Bounded-depth Circuits: Separating Wires from Gates

[Extended Abstract]

Michal Koucký[*]
XXX,CWI
Amsterdam, Netherlands
mkoucky@cs.mcgill.ca

Pavel Pudlák[†]
Mathematical Institute of the Academy of Sciences,
Prague, Czech Republic
pudlak@math.cas.cz

Denis Thérien[*]
McGill University
Montréal, Québec, Canada
denis@cs.mcgill.ca

Categories and Subject Descriptors

F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes

General Terms

Theory

Keywords

Circuits, gates, etc.

ABSTRACT

We develop a new method to analyze the flow of communication in constant-depth circuits. This point of view allows us to prove new lower bounds on the number of wires required to recognize certain languages. We are able to provide explicit languages that can be recognized by AC^0 circuits with $O(n)$ gates but not with $O(n)$ wires, and similarly for ACC^0 circuits. We are also able to characterize exactly the regular languages that can be recognized with $O(n)$ wires, both in AC^0 and ACC^0 framework.

1. INTRODUCTION

Shallow boolean circuits have been extensively studied for many years, among other reasons because they constitute one of those rare models for which non-trivial lower bounds can be established. For example, 20 years after its first proof, the theorem that the regular language PARITY does not belong to the class AC^0, i.e. cannot be recognized by a constant-depth, polynomial-size circuit constructed with AND and OR gates of arbitrary fan-in, remains one of the important jewels in the theory of computation.

In this paper we are interested in studying the bounded-depth circuit complexity of languages in terms of their size, in the sense of number of wires. In the introduction of [10] it is claimed that integer ADDITION is an example of a function that can be computed by constant-depth circuits with $O(n)$ gates but not $O(n)$ wires. The lower bound is correct [5] but the stated upper bound was unfounded and is still not known to hold. Ragde and Wigderson also pose the question of finding an explicit language (i.e., single output function) that witnesses the difference between constant-depth circuits with $O(n)$ wires and $O(n)$ gates. In this paper we present such an explicit example, as well as an example to separate ACC^0 circuits with $O(n)$ wires from those with $O(n)$ gates. (Recall, ACC^0 circuits are constant-depth, polynomial size circuits consisting of unbounded fan-in AND, OR and modular-counting gates.)

Our main technical contribution is a novel method to analyze communication in boolean circuits. That allows us to prove lower bounds by using previous results dealing with super-concentrators and derived graphs [7, 9].

Our initial goal in our investigation was to study the circuit complexity of regular languages. This is a meaningful task as this class is rich enough to provide examples of complete problems for NC^1 and most of its interesting subclasses. We are successful in that respect and we are able to give an explicit characterization of those regular languages that can be recognized by AC^0 circuits with $O(n)$ wires. This question is delicate since every regular language that can be recognized in AC^0 can actually be recognized by AC^0 circuits with $O(ng^{-1}(n))$ wires, where g^{-1} is the inverse of any unbounded primitive recursive function [6]. Our technique is powerful enough to also yield a characterization of those regular languages in ACC^0 that can be computed with $O(n)$ wires. (Interestingly, the classes of regular languages that are *easy* to compute by bounded-depth circuits are the same classes that have good algorithmic properties in other contexts as well, such as in the context of communication complexity.) In particular, we obtain regular languages that are computable by ACC^0-circuits with $O(n)$ gates but not $O(n)$ wires. We do not know (and we do not believe) that such languages can be found for AC^0.

The paper is organized as follows. In Section 2 we introduce notation and present some (easy) preliminary results. In Section 3 we analyze communication in constant-depth

[*]Partially supported by NSERC and FQRNT.

[†]Partially supported by grant IAA1019401 of the AV ČR and project No. 1M0021620808 of the Ministry of Education of the Czech Republic.

circuits and prove our main lower bounds. Section 4 is devoted to our complete analysis of wire complexity of regular languages.

