CIRCUITS ON CYLINDERS

Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and V Vinay

Abstract. We consider the computational power of constant width polynomial size cylindrical circuits and nondeterministic branching programs. We show that every function computed by a $\mathbf{\Pi_2} \circ \mathbf{MOD} \circ \mathbf{AC}^0$ circuit can also be computed by a constant width polynomial size cylindrical nondeterministic branching program (or cylindrical circuit) and that every function computed by a constant width polynomial size cylindrical circuit belongs to $\mathbf{ACC}^0$.
Keywords. Circuit complexity; branching programs.
Subject classification. 68Q05, 68Q70.

1. Introduction

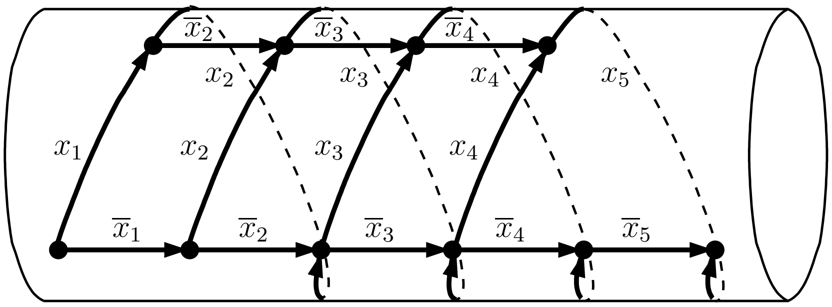

Figure 1.1: A cylindrical branching program of width 2 computing PARITY.

In this paper we consider the computational power of constant width, polynomial size *cylindrical* branching programs and circuits.

It is well known that there is a rough similarity between the computational power of *width* restricted circuits and *depth* restricted circuits, but that this similarity is not a complete equivalence. For instance, the class of functions computed by a family of circuits of *quasi*-polynomial size and polylogarithmic depth is equal to the class of functions computed by a family of circuits of quasi-polynomial size and polylogarithmic width. On the other hand, the class of functions computed by a family of circuits of *polynomial* size and polylogarithmic width (non-uniform **SC**) is, in general, conjectured to be different from the class of functions computed by a family of circuits of polynomial size and polylogarithmic depth (non-uniform **NC**). For the case of *constant* depth and width, there is a provable difference in computational power; the class of functions computable by constant depth circuits of polynomial size, i.e., $\mathbf{AC}^0$, is a proper subset of the functions computable by constant width circuits (or branching programs) of polynomial size, the latter being, by Barrington's Theorem (Barrington 1989), the bigger class $\mathbf{NC}^1$. On the other hand, Vinay (1996) and Barrington *et al.* (1998, 1999) showed that by putting a *geometric* restriction on the computation, the difference disappears: The class of functions computable by *upward planar*, constant width, polynomial size circuits (or nondeterministic branching programs) is exactly $\mathbf{AC}^0$. Thus, both $\mathbf{AC}^0$ and $\mathbf{NC}^1$ can be captured by a constant width as well as by a constant depth circuit model. It is then natural to ask if one can similarly capture classes between $\mathbf{AC}^0$ and $\mathbf{NC}^1$ defined by various constant depth circuit models, such as $\mathbf{ACC}^0$ and $\mathbf{TC}^0$, by some natural constant width circuit or branching program model.

Building upon the results in the present paper, such a characterization has recently been obtained for $\mathbf{ACC}^0$ (Hansen 2004): The class of functions computed by *planar* constant width, polynomial size circuits is exactly $\mathbf{ACC}^0$.

In this paper we consider a slightly more relaxed geometric restriction than upward planarity, yet more restrictive than planarity: We consider the functions computed by *cylindrical* polynomial size, constant width circuits (or nondeterministic branching programs). Informally (for formal definitions, see the next section), a layered circuit (branching program) is cylindrical if it can be embedded on the surface of a cylinder in such a way that each layer is embedded on a cross section of the cylinder (disjoint from the cross sections of the other layers), no wires intersect and all wires between two layers are embedded on the part of the cylinder between the two corresponding cross sections (see Figure 1.1).

It is immediate that constant width polynomial size cylindrical branching programs have more computational power than constant width polynomial size upward planar branching programs: The latter compute only functions in $\mathbf{AC}^0$ (Barrington *et al.* 1998) while the former may compute PARITY (see Figure 1.1). We ask what their exact computational power is and show that their power does not extend much beyond computing functions such as PARITY. Indeed, they can only compute functions in $\mathbf{ACC}^0$. To be precise, the first main result of this paper is the following *lower* bound on the power of cylindrical computation.

THEOREM 1.1. *Every Boolean function computed by a polynomial size* $\mathbf{\Pi_2} \circ \mathbf{MOD} \circ \mathbf{AC}^0$ *circuit is also computed by a constant width, polynomial size cylindrical nondeterministic branching program.*

By a $\mathbf{\Pi_2} \circ \mathbf{MOD} \circ \mathbf{AC}^0$ circuit we mean a polynomial size circuit with an AND gate at the output, a layer of OR gates feeding the AND gate, a layer of MOD_m gates (perhaps for many different constant-bounded values of m) feeding the OR gates and a (multi-output) $\mathbf{AC}^0$ circuit feeding the MOD gates. We prove Theorem 1.1 by a direct construction, generalizing and extending the simple idea of Figure 1.1.

Our second main result is the following *upper* bound on the power of cylindrical computation.

THEOREM 1.2. *Every Boolean function computed by a constant width, polynomial size cylindrical circuit is in* $\mathbf{ACC}^0$.

The proof of Theorem 1.2 is the most technical part of this paper. The simulation is done (as were many previous results about constant width computation) by using the theory of finite monoids and the results of Barrington & Thérien (1988). The notions of upward planarity and of cylindricality of digraphs share the property that all their arcs flow along a common direction. This allows these notions to be captured by local constraints, which allows one to transfer the analysis of the restricted branching programs and circuits into an appropriate algebraic setting. Thus, we show the inclusion by relating the computation of cylindrical circuits to solving the word problem of a certain finite monoid and then show that this monoid is solvable.

A standard simulation shows that every Boolean function computed by a constant width, polynomial size cylindrical nondeterministic branching program is also computed by a constant width, polynomial size cylindrical circuit. For completeness, we describe this simulation in Proposition 2.1. Thus, one can exchange "cylindrical nondeterministic branching program" with "cylindrical circuit" and vice versa in our two main results.

