer k feed into OR gates. Therefore, in this case, such an AND layer is redundant. Therefore, any such layer consisting of only AND gates can be completely removed from the circuit by fixing at most 2 bits. □

Therefore, we will now assume that we have a circuit D which computes parity of at least $n-2$ bits in which no layer consists of only AND gates.

Let $G = (V, E)$ denote the DAG underlying the circuit D. Let $X \subseteq V$ denote the subset of gates which have a path to the top gate via only OR gates. Note that all the vertices in this set are themselves OR gates. We refer to the set of vertices X as ORSET.

Let $X_{in} \subseteq V \setminus X$ denote the set of vertices which have an edge incident from it to some vertex in X. Similarly, let $X_{out} \subseteq V \setminus X$ denote the set of vertices which have an edge incident to them from some vertex in X.

(a) Disconnect all edges incident to the set X_{out} from X. Let the new dangling wires be labeled with the constant 0 input.

(b) If after step (a) any AND gates receives constant input 0 then delete the gate and if any OR gate receives constant input 0 then delete this input of the OR gate.

(c) In the graph obtained after steps (a) and (b), consider $V \setminus X$. This disconnects the DAG G and gives rise to some connected components, say $X_1, X_2, \ldots, X_\ell$. For each $i \in [\ell]$ and edge (u, v) such that $u \in X_i$ and $v \in X$ let $C_{i,u}$ be the subgraph (DAG) of X_i with u as its sink.

Proposition 24. *The step (a) above does not change the function computed by the circuit.*

Proof. Let u be a gate in X which feeds into some gate in X_{out} say v. Step (a) above disconnects u from v and feeds value 0 to v. Suppose g_u, the OR

gate corresponding to u evaluates to 1, then the circuit will evaluate to 1 irrespective of all other gates. On the other hand, if it evaluates to 0, then that is the value that we have fed into v and therefore, the modification in Step (a) does not change the output of the circuit in either case. □

Note that $V \setminus X$ is partitioned by X_is. Therefore, $\ell \leq s(n)$. Also, $\cup_{i=1}^{m} X_i \subseteq V \setminus X$. Therefore, $\sum_{i=1}^{m} |X_i| \leq s(n)$. The number of edges in G is at most $3s(n)$. Therefore, the total number of edges between any X_i and X is at most $3s(n)$. Therefore, the number of circuits $C_{u,i}$ is at most $3s(n)^2$ and each such circuit is of size at most $s(n)$. This proves parts $1, 2$ of Lemma 21.

We will now prove part 3 of Lemma 21. The top gate of D is the same as the top gate of C, and it is an OR gate. Therefore, this gate is in the ORSET. Now as long as there are OR gates on the layers, we have at least one gate per layer in the ORSET. Finally, if there is a layer with no OR gates, then this layer must have at least one input variable (as D does not have any layer with three AND gates). The other two gates at this layer be g, h. Let X_g, X_h be two DAGs rooted at g and h, respectively, and C_g and C_h be the two corresponding circuits. These are possibly width three circuits. However, all other connected components of $V \setminus X$ are of width at most 2 due to step (b) above. This gives Part 3 of Lemma 21.

5.2. Proof of Lemma 22

Let D be the circuit given by Lemma 21. Let n_0 denote the number of unfixed variables. Let $C_1, C_2, \ldots, C_m$ be circuits given by Lemma 21. We know that at most two circuits among these have width 3. Let us assume without loss of generality that the two circuits are C_1, C_2. The output gates of these circuits are AND gates, say G_1 and G_2, respectively. These being skew circuits, all but one of the inputs of the AND gates are input gates. We will first prove the following proposition.