2. LINEAR SIZE CIRCUITS

2.1 Preliminaries

We use similar notation as [10]. We denote by AC^0 the class of functions computable by families of constant-depth polynomial size circuits consisting of unbounded fan-in AND, OR and unary NOT gates. We call such circuits AC^0 circuits. For an integer $q \geq 1$, if the circuits use unbounded fan-in MOD-q gates in addition to AND, OR, NOT gates, we obtain a class of functions denoted by $AC^0[q]$; we call these circuits $AC^0[q]$ circuits. (A MOD-q gate is a binary gate that evaluates to true iff the number of its inputs that are set to true is divisible by q.) Furthermore, $ACC^0 = \bigcup_q AC^0[q]$. We denote by LC^0 the subclass of AC^0 functions that are computable by AC^0 circuits consisting of *linear number of gates*. Furthermore, WLC^0 denotes the subclass of LC^0 functions that are computable by AC^0 circuits containing *linear number of wires* (gate connections). Similarly for $LC^0[q], WLC^0[q], LCC^0$ and $WLCC^0$.

Beside languages over binary alphabet we consider also languages over arbitrary alphabet Σ. We use the following convention for inputs to boolean circuits. An input $w_1w_2\cdots w_n \in \Sigma^n$ is presented to a boolean circuit C as follows: C has an input gate $Q_{\sigma,i}$, for every $\sigma \in \Sigma$ and $i \in \{1,\ldots,n\}$, where $Q_{\sigma,i}$ is set to one iff $w_i = \sigma$. One should note that if a different convention of input encoding were chosen our bounds remain essentially unchanged.

2.2 Main results

In this section we shall state our separation results and present explicit functions that show that bounded-depth circuits with linear number of gates can compute more than bounded-depth circuits with linear number of wires. Our main separation results are:

THEOREM 1. *There are explicitly defined functions that witness the strictness of the following inclusions:*

1. $WLC^0 \subsetneq LC^0$.
2. *For any* $q \geq 2$, $WLC^0[q] \subsetneq LC^0[q]$.
3. $WLCC^0 \subsetneq LCC^0$.

We are not aware of it being previously stated that circuits with linear number of gates are strictly more powerful than circuits with linear number of wires. There is an easy counting argument to show that, though. For completeness we present the argument here. We show that there are functions computable by depth-2 circuits with $n+1$ gates that are not computable by any kind of circuits with $O(n)$ (or even with $O(n^2/\log n)$) wires: Consider a DNF formula F consisting of exactly n conjunctions of $r(n)$ positive literals. There are $\binom{n}{r(n)} = t(n)$ possible conjunctions of $r(n)$ positive literals on n variables. Since every conjunction corresponds to one *minterm* of F, different formulas give different functions. Overall, there are $\binom{t(n)}{n}$ such formulas which is $n^{\Omega(r(n)n)}$, for $r(n) < n^{1-\epsilon}$. Choosing $r(n)$ to be any slowly growing function, we have more DNF formulas computable by circuits with $n+1$ gates than there are possible functions computed by circuits with $O(n)$ wires, which is $n^{O(n)}$.

By the same argument one can show that for any function $m(n) \leq 2^{o(n)}$, there are functions computable by monotone depth-2 circuits with $m(n)$ gates which are not computable by any circuit with $O(m(n))$ wires.

A similar type of diagonalization argument was presented to us by Harry Buhrman, who showed that circuits with $O(n^{2k})$ gates are strictly more powerful than circuits with $O(n^k)$ gates, [4].

Unfortunately, the preceding argument does not reveal any information about the functions that witness the separation. We do present such functions here. Consider the following regular language $K = (c^*ac^*b)^*c^*$. As we shall see in this and next sections the characteristic function of this language is not computable by ACC^0 circuits with linear number of wires, however it is computable by $AC^0[q]$ circuits with linear number of gates for any $q \geq 2$. This gives an explicit function in $LC^0[q] \setminus LCC^0$.