Organization of Paper. In Section 2, we formally define the notions of cylindrical branching program and circuits. We also give an overview of the algebraic tools we use. In Section 3, we show Theorem 1.1. In Section 5, we show Theorem 1.2. As this proof is quite technical, in Section 4 we also show, by a somewhat easier proof, that cylindrical branching programs (rather than circuits) compute only functions in $\mathbf{ACC}^0$. We conclude with some discussions and open problems in Section 6.

2. Preliminaries

Bounded Depth Circuits. Let $A \subset \{0, \ldots, m-1\}$. Using the notation of Grolmusz & Tardos (2000), a MOD_m^A gate takes n Boolean inputs $x_1, \ldots, x_n$ and outputs 1 if $\sum_{i=1}^n x_i \in A \pmod m$ and 0 otherwise. We let **MOD** denote the family of MOD_m^A gates for all constants m and all A. Similarly will **AND** and **OR** denote the family of unbounded fanin AND and OR gates.

If G is a family of Boolean gates and $\mathcal{C}$ is a family of circuits we let $G \circ \mathcal{C}$ denote the class of polynomial size circuit families consisting of a G gate taking circuits from $\mathcal{C}$ as inputs.

$\mathbf{AC}^0$ is the class of functions computed by polynomial size bounded depth circuits consisting of NOT gates and unbounded fanin AND and OR gates. $\mathbf{ACC}^0$ is the class of functions computed when we also allow unbounded fanin MOD gates computing MOD_k for constants k. We will also use $\mathbf{AC}^0$ and $\mathbf{ACC}^0$ to denote the class of circuits computing the languages in the respective classes.

Cylindrical Branching Programs and Circuits. A digraph $D = (V, A)$ is called *layered* if there is a partition $V = V_0 \cup V_1 \cup \cdots \cup V_h$ such that all arcs of A go from *layer* V_i to the *next layer* V_{i+1} for some i. We call h the *depth* of D, $|V_i|$ the width of layer i and $k = \max |V_i|$ the width of D. Finally, by the *size* of D we refer to the number of nodes $|V|$ in D.

Let $[k]$ denote the integers $\{1, \ldots, k\}$. For $a, b \in [k]$ where $a \not\equiv b+1 \pmod k$ we define the (cyclic) interval $[a, b]$ to be the set $\{a, \ldots, b\}$ if $a \leq b$ and $\{a, \ldots, k\} \cup \{1, \ldots, b\}$ if $a > b$. Furthermore let $(a, b) = [a, b] \setminus \{a, b\}$, and let $(a, b) = [k] \setminus \{a, b\}$ if $a \equiv b + 1 \pmod k$.

Let D be a layered digraph in which all layers have width k. We will assume that the nodes in each layer are numbered $1, \ldots, k$, and refer to nodes by these numbers. Then, D is called *cylindrical* if the following property is satisfied: For every pair of arcs going from layer l to layer $l+1$ connecting node a to node c and node b to node d the following must hold: Nodes in the interval (a, b) of layer l can only connect to nodes in the interval $[c, d]$ of layer $l+1$ and nodes in the interval (b, a) of layer l can only connect to nodes in the interval $[d, c]$ of layer $l+1$.

Notice this is equivalent to saying that nodes in the interval (c, d) of layer $l+1$ can only connect to nodes in the interval $[a, b]$ of layer l and nodes in the interval (d, c) of layer $l+1$ can only connect to nodes in the interval $[b, a]$ of layer l.

A *nondeterministic branching program*[1] is an acyclic digraph where all arcs are labeled with either a literal, i.e., a Boolean variable or its negation, or a Boolean constant, and which contains an initial and a terminal node. An input is accepted if and only if there is a path from the initial node to the terminal node in the digraph that results from substituting constants for the literals according to the input and then deleting arcs labeled with 0.

We will only consider branching programs in *layered form*, that is, viewed as a digraph it is layered. We can assume that the initial node is in the first layer and the terminal node in the last layer, and furthermore that these are the only nodes incident to arcs in these layers. We can also assume that all layers have the same number of nodes, by the addition of dummy nodes.[2]

These assumptions are made to ease the translation of deciding whether a branching program accepts a given input to evaluating a product of binary relations. In particular we will have the property that the branching program accepts if and only if the product relation is not the empty relation.

By a *cylindrical branching program* we will then mean a nondeterministic branching program in layered form, which is cylindrical when viewed as a digraph.

A *cylindrical circuit* is a circuit consisting of fanin 2 AND and OR gates and fanin 1 COPY gates, which when viewed as a digraph is a cylindrical digraph. Source nodes are either labeled with a literal (a Boolean variable or its negation) or labeled with a Boolean constant (we will call these ZERO and ONE gates), and are allowed to appear anywhere in the circuit and not only the first layer (as otherwise only functions depending on a constant number of input variables can be computed). The output gate is in the last layer. We can assume that all layers have the same number of nodes by adding dummy ONE gates to the first layer and dummy COPY gates to the other layers (we can add a COPY gate next to an arbitrary gate v in the same layer, and take one of the inputs of v as input to the COPY gate).

A standard simulation of nondeterministic branching programs by circuits extends to cylindrical branching programs and cylindrical circuits. We give the details for completeness.

[1] Our definition deviates slightly from the usual definition where nodes rather than arcs are labeled with literals and unlabeled nodes serve as special nondeterministic “choice”-nodes, but it is easily seen to be polynomially equivalent - also in the cylindrical case - and it is more convenient for us.

[2] In the figures, however, we have not added these dummy nodes. In particular, the first and last layers only contain the initial and terminal nodes.

Proposition 2.1. *Every function computed by a width k, depth d cylindrical branching program is also computed by a width $O(k)$, depth $O(d \log k)$ cylindrical circuit.*

Proof. Replace every node in the branching program by an OR-gate. Replace each arc, going from, say, node u to node v and labeled with the literal x, by a small circuit consisting of a COPY gate taking node u as its input and feeding into an AND gate which takes the literal x as its other input. The output of the new AND feeds gate v.

This transformation clearly preserves the cylindricality of the graph. Also, the width of the circuit is linear in the width of the branching program, since cylindricality implies there are only $O(k)$ arcs between layers. The resulting OR-gates may have fan-in bigger than two. We replace each such gate with a tree of fan-in two OR-gates, preserving the width and blowing up the depth by at most a factor of $O(\log k)$. □

Monoids and Groups. Let x and y be elements of a group G. The *commutator* of x and y is the element $x^{-1}y^{-1}xy$. The subgroup $G^{(1)}$ of G generated by all of the commutators in G is called the *commutator subgroup* of G. In general, let $G^{(i+1)}$ denote the commutator subgroup of $G^{(i)}$. G is *solvable* if $G^{(n)}$ is the trivial group for some n. It follows that an Abelian group, and in particular a cyclic group, is solvable.