Proposition 25. *By fixing at most two variables both G_1 and G_2 can be set to zero.*

Proof. Let $G_1, G_2, \ldots, G_m$ be top gates of the circuits $C_1, C_2, \ldots, C_m$. Note that G_is are AND gates because if they were OR gates they would have been in the ORSET. Say they appear on layers $k_1, k_2, \ldots, k_m$ in the original width 3 circuit C. If both C_1, C_2 are width 3 circuits, then it is easy to see that $k_1 \leq k_i$ for $3 \leq i \leq m$ and $k_2 \leq k_i$ for $3 \leq i \leq m$. Let us assume without

loss of generality $k_1 \leq k_2$, i.e the gate G_1 appears on a lower layer (closer to Layer 1) than G_2.

Let x_i, Y_1, Y_2 be the gates on the layer $k_1 - 1$. As G_1 is an AND gate, it must query one variable. Let that be x_i. Let the other input to G_1 be Y_1 without loss of generality.

Now note that Y_2 must be connected to the ORSET via G_2. If it is not connected via G_2 then it is easy to see that C_2 cannot have width 3. (since $k_2 > k_1$ and ORSET extends all the way upto $k_1 + 1$, therefore, the circuit C_2, which starts at k_2 and is disjoint from the ORSET and disjoint from the path which connected Y_2 to the ORSET, can have width at most 2.)

Our proof for the proposition is a case analysis of the various settings for x_i, Y_1, and Y_2.

(a) Suppose input gate of G_2 reads a variable $y \neq \neg x_i$, then set $x_i = 0, y = 0$ which will eliminate both G_1, G_2 by setting at most two variables as desired.

(b) Suppose $y = \neg x_i$. Here, by minimality of the circuit we can assume that the output of the entire circuit can be written as $\vee_{i=3}^{O(m)} C_i \vee (Y_1 \wedge x_i) \vee (Y_2 \wedge \neg x_i)$. To handle this we need to look at the next layer below $k_1 - 1$ that reads an input variable. Say the variable read is x_j and the layer number is k_0. Let the other two gates on this layer be Z_1, Z_2. Here again, by observing that there cannot be any redundant gates in the minimal circuit (and using distributivity of AND/ORs) it is easy to see that $x_j \neq x_i$. If either of Y_1, Y_2 is an AND gate, then $k_0 = k_1 - 1$ else both are OR gates.

Therefore, Y_1, Y_2 are ORs over $\{(x_j \wedge Z_1), Z_1, Z_2\}$. (The analysis for the case of ORs over $\{(x_j \wedge Z_2), Z_1, Z_2\}$ is symmetric. The case of ORs over simply $\{x_j, Z_1, Z_2\}$ cannot arise as otherwise Y_1 or Y_2 will be connected to the ORSET directly, but this is not possible as we have that both are connected to the ORSET via ANDs.)

(i) If both Y_1 and Y_2 do not query $x_j \wedge Z_1$, then the AND is redundant, which is not possible in a minimal circuit.

(ii) If Y_2 (Y_1) queries only $x_j \wedge Z_1$. Then by setting $x_i = 0, x_j = 0$ ($\neg x_i = 0, x_j = 0$, respectively) we can set both G_1 and G_2 to zero.

(iii) If Y_2 queries $(x_j \wedge Z_1)$ and Z_1, but not Z_2. Then again the AND is redundant. (The case that Y_1 queries $(x_j \wedge Z_1)$ and Z_1, but not Z_2 is similar.)

(iv) If Y_2 queries $(x_j \wedge Z_1)$ and Z_2, but not Z_1 and Y_1 queries Z_2 (Y_1 may query Z_1 and $x_j \wedge Z_1$ as well) then $G_1 \vee G_2 = (((x_j \wedge Z_1) \vee Z_2) \wedge x_i) \vee (Z_2 \wedge x_i) = ((x_j \wedge Z_1) \wedge x_i) \vee (Z_2 \wedge x_i) \vee (Z \wedge \neg x_i) = ((x_j \wedge Z_1) \wedge x_i) \vee Z_2$, i.e. Z_2 directly feeds into the ORSET, but this contradicts minimality. (The case that Y_1 queries $(x_j \wedge Z_1)$ and Z_2, but not Z_1 and Y_2 queries Z_2 is similar).