We conjecture that K is not in LC^0, though we are not able to prove it yet. Thus in order to separate LC^0 from WLC^0 we shall use a different function. Let $n = 2^m$. We define a function $P_n : \{0,1\}^{4n+m} \rightarrow \{0,1\}$ on variables $x_0,\ldots,x_{n-1}$, $y_0,\ldots,y_{n-1}$, $u_0,\ldots,u_{n-1}$, $v_0,\ldots,v_{n-1}$, $w_0,\ldots,w_m$ by the following formula:

$$\bigvee_{t=0}^{m}\left(w_t \wedge \bigwedge_{i=0}^{2^t-1}\left(\bigvee_{j=i2^{m-t}}^{(i+1)2^{m-t}-1}(x_j \wedge u_j) \equiv \bigvee_{j=i2^{m-t}}^{(i+1)2^{m-t}-1}(y_j \wedge v_j)\right)\right).$$

One should interpret the formula as follows. Variables $u_0,\ldots,u_{n-1}$ and $v_0,\ldots,v_{n-1}$ should be thought of as a mask for variables $x_0,\ldots,x_{n-1}$ and $y_0,\ldots,y_{n-1}$, respectively. Variables $w_0,\ldots,w_m$ are choosing a size of blocks in which remaining variables are to be partitioned: setting w_t to one partitions $x_0,\ldots,x_{n-1}$ and $y_0,\ldots,y_{n-1}$ into blocks of size 2^t. For each block exactly one variable u_j and $v_{j'}$ is set to one. Once we fix setting to w_t, v_j and $v_{j'}$, the formula compares x_j against $y_{j'}$, in every block.

THEOREM 2.

- *For every* $q \geq 2$, $K \in LC^0[q] \setminus WLCC^0$.
- *Function* $P_n \in LC^0 \setminus WLCC^0$.

We first prove the simple upper bounds. The next section is devoted to proving the lower bounds.

PROPOSITION 3.

1. *For every* $q \geq 2$, K *is in* $LC^0[q]$.
2. $P_n \in LC^0$.

PROOF. 1. For a word $w = w_1w_2\cdots w_n$ and arbitrary symbol σ, let $\nu_\sigma(w) = |\{i;\ w_i = \sigma\}|$. It is straightforward to verify that for every word $w \in \{a,b,c\}^*$, $w \in K$ iff for all $i \in \{1,\ldots,n\}$, $x_i = a$ implies $\nu_a(x_1x_2\cdots x_i) - 1 \equiv \nu_b(x_1x_2\cdots x_i) \pmod q$ and $x_i = b$ implies $\nu_a(x_1x_2\cdots x_i) \equiv \nu_b(x_1x_2\cdots x_i) \pmod q$. Using MOD-$q$ gates one can express this condition by a constant depth circuit with $O(n)$ gates.

2. To compute the function P_n by a circuit with a linear number of gates and bounded-depth we can use essentially its formula above. The only thing that we have to change

is to compute each term $x_j \wedge u_j$ and $y_j \wedge v_j$ only once and replace the equivalence by a (constant size) AC^0 circuit. Then each circuit

$$\left(\bigvee_{j=i2^{m-t}}^{(i+1)2^{m-t}-1} (x_j \wedge u_j) \equiv \bigvee_{j=i2^{m-t}}^{(i+1)2^{m-t}-1} (y_j \wedge v_j)\right)$$

has a constant number of gates, if we do not count the terms $x_j \wedge u_j$ and $y_j \wedge v_j$ and the number of these circuits is $\sum_{t=0}^{m} 2^t = 2n - 1$. Thus these circuits have only a linear number of gates. Then we have m gates for the part $w_t \wedge \bigwedge_{i=0}^{2^t-1}$ and one $\bigvee$ as the output. Hence the number of gates is linear. $\square$