A *monoid* is a set M with an associative binary operation and a two sided identity. A subset G of M is a *group in M* if it is a group with respect to the operation of M. Note that a group G in M is not necessarily a submonoid of M as the identity element of G may not be equal to the identity element of M. M is called *solvable* if every group in M is solvable. The *word problem* for a finite monoid M is the computation of the product $x_1 x_2 \ldots x_n$ given $x_1, x_2, \ldots, x_n$ as input. A theorem by Barrington & Thérien (1988) states that the word problem for a solvable finite monoid is in $\mathbf{ACC}^0$.

Representation of Inputs. We will several times construct circuits which take as input or compute results that are not Boolean, but can be quite complicated. In all cases, however, there is a natural way to divide these into elements of a finite set, which can then be encoded separately in an arbitrary way using a finite number of Boolean variables. In particular, for constant width cylindrical branching programs and cylindrical circuits, we encode each layer separately. When dealing with elements of monoids, the monoid will be fixed and we can thus encode any element by a finite number of Boolean variables.

3. Simulation of Bounded Depth Circuits by Cylindrical Branching Programs

In this section, we prove Theorem 1.1. As a starting point, we shall use the "only if" part of the following correspondence established by Vinay (1996) and Barrington *et al.* (1998). We include here a proof of the "only if" part for completeness.

THEOREM 3.1. *A language is in* $\mathbf{AC}^0$ *if and only if it is accepted by a polynomial size, constant width upward planar branching program.*

Here an *upward planar* branching program is a layered branching program satisfying that for every pair of arcs going from layer l to layer $l+1$ connecting node a to node c and node b to node d, if $a < b$ then $c \leq d$.

We need some simple observations. First observe that if we can simulate a class of circuits $\mathcal{C}$ by upward planar branching programs, then we can also simulate $\mathbf{AND} \circ \mathcal{C}$ by upward planar branching programs by simply concatenating the appropriate branching programs. Similarly, if we can simulate a class of circuits $\mathcal{C}$ by cylindrical branching programs, then we can also simulate $\mathbf{AND} \circ \mathcal{C}$ by cylindrical branching programs.

Another way to combine branching programs is by *substitution* where we simply substitute a branching program for the arcs corresponding to a particular literal. The effect of this is captured in the following two lemmas.

LEMMA 3.2. *If* $f(x_1, \ldots, x_n)$ *is computed by an upward planar branching program of size* s_1 *and width* w_1 *and* $g_1, \ldots, g_n$ *and* $\overline{g}_1, \ldots, \overline{g}_n$ *are computed by upward planar branching programs, each of size* s_2 *and width* w_2 *then* $f(g_1, \ldots, g_n)$ *is computed by an upward planar branching program of size* $O(s_1 s_2)$ *and width* $O(w_1 w_2)$.

LEMMA 3.3. *If* $f(x_1, \ldots, x_n)$ *is computed by a cylindrical branching program of size* s_1 *and width* w_1 *and* $g_1, \ldots, g_n$ *and* $\overline{g}_1, \ldots, \overline{g}_n$ *are computed by upward planar branching programs, each of size* s_2 *and width* w_2 *then* $f(g_1, \ldots, g_n)$ *is computed by a cylindrical branching program of size* $O(s_1 s_2)$ *and width* $O(w_1 w_2)$.

Combining Lemma 3.2 with the construction in Figure 3.1, simulating an OR gate, we have established the "only if" part of Theorem 3.1.

Simulation of a MOD_m^A gate can be done as shown in Figure 3.2 if one disregards the top nodes in the first and last layers and modifies the connections

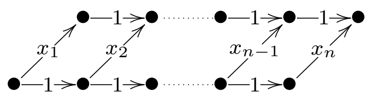

Figure 3.1: An upward planar branching program computing OR.

from the second-to-last layer to take the set A into account. Thus, combining this construction with Lemma 3.3, the "only if" part of Theorem 3.1 and the closure of cylindrical branching programs under polynomial fan-in AND, we have established that we can simulate $\mathbf{AND} \circ \mathbf{MOD} \circ \mathbf{AC}^0$ circuits by bounded width polynomial size cylindrical circuits.

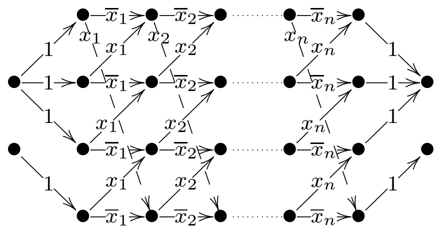

Figure 3.2: A cylindrical branching program fragment for MOD_4. The dashed arcs indicate a cylindrical embedding. As shown this does not constitute a cylindrical branching program, since there is more than one node in the first and last layers connected to their neighboring layers.

The construction as shown in Figure 3.2 has actually more use, by seeing it as computing elements of M_2, where M_2 is the monoid of binary relations on $\{1, 2\}$. This is done as follows: In the first and last layer there are exactly two nodes. The relation computed describes exactly the paths from a node in the first to a node in the last layer in the graph obtained by substituting constants for the literals according to a given input and deleting arcs labeled with 0. The general construction of a branching program fragment for MOD_m^A taking n inputs is as follows: Without loss of generality we can assume that $|A| = 1$ and in fact $A = \{0\}$ since we aim for simulating $\mathbf{OR} \circ \mathbf{MOD}$. The branching program fragment will have $n + 3$ layers. The first and last layer of width 2 and the middle layers of width m. The top node in the first layer has arcs to all nodes in the second layer but node 1 and the bottom node has an arc to node 1. Similarly, the top node in the last layer has arcs from all nodes in the next-to-last layer but node 1 and the bottom node has an arc from node

1. The nodes in the middle layers represent the sum of a prefix of the input modulo m in the obvious way.

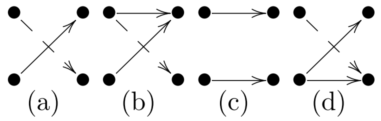

Figure 3.3: Some elements of M_2.

Consider now the elements of M_2 shown in Figure 3.3. The branching program fragment just described computes (a) and (b) for $m = 2$ and $m > 2$ respectively, when the simulated MOD gate evaluates to 0. In both cases, the fragment computes (c) when the simulated MOD gate evaluates to 1.

We can now describe our construction for simulating **OR** ∘ **MOD** circuits. The construction interleaves branching program fragments computing (d) between the branching program fragments for the MOD gates. This can be seen as a way of "short circuiting" the branching program in the case that one of the MOD gates evaluates to 1. Finally we add layers at both ends, connecting to the bottom nodes in the previous first and last layers, effectively making these the initial and terminal nodes. The entire construction is shown in Figure 3.4. The correctness can easily be verified.