(v) If Y_2 queries $(x_j \wedge Z_1)$ and Z_2, but not Z_1 and Y_1 queries Z_1 but nothing else then Y_1 is redundant.

This exhausts all the cases and finally by setting at most 2 variables we have managed to eliminate both G_1, G_2. This finishes the proof of the proposition.

□

Let N denote the number of variables which were not set. Here, $N \geq n_0 - 2$. The new circuit, say D', is now an OR of $C_3, C_4, \ldots, C_m$ and by our assumption, it computes Parity of N variables. We will show that OR of polynomially many polynomial size width-2 skew circuits cannot compute Parity.

Let us fix some notation. Let $L_\oplus$ denote the 1 set of parity, i.e. $L_\oplus = \{x \in \{0,1\}^N \mid Parity(x) = 1\}$. We know that $|L_\oplus| = 2^{N-1}$. For any Boolean circuit C, let L_C denote $\{x \in \{0,1\}^N \mid C(x) = 1\}$. Note that as D' above is an OR of $C_3, C_4 \ldots, C_m$, we have $L_{D'} = \cup_{i=3}^m L_{C_i}$.

Definition 26. *We say that a Boolean circuit C* α-approximates *a function $f : \{0,1\}^n \to \{0,1\}$ if the following conditions hold:*

- $\forall x \in \{0,1\}^n$, *if* $f(x) = 0$ *then* $C(x) = 0$, *i.e. C has no false positives.*
- *The ratio of* $|\{x \mid C(x) = 1\}|$ *to* $|\{x \mid f(x) = 1\}|$ *is at least* α

For the sake of contradiction we have assumed that D' computes parity of N bits. Assuming this and from the fact that $L_{D'} = \cup_{i=3}^m L_{C_i}$, we get that there exists an $i \in \{3, 4, \ldots, m\}$ such that C_i $1/m$-approximates parity of N bits. We will now prove that no such C_i exists, which will give us the contradiction. Formally, we prove the following:

Claim 27. *Let D' and $C_3, C_4, \ldots, C_m$ be defined as above. There does not exists $i \in \{3, 4, \ldots, m\}$ such that C_i $1/m$-approximates Parity of N bits.*

Proof. Suppose there exists a C_i which $1/m$-approximates parity of N bits. Recall that C_i is a width 2 skew circuit. Let the last layer be L and the first layer be 1. Let $\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_t}$ be the layers in which there is one input gate, with ℓ_{i_1} being closest to layer 1 and ℓ_{i_t} being closest to layer L. (Note that, we can assume without loss of generality that layer 1 is the only layer which has two input gates.) Let the variables queried by these gates be $x_{i_1}, x_{i_2}, \ldots, x_{i_t}$, respectively. Let $h_{i_{t+1}}$ denote the output gate in layer L. Similarly, let h_{i_1} be the gate in layer ℓ_{i_1} (other than the input gate), h_{i_2} be the gate in layer ℓ_{i_2} (other than the input gate) and so on till h_{i_t} be the gate in layer ℓ_{i_t} (other than the input gate).

As there are no NOT gates in the circuit, $h_{i_{j+1}}$ is a monotone function of x_{i_j}, h_{i_j} for every $1 \leq j \leq t$. There is a unique value of x_{i_j}, say $b_{i_j} \in \{0,1\}$, such that by setting $x_{i_j} = b_{i_j}$, $h_{i_{j+1}}$ becomes a non-trivial function of h_{i_j}. (This is because, there are at most 6 different monotone functions on two bits, two of which cannot occur in a minimal circuit. And the other four (AND, OR, NAND, NOR) have this property.)

Note that, the setting of $x_{i_t} = b_{i_t}$ will not fix the value of h_{i_t}. Suppose h_{i_t} gets fixed due to this setting. In that case, value of $h_{i_{t+1}}$ will also get fixed. Suppose the value of $h_{i_{t+1}}$ becomes 1, then for all settings of $x \neq x_{i_t}$, $h_{i_{t+1}}$ will continue to have value 1. But we have assumed that C_i has no false positives. Therefore, this is not possible. On the other hand, if the value of $h_{i_{t+1}}$ gets fixed to 0, then for all settings of variables $x \neq x_{i_t}$ the circuit will output 0. That is, for 2^{N-1} different inputs the circuit will output 0. However, we have assumed that the circuit outputs 1 for at least $2^{N-1}/m$ many inputs.