3. LOWER BOUNDS

Our proofs greatly extend the technique of [5] who proved lower bounds on the number of wires in *multi-output* circuits. Our lower bounds are based upon new relationship between communication complexity of *boolean* functions and *"bottlenecks"* in boolean circuits computing these functions. We show that if a boolean function has a large communication complexity then there cannot be a bottleneck in any circuit computing the function. Using known lower bounds on the number of edges in graphs with no bottlenecks (graphs related to super-concentrators) we obtain our lower bounds. In the next subsection we introduce our tools and in the subsequent subsection we prove the new lower bounds.

3.1 Communication in circuits

We consider the usual two-party communication model for computing functions. Let $f : \mathbf{X} \times \mathbf{Y} \to \{0, 1\}$ be a boolean function with finite domains $\mathbf{X}$ and $\mathbf{Y}$. There are two parties—Alice and Bob—holding input $x \in \mathbf{X}$ and $y \in \mathbf{Y}$, respectively. Alice and Bob try to compute the value of $f(x, y)$. They do it by exchanging messages according to some fixed *protocol*. Eventually, they both learn the value of $f(x, y)$. For a pair of strings $(x, y) \in \mathbf{X} \times \mathbf{Y}$ we measure how many bits Alice and Bob communicate to compute the value of $f(x, y)$. The *cost of the protocol* is the maximum over all inputs $(x, y) \in \mathbf{X} \times \mathbf{Y}$ of the overall number of bits communicated between Alice and Bob while computing $f(x, y)$ according to the protocol. The *communication complexity* $CC(f)$ *of a function* f is the cost of the best protocol for f (a protocol for f of the minimal cost). We refer the reader to [8] for further details. For a function $f : \Sigma^n \to \{0, 1\}$, the communication complexity of f shall be the maximum communication complexity of f for any possible partitioning of the input into two (possibly interleaving) parts $\mathbf{X}$ and $\mathbf{Y}$.

We introduce the following notation. For a directed acyclic graph G, a *path in* G *of length* m is a sequence $v_0, v_1, \ldots, v_m$ of vertices of G such that for all $0 \le i < m$, (v_i, v_{i+1}) is an edge of G. The *depth of* G is the length of the longest path in G. Let X, Y be two disjoint sets of vertices of in-degree zero in G. An (X, Y)*-path in* G is a pair of paths (p_1, p_2) such that path p_1 starts in X, path p_2 starts in Y and p_1 and p_2 intersect in a unique vertex r which is the last vertex of both paths. The vertex r will be called *the root* of (p_1, p_2). Think of an (X, Y)-path as a path in which the edges, starting from X, are directed forward until the root and since then on they are directed backward until Y. An (X, Y)*-cut in* G is any subset S of vertices of G such that removing S together with the adjacent edges from G gives a graph with no (X, Y)-path. For a circuit C, we denote by $G(C)$ the underlying directed acyclic graph of C, i.e., $G(C) = (V, A)$, where V is the set of gates of C and $(u, v) \in A$ iff gate u is an input to gate v. We often identify C with $G(C)$, e.g., the *depth of* C is the depth of $G(C)$.

We state the following lemma that relates the deterministic communication complexity of a boolean function f to the size of an (X, Y)-cut in circuits C computing f.

LEMMA 4. *Let $n, c \ge 1$ and $q \ge 2$ be integers. Let $\mathbf{X} = \mathbf{Y} = \Sigma^n$ be sets and $f : \mathbf{X} \times \mathbf{Y} \to \{0, 1\}$ be a boolean function. Let C be a boolean circuit computing f consisting of arbitrary gates, all of which have communication complexity at most c. Let X and Y be the sets of the input gates of C that correspond to the parts $\mathbf{X}$ and $\mathbf{Y}$ of the input, respectively. If there is an (X, Y)-cut S of size k in C, then $CC(f) \le kc + 1$.*

In particular, if C is a circuit consisting of unbounded fan-in AND, OR, MOD-*q and unary* NOT *gates, then $CC(f) \le 2k\lceil \log_2 q \rceil + 1$.*

PROOF. Consider a circuit C that computes f. Let S be an (X, Y)-cut in C of size k. We say that a gate g of C is in the *component of* X if it is reachable by a directed path in $G(C)$ from some gate in X without going through any gate in S. Similarly for g in the *component of* Y. Note that g cannot be in the component of X and Y simultaneously.