The simulation of **OR** ∘ **MOD** circuits, the "only if" part of Theorem 3.1, Lemma 3.3, and the closure of cylindrical branching programs under polynomial fan-in AND, together completes the proof of Theorem 1.1.

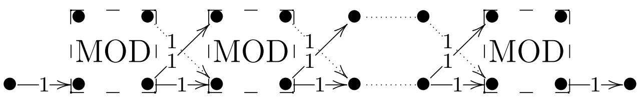

Figure 3.4: A cylindrical branching program computing **OR** ∘ **MOD**.

4. Simulation of Cylindrical Branching Programs by Bounded Depth Circuits

In this section, we prove that constant width polynomial size cylindrical nondeterministic branching programs compute only functions in $\mathbf{ACC}^0$. The proof is simpler than (but similar to) the proof to be presented in the next section of the stronger statement of Theorem 1.2.

In fact, we shall prove that for fixed k the following "branching program value problem" BPV_k is in $\mathbf{ACC}^0$: Given a width k cylindrical branching program *and* a truth assignment to its variables, decide if the program accepts. As any function computed by a width k cylindrical polynomial size branching program is a Skyum-Valiant projection (Skyum & Valiant 1985) of BPV_k (we can fix the cylindrical branching program by assigning constants to the first part of the input to BPV_k), we will be done.

We shall prove that BPV_k is in $\mathbf{ACC}^0$ by showing that it reduces, by an $\mathbf{AC}^0$ reduction, to the word problem of the monoid M_k we define next. Then, we show that the monoid M_k is solvable, and since this implies, by the result of Barrington & Thérien (1988) that the word problem for M_k is in $\mathbf{ACC}^0$, our proof will be complete.

We define M_k to be the monoid of *binary relations* on $[k]$ which capture the calculation of width k branching programs embedded on a cylinder in the following sense: M_k is the monoid generated by all the relations which express how arcs can connect two adjacent layers in a width k cylindrical digraph. The monoid operation is the usual composition operation of binary relations, i.e., if $A, B \in M_k$ and $x, y \in [k]$ then $xABy \Leftrightarrow \exists z : xAz \wedge zBy$.

BPV_k reduces to the word problem for M_k by the following $\mathbf{AC}^0$ reduction: Substitute constants for the literals in the branching program according to the truth assignment. Consider now the cylindrical digraph D consisting only of arcs which have the constant 1 associated. Then, the branching program accepts the input given if and only if there is a path from the initial node in the first layer to the terminal node in the last layer of D. We can decide this by simply decomposing D into a sequence $A_1, A_2, \ldots, A_h$ of elements from M_k, computing the product $A = A_1 A_2 \cdots A_h$ and checking whether this is different from the zero element of M_k (i.e., the empty relation).

Thus, we just need to show that M_k is solvable. Our proof is finished by the following much stronger statement.

Proposition 4.1. *All groups in M_k are cyclic.*

Proof. Let $G \subseteq M_k$ be a group with identity E. Let $A \in G$ and let R be the set of all x such that xEx. As will be shown next it will be enough to consider elements of R to capture the structure of A.

Let $x \in R$. Since $AA^{-1} = E$ there exists z such that xAz and $zA^{-1}x$. Since $A^{-1}A = E$ it follows zEz, that is, $z \in R$. Hence there exists a function $\pi_A : R \to R$ such that

$$\forall x \in R : xA\pi_A(x) \wedge \pi_A(x)A^{-1}x \ .$$

To see that A is completely described by π_A, we define a relation $\hat{A}$ on $[k]$ such that $x\hat{A}y \Leftrightarrow \pi_A(x) = y$. That is, $\hat{A}$ is just π_A viewed as a relation. Since $\hat{A} \subseteq A$ it follows $E\hat{A}E \subseteq EAE = A$. Conversely let xAy. Since $E^kA = A$ there exists $z \in R$ such that xEz and zAy. Since $\pi_A(z)A^{-1}z$ we get $\pi_A(z)Ey$. That is xEz, $z\hat{A}\pi_A(z)$ and $\pi_A(z)Ey$. Thus $xE\hat{A}Ey$. Hence we obtain that $A = E\hat{A}E$.

To ease the remainder of the proof, we will first simplify the structure of the elements of G using the following equivalence relation on $[k]$ defined by

$$x \sim y \Leftrightarrow (xEy \wedge yEx) \vee x = y \ .$$

Basically π_A describes a permutation of the equivalence classes consisting of elements of R. We will show that, without loss of generality we can assume that the equivalence classes with respect to $\sim$ are of size 1. This will then imply that π_A is a permutation and that $\{\pi_A \mid A \in G\}$ is a group isomorphic to G. Finally by showing that π_A is in fact a *cyclic shift* of R (i.e., π_A is in the cyclic group generated by the cycle $(r_1\ r_2 \cdots r_m)$ where $r_1 < r_2 < \cdots < r_m$ are the elements of R) we will be done.

Let $A \in G$. If $x \sim x'$ and $y \sim y'$ then $xAy \Leftrightarrow x'Ay'$, since $EAE = A$. Thus A gives rise to a relation $\tilde{A}$ on $[k]/\sim$ where $xAy \Leftrightarrow [k]_x\tilde{A}[k]_y$ and it will follow that $\{\tilde{A} \mid A \in G\}$ is an isomorphic group of G.

For this we need to show that $\widetilde{AB} = \tilde{A}\tilde{B}$. This follows since $[k]_x\widetilde{AB}[k]_z \Leftrightarrow xABz \Leftrightarrow \exists y : xAy \wedge yBz \Leftrightarrow \exists y : [k]_x\tilde{A}[k]_y \wedge [k]_y\tilde{B}[k]_z \Leftrightarrow [k]_x\tilde{A}\tilde{B}[k]_z$

We can find an isomorphic copy of this group in M_k as follows. Choose for each equivalence class $[k]_x$ a representative $r([k]_x)$ in $[k]_x$. Define a relation C on $[k]$ such that $xCy \Leftrightarrow x = y = r([k]_x)$. Thus $\forall x : r([k]_x)Cr([k]_x)$. Let $\sigma : G \to M_k$ be given by $\sigma(A) = CAC$. Then $\sigma(G)$ is the desired isomorphic copy of G. We can thus assume that the equivalence classes with respect to $\sim$ are of size 1.

We now return to the study of π_A. The following property, that for $x, y \in R$ it holds that $xEy \Leftrightarrow \pi_A(x)E\pi_A(y)$, is satisfied:

If xEy then $\pi_A(x)A^{-1}y$ since $A^{-1}E = A^{-1}$. As $A^{-1}A = E$ it follows that $\pi_A(x)E\pi_A(y)$.