Assuming $x_{i_t} = b_{i_t}$ and $x_{i_t} \neq x_{i_{t-1}}$, we will repeat this argument for $x_{i_{t-1}}$. Let $x_{i_{t-1}} = b_{i_{t-1}}$ be the setting of $x_{i_{t-1}}$ which makes h_{i_t} a function of $h_{i_{t-1}}$. Suppose this setting of $x_{i_{t-1}}$ fixes h_{i_t} then that will in turn fix $h_{i_{t+1}}$. As before, to avoid false positives, the value of $h_{i_{t+1}}$ cannot be fixed to 1. And to ensure that the circuit evaluated to 1 on at least $2^{N-1}/m$ inputs, it cannot be fixed to 0.

In this way, we can repeat the argument for k distinct variables as long as $k < (N-1) - \lceil \log m \rceil$. Let k_0 be such that $k_0 = \omega(\log m)$ and $k_0 < (N-1) - \lceil \log m \rceil$. We fix k_0 distinct variables as above. But now note that any other setting of these k_0 variables fixes the value of $h_{i_{t+1}}$ to 0. Therefore, the circuit can be 1 on at most $O(2^{N-k_0})$ inputs. But this contradicts our assumption that $h_{i_{t+1}}$ evaluated to 1 on at least $2^N/m$ inputs from $L_\oplus$.

□

6. Discussion

The above study provides a wide range of interesting questions, answers to which may improve our understanding of functions in NC^1. Namely, the questions regarding lower bounds for width k skew circuits for $4 \leq k \leq 6$. Some of these questions could be more tractable than the daunting question of proving lower bounds for NC^1 circuits. Plenty of gaps remain in our understanding of bounded width branching programs and skew circuits:

- At what width are skew circuits capable of computing the Majority function? It is clear from Barrington's work and Lemma 10 that width 7 suffices. Can we do better?
- Can we prove an exponential lower bound for width-3 or width-4 skew circuits computing the Majority function? Note that even though Parity reduces to Majority and we have shown that Parity requires exponential size width 3 skew circuits, we do not know of a reduction between Parity to Majority that can be implemented by width 3 skew circuits. Towards this end, the first step might be to obtain a normal form for any width 3 circuit computing Majority, like the one we have for Parity in Lemma 21.
- Can we show the hierarchy between width 4 to width 7 skew circuits is strict under a plausible Complexity-theoretic assumption (say for e.g. $\mathsf{NC}^1 \not\subseteq \mathsf{TC}^0 \not\subseteq \mathsf{ACC}^0$)?

References

[1] A. Borodin, On relating time and space to size and depth, SIAM journal on computing 6 (4) (1977) 733–744.

[2] P. W. Beame, S. A. Cook, H. J. Hoover, Log depth circuits for division and related problems, SIAM Journal on Computing 15 (1986) 994–1003.

[3] W. Hesse, E. Allender, D. Barrington, Uniform constant-depth threshold circuits for division and iterated multiplication, Journal of Computer and System Sciences 65 (2002) 695–716.

[4] R. J. Lipton, Y. Zalcstein, Word problems solvable in logspace, Journal of the ACM (JACM) 24 (3) (1977) 522–526.

[5] P. M. Spira, On time-hardware complexity tradeoffs for boolean functions, in: Proceedings of the 4th Hawaii Symposium on System Sciences, 1971, pp. 525–527.

[6] S. R. Buss, The boolean formula value problem is in alogtime, in: Proceedings of the nineteenth annual ACM symposium on Theory of computing, ACM, 1987, pp. 123–131.

[7] D. Barrington, Bounded-width polynomial-size branching programs can recognize exactly those languages in NC^1, J. Comput. Syst. Sci. 38 (1989) 150–164.