We define a protocol for Alice and Bob that proceeds as follows. Alice and Bob will evaluate C on (x, y) while exchanging $c|S|$ bits of communication. Alice will evaluate all gates in C that are not in the component of Y and Bob will evaluate all gates in C that are not in the component of X. Let the gates $g_1, g_2, \ldots, g_m$ of circuit C be ordered so that input to the gate g_i is only from gates g_j, where $j < i$. Since C is acyclic such an ordering is possible.

Alice and Bob will evaluate gates in order of increasing i. Whenever they reach a gate $g \in S$ they invoke the communication protocol of length c for computing its value. Observe, gates not in the component of Y except for gates in S, receive values only from other gates not in the component of Y. Similarly for X. Hence, by induction on i one can see that Alice and Bob are able to evaluate all the gates that are not in the component of their partners, provided that they evaluate gates in S together.

In particular, the output gate of C will be evaluated by Alice or Bob. By exchanging an additional bit of information both of them may learn the value of the output gate. Overall number of bits exchanged is thus $c|S| + 1 = ck + 1$. The special case for AND, OR, NOT and MOD-q gates follows from: $CC(\text{AND}) = CC(\text{OR}) = 2$, $CC(\text{NOT}) = 1$, $CC(\text{MOD}-\text{q}) = 1 + \lceil \log_2 q \rceil$. $\square$

A partial converse of our lemma is also true. If a function f has small communication complexity then there is a circuit computing f that has small (X, Y)-cut.

We will typically use the previous lemma in situations where part of the input to C is fixed. The following corollary is an immediate consequence of the lemma.

COROLLARY 5. *Let $n, m \ge 1$ and $q \ge 2$ be integers. Let $\mathbf{X} = \mathbf{Y} = \Sigma^n$, $\mathbf{Z} = \Sigma^m$ be sets and $f : \mathbf{X} \times \mathbf{Y} \times \mathbf{Z} \to \{0, 1\}$ be a boolean function. Let C be a boolean circuit computing f consisting of unbounded fan-in* AND, OR, MOD-*q and unary* NOT *gates. Let X and Y be the sets of the input gates* *of C that correspond to the parts $\mathbf{X}$ and $\mathbf{Y}$ of the input, respectively. Let $z \in \mathbf{Z}$, $\mathbf{X}' \subseteq \mathbf{X}$, $\mathbf{Y}' \subseteq \mathbf{Y}$ and $f' : \mathbf{X}' \times \mathbf{Y}' \to \{0,1\}$ be such that $f'(x,y) = f(x,y,z)$. If $CC(f') \geq k'+1$ then every (X,Y)-cut in C has size at least $k'/2\lceil \log_2 q \rceil$.*

That is the base for our circuit lower bounds. In general, we could obtain non-linear circuit lower bounds for bounded-depth circuits which contain also a small number of gates of non-constant communication complexity, e.g., we could allow $o(n/\log n)$ gates computing symmetric functions such as MAJORITY.

We need to establish the following lemma, a variant of Menger's theorem. Our original version of this lemma depended on the depth. We include a stronger version which is due to Jiří Sgall [**?**]. (The fact that the estimate is independent of the depth is interesting, but it has almost no influence on our results.)