Conversely if $\pi_A(x)E\pi_A(y)$ then $xA\pi_A(y)$ since $xA\pi_A(x)$ and $AE = A$. As $\pi_A(y)A^{-1}y$ and $AA^{-1} = E$ it then follows that xEy.

We can now conclude that π_A is a permutation on R: If $\pi_A(x) = \pi_A(y)$ then $\pi_A(x) \sim \pi_A(y)$ so $x \sim y$, that is, $x = y$. Also π_A is uniquely defined : Assume $\hat{\pi}_A : R \to R$ satisfies

$$\forall x \in R : xA\hat{\pi}_A(x) \wedge \hat{\pi}_A(x)A^{-1}x \ .$$

Let $x \in R$. We then obtain $\pi_A(x) \sim \hat{\pi}_A(x)$ so $\pi_A(x) = \hat{\pi}_A(x)$. Hence $\pi_A = \hat{\pi}_A$.

Now we can conclude that $\{\pi_A \mid A \in G\}$ is a permutation group which is isomorphic to G. For this we need to show that $\pi_{AB} = \pi_B \circ \pi_A$.

Let $x \in R$. Since $xA\pi_A(x)$ and $\pi_A(x)B\pi_B \circ \pi_A(x)$ it follows that $xAB\pi_B \circ \pi_A(x)$. Since $\pi_B \circ \pi_A(x)B^{-1}\pi_A(x)$ and $\pi_A(x)A^{-1}x$ it follows $\pi_B \circ \pi_A(x)B^{-1}A^{-1}x$, i.e., $\pi_B \circ \pi_A(x)(AB)^{-1}x$.

Since π_{AB} is uniquely defined the result follows.

To show that $\{\pi_A \mid A \in G\}$ is cyclic we need the following fact, which easily follows from the definition of cylindricality

Fact. *Let A be a relation which can be directly embedded on a cylinder. Let $p_1 < p_2 < \dots p_m$ and $q_1 < q_2 < \cdots < q_m$ and π a permutation on $[m]$ such that $\forall i : p_i A q_{\pi(i)}$. Then π is a cyclic shift of $[m]$ (i.e., π is in the cyclic group generated by the cycle $(1\ 2 \cdots m)$).*

Now let $r_1 < r_2 < \cdots < r_m$ be the elements of R. Write $A \in G$ as $A = A_1 A_2 \dots A_h$ where the A_i's can be directly embedded on the cylinder. Since $r_i A\pi_A(r_i)$, we have for all i, elements of $[k]$, $r_i = q_i^0, q_i^1, \dots, q_i^h = \pi_A(r_i)$ such that $q_i^j A_{j+1} q_i^{j+1}$. For fixed j, all the q_i^j's are distinct: If not we would have i_1 and i_2 such that $r_{i_1} A\pi_A(r_{i_2})$ and $r_{i_2} A\pi_A(r_{i_1})$. But then since $\pi_A(r_{i_1})A^{-1}r_{i_1}$ and $\pi_A(r_{i_2})A^{-1}r_{i_2}$ we get $r_{i_1} E r_{i_2}$ and $r_{i_2} E r_{i_1}$. That is $r_{i_1} \sim r_{i_2}$ which implies $r_{i_1} = r_{i_2}$. Now by the fact and induction on h we have a permutation π which is a cyclic shift of $[m]$ such that $r_{\pi(i)} = \pi_A(r_i)$. Thus π_A is a cyclic shift of R and we can conclude that G is cyclic. □

5. Simulation of Cylindrical Circuits by Bounded Depth Circuits

In this section we prove Theorem 1.2. Following the outline of the last section, we consider for fixed k the following "circuit value problem" CV_k: Given a width k cylindrical circuit *and* a truth assignment to its input variables, decide if the circuit evaluates to 1. We shall reduce CV_k, by an $\mathbf{AC}^0$ reduction, to the word problem of the monoid $\hat{N}_k$ defined next, which will be proved to be solvable. By the result of Barrington & Thérien (1988) it then follows that CV_k is in $\mathbf{ACC}^0$.

Consider a width k cylindrical circuit C of the following form: The first layer consists of k gates labeled with k distinct unnegated variables. If C contains source nodes in other layers, these must be ZERO or ONE gates. We shall call such a circuit *simplistic*. We can view C as computing a function mapping $\{0,1\}^k$ to $\{0,1\}^k$ by interpreting the gates in the last layer as output gates.

We let N_k be the monoid of functions mapping $\{0,1\}^k$ to $\{0,1\}^k$ which are computed by simplistic circuits with no ZERO or ONE gates anywhere in the circuit.

We let $\hat{N}_k$ be the monoid of functions mapping $\{0,1\}^k$ to $\{0,1\}^k$ which are computed by simplistic circuits, where we do allow gates to be ZERO or ONE gates, but *only* in the last layer of the circuit.

We will call C a N_k circuit and $\hat{N}_k$ circuit, respectively. Note that the functions we have just defined are *monotone* functions. That $\hat{N}_k$ is in fact a monoid follows from the following lemma.

LEMMA 5.1. *Any function computed by a simplistic circuit is in* $\hat{N}_k$.

PROOF. Consider any simplistic circuit of depth d. By induction, the depth $d-1$ subcircuit consisting of the first d of the $d+1$ layers can be computed by a simplistic circuit with all the ZERO and ONE gates in layer $d-1$. Consider now a gate in layer d which has an arc from a ZERO or ONE gate. If it always evaluates to a constant we can simply replace it by a ZERO or ONE gate. Otherwise the gate is an OR or an AND gate which can be replaced by a COPY gate, copying the non-constant input of the old gate. Now all ZERO or ONE gates in layer $d-1$ can simply be replaced by dummy COPY gates, since they have no outgoing arcs anymore. $\square$

We are now able to describe the $\mathbf{AC}^0$ reduction of CV_k to the word problem for $\hat{N}_k$: Substitute constants according to the truth assignment for all source nodes labeled with literals. Each layer of the circuit except the first can now be viewed as a depth 1 $\hat{N}_k$ circuit by preceding it by a layer of k source nodes labeled with k distinct unnegated variables. Let $C_1, C_2, \ldots, C_h$ be the circuits obtained this way, and compute the corresponding elements $f_1, f_2, \ldots, f_h$ of $\hat{N}_k$. Now compute the product $f = f_h \circ \cdots \circ f_2 \circ f_1$ and evaluate it on the constants appearing in the first layer of the circuit. The output of the circuit can then be found in the entry of this result corresponding to the output gate of the circuit.