[8] D. A. M. Barrington, D. Thérien, Finite monoids and the fine structure of nc1, J. ACM 35 (4) (1988) 941–952. doi:10.1145/48014.63138. URL http://doi.acm.org/10.1145/48014.63138

[9] J. Hastad, Almost optimal lower bounds for small depth circuits, in: Proceedings of the eighteenth annual ACM symposium on Theory of computing, ACM, 1986, pp. 6–20.

[10] M. Furst, J. B. Saxe, M. Sipser, Parity, circuits, and the polynomial-time hierarchy, Mathematical Systems Theory 17 (1) (1984) 13–27.

[11] A. C.-C. Yao, Separating the polynomial-time hierarchy by oracles, in: — 26th Annual Symposium on Foundations of Computer Science, IEEE, 1985, pp. 1–10.

[12] A. A. Razborov, Lower bounds on the size of bounded depth circuits over a complete basis with logical addition, Mathematical Notes 41 (4) (1987) 333–338.

[13] R. Smolensky, Algebraic methods in the theory of lower bounds for boolean circuit complexity, in: Proceedings of the nineteenth annual ACM symposium on Theory of computing, ACM, 1987, pp. 77–82.

[14] R. Beigel, The polynomial method in circuit complexity., in: Structure in Complexity Theory Conference, 1993, pp. 82–95.

[15] R. Williams, Nonuniform acc circuit lower bounds, Journal of the ACM (JACM) 61 (1) (2014) 2.

[16] C.-Y. Lee, Representation of switching circuits by binary-decision programs, Bell System Technical Journal 38 (4) (1959) 985–999.

[17] W. J. Masek, A fast algorithm for the string editing problem and decision graph complexity, Ph.D. thesis, Massachusetts Institute of Technology (1976).

[18] A. Borodin, D. Dolev, F. E. Fich, W. Paul, Bounds for width two branching programs, SIAM Journal on Computing 15 (2) (1986) 549–560.

[19] A. A. Razborov, Lower bounds for deterministic and nondeterministic branching programs, in: Fundamentals of Computation Theory, Springer, 1991, pp. 47–60.

[20] S. Jukna, Boolean function complexity: advances and frontiers, Vol. 27, Springer Science & Business Media, 2012.

[21] D. A. Barrington, Width-3 permutation branching programs, technical Memo MIT/LCS/TM-293, Massachusetts Institute of Technology, Laboratory for Computer Science (1985).

[22] H. Venkateswaran, Circuit definitions of nondeterministic complexity classes, SIAM Journal on Computing 21 (4) (1992) 655–670.

[23] S. Toda, Classes of arithmetic circuits capturing the complexity of computing the determinant, IEICE Transactions on Information and Systems E75-D (1992) 116–124.

[24] G. Malod, N. Portier, Characterizing valiant's algebraic complexity classes, Journal of complexity 24 (1) (2008) 16–38.

[25] E. Allender, J. Jiao, M. Mahajan, V. Vinay, Non-commutative arithmetic circuits: depth reduction and size lower bounds, Theoretical Computer Science 209 (1) (1998) 47–86.

[26] N. Limaye, G. Malod, S. Srinivasan, Lower bounds for non-commutative skew circuits, Electronic Colloquium on Computational Complexity, Report 2015/022, http://www.eccc.hpi-web.de/report/2015/022/ (2015).

[27] B. Raghavendra Rao, A study of width bounded arithmetic circuits and the complexity of matroid isomorphism, [HBNI TH 17].

[28] A. Brodsky, An impossibility gap between width-4 and width-5 permutation branching programs, Information Processing Letters 94 (4) (2005) 159–164. doi:10.1016/j.ipl.2005.01.012.
URL http://dx.doi.org/10.1016/j.ipl.2005.01.012

[29] I. N. Herstein, Topics in algebra, John Wiley & Sons, 2006.

[30] E. Viola, On approximate majority and probabilistic time, Computational Complexity 18 (3) (2009) 337–375.