LEMMA 6. *Let G be a directed acyclic graph. Let X, Y be two disjoint sets of vertices of in-degree zero in G. If every (X,Y)-cut has size at least k, then there exists a family of $k/2$ vertex-disjoint (X,Y)-paths in G.*

PROOF. Let ℓ be the maximal number such that there exist ℓ vertex disjoint (X,Y)-paths in G. We shall show that there exists a (X,Y)-cut of size at most 2ℓ.

Let $(p_1,q_1),\ldots,(p_\ell,q_\ell)$ be such a system of paths, let $r_1,\ldots,r_\ell$ be the roots of these paths. We can moreover assume that the set $\{r_1,\ldots,r_\ell\}$ satisfies the following maximality condition. It is not possible to find another system of paths with the same roots except for one, say r_i', which is before r_i, which means that there exists a path of length at least one from r_i' to r_i.

We say that p is an *augmenting X-path,* if it starts in some vertex of X not used by $p_1,\ldots,p_\ell$, then it goes forward until it meets a vertex on some path p_i. When this happen the path has to go at least one edge backwards, but it can also go more edges backwards along p_i. Then it can again go forward to a vertex not on p_i and continue until it possibly meets another path p_j and so on. Augmenting Y-paths are defined in the same way.

First we prove that an augmenting X-path can never get to a vertex on q_i different from the root r_i. Suppose there exists an augmenting path p containing a vertex v from q_i, $v \neq r_i$. W.l.o.g. we can assume that v is the last vertex of p and p does not contain any other vertex with the above property. Then we can switch the paths $p_1,\ldots,p_\ell$ and p so that we obtain $\ell+1$ vertex disjoint X-paths ending at $r_1,\ldots,r_\ell$ and v. Now we can take those that end at r_j for $j \neq i$ and the one ending at v and combine them with the Y-paths $q_1,\ldots,q_\ell$. This would violate the maximality condition that the roots $r_1,\ldots,r_\ell$ satisfy.

Now we show that an augmenting X-path and an augmenting Y-path can never meet except at some vertices r_i. Suppose there is such a pair p,q. We may assume w.l.o.g. that their root r is the unique common vertex that does not belong to the set $\{r_1,\ldots,r_\ell\}$. We also know that p does not contain any vertex of q_i except possibly for r_i and symmetrically q. Thus if we switch paths $p_1,\ldots,p_\ell,p$ and paths $q_1,\ldots,q_\ell,q$, we get $\ell+1$ vertex disjoint (X,Y)-paths.

Let u_i (v_i) be the last vertex reachable by an augmenting X-path (by an augmenting Y-path, respectively), if there is any. If no vertex of u_i (v_i) is reachable by an augmenting X-path (by an augmenting Y-path, respectively), we define u_i (v_i) to be the first vertex of p_i (q_i respectively). It is possible that, for some i, $u_i = r_i$, or $v_i = r_i$ or both. We claim that $\{u_1,\ldots,u_\ell,v_1,\ldots,v_\ell\}$ is an (X,Y)-cut.

Let (p,q) be an (X,Y)-path. We shall show that either p contains some u_i or q contains some v_i. To this end it suffices to show that if an X-path (Y-path) does not contain any vertex u_i, then every vertex of this path is reachable by an augmenting X-path (Y-path, respectively). Indeed, if an (X,Y)-path (p,q) does not meet $\{u_1,\ldots,u_\ell,v_1,\ldots,v_\ell\}$, then, in particular, its root r must be reachable by both an augmenting X-path and an augmenting Y-path. But this is possible only if $r \in \{r_1,\ldots,r_\ell\}$, in which case r does belong to the cut contrary to the assumption.

By symmetry it suffices only to consider an X-path p that does not contain any u_i. We shall use induction. Let w_0 be the first vertex of p. If w_0 is outside of all paths $p_1,\ldots,p_\ell$, then it is reachable by the trivial one-vertex augmenting X-path. If it is on some path p_i, then u_i must be after w_0. Hence we can take an augmenting path going to u_i and then backwards to w_0.