Now we just need to show that $\hat{N}_k$ is solvable. We will in fact, as in the previous section, obtain the stronger result that all its groups are cyclic. First we show that it is sufficient to consider N_k.

PROPOSITION 5.2. *Every group in* $\hat{N}_k$ *is isomorphic to a group in* N_k.

PROOF. Suppose $G \subseteq \hat{N}_k$ is a group with identity e.

We construct an injective (monoid) homomorphism $\phi : G \to N_k$. This proves the result. First we need to prove the following claims.

Claim 5.3. *An output gate of a $\hat{N}_k$ circuit computes a constant function if and only if it is a ZERO or ONE gate.*

Proof. An AND, OR, or COPY gate will output 0 if the circuit is fed the input 0^k and 1 if the circuit is fed the input 1^k. □

Claim 5.4. *Let f_1 and f_2 be elements of G computed by $\hat{N}_k$ circuits C_1 and C_2. The output gates of C_1 and C_2 that are constant are the same and their output values are the same in both circuits.*

Proof. Let C_1' be a $\hat{N}_k$ circuit computing f_1^{-1}. Note that $C_1 \circ C_1' \circ C_2$ also computes f_2. Hence, if C_1 has a ONE gate as output gate, then by Claim 5.3, the same output gate is also a ONE gate in C_2. Analogously, if C_1 has a ZERO gate as output gate, the same output gate is also a ZERO gate in C_2. By repeating the argument with the roles of C_1 and C_2 reversed, we have proved the claim. □

By Claim 5.4, a fixed set of output gates are constant, ZERO or ONE, for all $\hat{N}_k$ circuits computing functions in G. Let $S \subseteq [k]$ be the set of indices of those gates. We may also identify S with a subset of the gates in an arbitrary layer of the circuit, and we shall do so implicitly in the following. Construct the homomorphism ϕ as follows: Let $f \in G$ be computed by an $\hat{N}_k$ circuit C. Then $\phi(f)$ is the function computed by the following circuit $\phi(C)$, obtained by modifying C as follows: We first add an extra layer of source nodes before the first layer C. We next modify the source nodes in the first layer of C. Those not in S are replaced by COPY gates copying the corresponding source node from the new layer. Those in S are replaced by constant gates (ZERO or ONE) in accordance with the constant output gates. Then an extra layer of COPY gates is added after the last layer of the circuit. Those with an index not in S copy the corresponding gate in the previous layer. For a gate with an index $a \in S$ we pick an index $b \notin S$, where $a \equiv b + i \pmod{k}$ and $i > 0$ is minimal, and let the gate with index a copy the gate with index b in the previous layer. The resulting circuit is then reduced as in Lemma 5.1, eliminating all ZERO and ONE gates, except possibly those in the last layer. The definition is clearly independent of the choice of C, i.e., it depends only on f.

First we make the observation that ϕ is a homomorphism from G to $\hat{N}_k$. For a circuit C computing a function in G, we will in the following by $\phi(C)$ denote the circuit modified in the way described in the definition of ϕ. Let f_1 and f_2 be elements of G computed by $\hat{N}_k$ circuits C_1 and C_2. We will argue that $\phi(C_1 \circ C_2)$ computes the same function as $\phi(C_1) \circ \phi(C_2)$, that is that the two extra layers in $\phi(C_1) \circ \phi(C_2)$ do not have influence on the computation. The output gates of $\phi(C_2)$ in S are ignored by $\phi(C_1)$ which is instead fed the same constants as C_2 is feeding C_1. The output gates of $\phi(C_2)$ not in S are feeding the inputs of $\phi(C_1)$ exactly as the outputs of C_2 are feeding C_1. Thus an output gate of $\phi(C_1) \circ \phi(C_2)$ computes the same function as the corresponding output gate of $\phi(C_1 \circ C_2)$.

We next argue that $\phi(C)$ is a N_k circuit: It contains no ZERO or ONE gates, not even in its last layer. Let C_e be a $\hat{N}_k$ circuit computing the identity e of G. Each output gate v in $\phi(C)$ computes the same function as some output gate v' in $C \circ C_e$, where v' is not in S, and hence v is not computing a constant function.

Thus ϕ is a homomorphism from G to N_k. We now just have to argue that it is injective and we are done. Let C_1 and C_2 be $\hat{N}_k$ circuits computing $f_1 \neq f_2$. Let $i \in [k]$ be an index so that output i of C_1 and output i of C_2 compute different functions. The circuit $C_1 \circ C_e$ also computes f_1 and the circuit $C_2 \circ C_e$ also computes f_2, so output i of $C_1 \circ C_e$ and output i of $C_2 \circ C_e$ compute different functions. Since output i is not in S, output i of $\phi(C_1)$ computes the same function as output i of $C_1 \circ C_e$ and output i of $\phi(C_2)$ computes the same function as output i of $C_2 \circ C_e$, hence $\phi(C_1)$ and $\phi(C_2)$ compute different functions. $\square$

Now we can turn to the study of the monoid N_k. Note that it is generated by the depth 1 N_k circuits. This will be useful in some of the following properties we will prove, since it allows us, by induction, to just consider depth 1 circuits in many of the following lemmas.

As in Barrington *et al.* (1999) it will be convenient to identify input vectors in $\{0,1\}^k$ with its set of *maximal 1-intervals*, only here we consider *cyclic* intervals. For example is the vector 1010011011 identified with the set of intervals $\{[3,3],[6,7],[9,1]\}$.

We will only do this identification for inputs which contain at least one interval, that is, we disregard the vectors 0^k and 1^k (The vector 1^k is omitted since we have defined cyclic intervals in terms of their endpoints).

LEMMA 5.5. *Let $x \in \{0,1\}^k$ contain m intervals and $f \in N_k$. Then $f(x)$ contains at most m intervals. If $f(x)$ in fact contains m intervals and $m \geq 1$ then $f(x) = \{f(I) \mid I \in x\}$.*

PROOF. Clearly $f(0^k) = 0^k$ and $f(1^k) = 1^k$ so we can assume $m \geq 1$. Assume that C is a depth 1 N_k circuit computing f. Let $c, d \in [k]$ be gates which evaluate to 1 on input x. Each of c and d must have an arc from an interval of

x. Suppose c and d have arcs from the same interval I in x, from a to c and b to d, say. Either $[a, b] \subseteq I$ or $[b, a] \subseteq I$. If $[a, b] \subseteq I$ then all gates in the interval (c, d) evaluate to 1 since they must take arcs from $[a, b] \subseteq I$. Similarly if $[b, a] \subseteq I$ then all gates in the interval (d, c) evaluate to 1. Thus, gates which have arcs from the same interval in x are in the same interval of $f(x)$, and the first part of the lemma follows.

Now assume that there also are m intervals in $f(x)$. To show the second part we have to rule out that an interval in $f(x)$ can take arcs from more than one interval in x. But if that were the case there would also be different intervals with arcs from the same interval in x, contradicting the above. $\square$

Any subset of $\{0, 1\}^k$ becomes a poset by lifting the order $0 < 1$ pointwise. The functions in N_k are monotone with respect to this order. When considering only intervals, this order coincides with the subset inclusion order and we will use these orders interchangeably.

LEMMA 5.6. *Let $\{[a_i, b_i] \mid i = 1, \ldots, m\}$ and $\{[c_i, d_i] \mid i = 1, \ldots, m\}$ be antichains of intervals in $\{0, 1\}^k$ (i.e., they are sets of pairwise incomparable elements), such that $a_1 < \cdots < a_m$ and $c_1 < \cdots < c_m$. Let $f \in N_k$ and assume π is a permutation on $[m]$ such that $f([a_i, b_i]) = [c_{\pi(i)}, d_{\pi(i)}]$. Then π is a cyclic shift of $[m]$.*

PROOF. Assume C is a depth 1 N_k circuit computing f. In the following all index arithmetic is done modulo m. There must be an arc from $[a_i, b_i] \setminus [a_{i+1}, b_{i+1}]$ to $[c_{\pi(i)}, d_{\pi(i)}] \setminus [c_{\pi(i)+1}, d_{\pi(i)+1}]$ for all i since otherwise $[c_{\pi(i)}, d_{\pi(i)}] \subseteq [c_{\pi(i)+1}, d_{\pi(i)+1}]$. From this it follows that $\pi(i) + 1 = \pi(i + 1)$ for all i and π is thus a cyclic shift of $[m]$. $\square$

Say that an interval $[a_0, b_0] \subseteq [a_1, b_1] \cap [a_2, b_2]$ is *in between* intervals $[a_1, b_1]$ and $[a_2, b_2]$ if there is $a_3 \in [a_1, b_1] \setminus [a_2, b_2]$ and $b_3 \in [a_2, b_2] \setminus [a_1, b_1]$ such that $[a_0, b_0] \subseteq (a_3, b_3)$.

The following lemma, which provides an alternative definition, is easily verified.

LEMMA 5.7. *For an interval $[a_0, b_0] \subseteq [a_1, b_1] \cap [a_2, b_2]$ the following is equivalent.*

(i) $[a_0, b_0]$ *is in between* $[a_1, b_1]$ *and* $[a_2, b_2]$.

(ii) $\{[a_1, b_1], [a_2, b_2]\}$ *form an antichain and* $[a_2, b_0] \subseteq [a_1, b_0]$.

From this alternative definition we observe that for a fixed interval $[a_0, b_0]$ we have defined a *linear order* on any antichain of intervals, which all contain $[a_0, b_0]$.

The next lemma, which uses the first definition in its proof, states that these linear orders are respected by the functions in N_k.

LEMMA 5.8. *Suppose* $[c_0, d_0] \subseteq [c_1, d_1] \cap [c_2, d_2]$ *is in between* $[c_1, d_1]$ *and* $[c_2, d_2]$. *Suppose* $f \in N_k$ *maps intervals* $[a_i, b_i]$ *to* $[c_i, d_i]$ *for* $i = 0, 1, 2$. *If* $[a_0, b_0] \subseteq [a_1, b_1] \cap [a_2, b_2]$ *then* $[a_0, b_0]$ *is in between* $[a_1, b_1]$ *and* $[a_2, b_2]$.

PROOF. Assume f is computed by a depth 1 N_k circuit C.

By assumption there is $c_3 \in [c_1, d_1] \setminus [c_2, d_2]$ and $d_3 \in [c_2, d_2] \setminus [c_1, d_1]$ such that $[c_0, d_0] \subseteq (c_3, d_3)$. There must exist $a_3 \in [a_1, b_1] \setminus [a_2, b_2]$ such that there is an arc from a_3 to c_3 in C, since otherwise we would have $c_3 \in [c_2, d_2]$. Similarly there exists $b_3 \in [a_2, b_2] \setminus [a_1, b_1]$ such that there is an arc from b_3 to d_3 in C. Since $[c_0, d_0] \subseteq (c_3, d_3)$ all arcs to $[c_0, d_0]$ must therefore go from $[a_3, b_3]$. It follows $[a_0, b_0] \subseteq [a_3, b_3]$, in fact $[a_0, b_0] \subseteq (a_3, b_3)$ since $[a_0, b_0] \subseteq [a_1, b_1] \cap [a_2, b_2]$. □

In the following, let G be a group in N_k with identity e. Since $e \circ e = e$ we get that e is the identity mapping on the image of e, $\operatorname{Im} e$. Thus any $f \in G$ is a permutation of $\operatorname{Im} e$, since $f \circ f^{-1} = f^{-1} \circ f = e$ and $e \circ f = f$.

LEMMA 5.9. *Let* $f \in G$ *and* $\mathcal{S} \subseteq \operatorname{Im} e$. *If* f *is a permutation on* $\mathcal{S}$ *then* f *is a permutation on the set of minimal elements in* $\mathcal{S}$, $\min(\mathcal{S})$, *with respect to* $<$.

PROOF. Assume on the contrary that $x \in \min(\mathcal{S})$ such that $f(x) \notin \min(\mathcal{S})$. Then there exists $y \in \min(\mathcal{S})$ such that $y < f(x)$. But then $f^{-1}(y) < x$, which contradicts $x \in \min(\mathcal{S})$. □

Now we can finally complete the proof of Theorem 1.2 by the following proposition.

PROPOSITION 5.10. *All groups in* N_k *are cyclic.*

PROOF. Let G be a group in N_k with identity e. Viewing e as a map taking sets of intervals to sets of intervals, we let $\mathcal{I}$ be the union of all sets in $\operatorname{Im} e$.

Let $f \in G$. Since $f \circ e = f$ it follows that f is completely described by its restriction to $\operatorname{Im} e$. By Lemma 5.5, f is furthermore completely described by its restriction to $\mathcal{I}$. (This follows since, as f has an inverse it must preserve the number of intervals in any input from $\operatorname{Im} e$).

We decompose $\mathcal{I}$ into antichains, by repeatedly removing the set of minimal elements, i.e., we let $\mathcal{I}_1 = \min(\mathcal{I})$ and for $j > 1$, $\mathcal{I}_j = \min\left(\mathcal{I} \setminus \bigcup_{i<j} \mathcal{I}_i\right)$.

By Lemma 5.9 and induction, f is a permutation on $\mathcal{I}_j$ and furthermore a *cyclic shift* by Lemma 5.6. Thus G acts as a cyclic group on $\mathcal{I}_j$. It follows that G is Abelian, which is clearly more than enough to prove Theorem 1.2. To obtain that G is in fact cyclic, we will argue that f is in fact completely described by its restriction to $\mathcal{I}_1$.

For an interval I let $\mathcal{I}_j(I)$ denote the set $\{J \in \mathcal{I}_j \mid I \subseteq J\}$. Pick an interval $I \in \mathcal{I}_1$ such that $\mathcal{I}_j(I) \neq \emptyset$. Observe that $\mathcal{I}_j(f(I)) = \{f(J) \in \mathcal{I}_j \mid I \subseteq J\}$. The intervals of $\mathcal{I}_j(I)$ are linearly ordered by the property of having I in between, and likewise are the intervals of $\mathcal{I}_j(f(I))$ linearly ordered by the property of having $f(I)$ in between. By Lemma 5.8, (applied to f^{-1}) the order of the intervals in $\mathcal{I}_j(I)$ is the same as the order of the corresponding intervals in $\mathcal{I}_j(f(I))$. In particular is the first interval in $\mathcal{I}_j(I)$ mapped into the first interval in $\mathcal{I}_j(f(I))$ by f, which is sufficient to describe how f acts on $\mathcal{I}_j$, since f is a cyclic shift on $\mathcal{I}_j$. $\square$

6. Conclusion and Open Problems

We have located the class of functions computed by constant width polynomial size cylindrical circuits (or nondeterministic branching programs) between $\mathbf{\Pi}_2 \circ \mathbf{MOD} \circ \mathbf{AC}^0$ and $\mathbf{ACC}^0$. It would be very interesting to get an exact characterization of the power of cylindrical circuits and branching programs in terms of bounded depth circuits. It is not known whether $\mathbf{\Pi}_2 \circ \mathbf{MOD} \circ \mathbf{AC}^0$ is different from $\mathbf{ACC}^0$ and this seems a difficult problem to resolve, so we cannot hope for an unconditional separation of the power of cylindrical circuits from $\mathbf{ACC}^0$. On the other hand, it seems difficult to generalize the simulation of $\mathbf{\Pi}_2 \circ \mathbf{MOD} \circ \mathbf{AC}^0$ by cylindrical branching programs to handle more than one layer of MOD gates and we tend to believe that such a simulation is in general not possible. Thus, one could hope that by better understanding the structure of the monoids we have considered in this paper, it would be possible to prove an upper bound seemingly better than $\mathbf{ACC}^0$, such as for instance $\mathbf{AC}^0 \circ \mathbf{MOD} \circ \mathbf{AC}^0$.

It would also be interesting to separate the power of branching programs from the power of circuits. As circuits can be trivially negated while preserving cylindricality, we immediately have that not only $\mathbf{\Pi}_2 \circ \mathbf{MOD} \circ \mathbf{AC}^0$ but also $\mathbf{\Sigma_2} \circ \mathbf{MOD} \circ \mathbf{AC}^0$ can be simulated by constant width polynomial size cylindrical circuits. On the other hand, we don't know if $\mathbf{\Sigma_2} \circ \mathbf{MOD} \circ \mathbf{AC}^0$ can be simulated by polynomial size constant width cylindrical branching programs. Note that in the upward planar case, both models capture $\mathbf{AC}^0$ and in the geometrically unrestricted case, both models capture $\mathbf{NC}^1$, so it is not clear if one should *a priori* conjecture the cylindrical models to have different power. Note that if the models have identical power then they can simulate $\mathbf{AC}^0 \circ \mathbf{MOD} \circ \mathbf{AC}^0$. This follows from the fact that the branching program model is closed under polynomial fan-in AND while the circuit model is closed under negation.

An interesting problem concerns the blowup of width to depth when going from a cylindrical circuit or branching program to an $\mathbf{ACC}^0$ circuit. Our proof does not yield anything better than a doubly exponential blowup. Again, by better understanding the structure of the monoids we have considered, one could hope for a better upper bound.

Acknowledgements

The first two authors are supported by BRICS, Basic Research in Computer Science, a Centre of the Danish National Research Foundation.

We would like to thank the reviewers for their help in improving the presentation of the present work.

A preliminary version of this paper appeared in the proceedings of the 14th International Symposium on Fundamentals of Computation Theory (FCT 2003) (Hansen *et al.* 2003).

References

David A. Barrington (1989). Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$. *J. Comput. System Sci.* **38**(1), 150–164.

David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen & Sven Skyum (1998). Searching constant width mazes captures the $\mathbf{AC}^0$ hierarchy. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, 73–83. Springer.

David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen & Sven Skyum (1999). On monotone planar circuits. In *14th Annual IEEE Conference on Computational Complexity*, 24–31. IEEE Computer Society Press.

David A. Mix Barrington & Denis Thérien (1988). Finite monoids and the fine structure of $\mathbf{NC}^1$. *Journal of the ACM (JACM)* **35**(4), 941–952.

Vince Grolmusz & Gábor Tardos (2000). Lower Bounds for $(\mathrm{MOD}_p - \mathrm{MOD}_m)$ Circuits. *SIAM Journal on Computing* **29**(4), 1209–1222.

Kristoffer Arnsfelt Hansen (2004). Constant Width Planar Computation Characterizes $\mathbf{ACC}^0$. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, 44–55. Springer.

Kristoffer Arnsfelt Hansen, Peter Bro Miltersen & V Vinay (2003). Circuits on Cylinders. In *14th International Symposium on Fundamentals of Computation Theory*, volume 2751 of *Lecture Notes in Computer Science*, 171–182. Springer.

S. Skyum & L. G. Valiant (1985). A complexity theory based on Boolean algebra. *Journal of the ACM (JACM)* **32**(2), 484–502.

V Vinay (1996). Hierarchies of Circuit Classes that are Closed under Complement. In *11th Annual IEEE Conference on Computational Complexity*, 108–117. IEEE Computer Society Press.

Manuscript received 20 February 2004

Kristoffer Arnsfelt Hansen
Department of Computer Science
University of Aarhus
Århus, Denmark
arnsfelt@daimi.au.dk

Peter Bro Miltersen
Department of Computer Science
University of Aarhus
Århus, Denmark
bromille@daimi.au.dk

V Vinay
Indian Institute of Science
Bangalore, India
vinay@csa.iisc.ernet.in