Now assume all vertices on p until a vertex w are reachable by augmenting paths. Let w' be the next vertex on p. If w' is outside of all paths $p_1,\ldots,p_\ell$, then we can take an augmenting path to w and continue to w'. Otherwise we argue in the same way as in the induction basis. $\square$

The last tool that we will use is provided to us in [9] and it is related to super-concentrators. Let G be a directed acyclic graph with two disjoint sets IN and OUT of distinguished vertices, each of size n; vertices in set IN have in-degree zero and vertices in set OUT have out-degree zero. Let $IN = \{x_1,\ldots,x_n\}$ and $OUT = \{y_1,\ldots,y_n\}$. Let $0 < \epsilon,\delta \leq 1$. We say that G *satisfies the ϵ,δ–DR property,* if for every $1 \leq k \leq n$, there are non-empty sets of k element subsets $\mathcal{X}_k, \mathcal{Y}_k \subseteq \{Z;\ Z \subseteq \{1,\ldots,n\}\ \&\ |Z| = k\}$ such that for every $i = 1,\ldots,n$, $\delta\Pr[i \in X] \leq k/n$ and $\delta\Pr[i \in Y] \leq k/n$, where $X \in \mathcal{X}_k$ and $Y \in \mathcal{Y}_k$ are chosen with certain probability distributions. Furthermore, we require that for random $X \in \mathcal{X}_k$ and $Y \in \mathcal{Y}_k$, the mean number of vertex disjoint paths between sets $\{x_i, i \in X\}$ and $\{y_i, i \in Y\}$ is at least ϵk. (This concept was introduced in [9], where the property had one more parameter η. Since the parameter η would always be equal to 0 in this paper, we omit it.)

Also the following definition will be needed: for a non-negative function $f(n)$, where $f(n) < n$ for $n \geq 1$, define $f^*(n) = \min\{i;\ f^{(i)}(n) \leq 1\}$, where $f^{(i)}(\cdot)$ denotes $f(\cdot)$ iterated i-times. Furthermore, let $\lambda_1 = \lceil \log_2 n \rceil$ and for $i \geq 1$, $\lambda_{i+1} = \lambda_i^*$.

THEOREM 7 ([9], THEOREM 3). *Let $\epsilon,\delta > 0$ be reals, $d \geq 2$ be an integer. There exists a constant $c > 0$ such that for all integers $n \geq 1$ the following holds. If G is a directed acyclic graph of depth $2d$ with two disjoint sets IN and OUT of n vertices that satisfies the ϵ,δ–DR property, then G has at least $cn\,\lambda_d(n)$ edges.*

For a boolean circuit C of depth d, we cannot apply Theorem 7 directly to the graph of the circuit $G(C) = (V,A)$ since there is only one node of out-degree zero. The intuition is that we would like IN and OUT both to correspond to sets of input gates (while maintaining the condition that vertices OUT have out-degree zero); to achieve this we shall use the following auxiliary graph $\widetilde{G}(C) = (\widetilde{V},\widetilde{A})$, where $\widetilde{V} =$

$\{0,\ldots,2d\}\times V$ and $((i,u),(j,v))\in\widetilde{A}$ iff $1\le i+1=j\le 2d$ and either $u=v$ or $(u,v)\in A$ or $(v,u)\in A$. This graph has depth $2d$, the number of edges in $\widetilde{G}(C)$ is bounded by a linear function of the number of edges of $G(C)$, and disjoint paths in $G(C)$ of length at most $2d$ in which the edges are oriented arbitrarily translate into disjoint directed paths in $\widetilde{G}(C)$. In particular, if $X,Y\subseteq V$ and there are k vertex disjoint (X,Y)-paths, then there are k vertex disjoint directed paths from $\{0\}\times X$ to $\{2d\}\times Y$.

3.2 Putting things together

We will first prove that K is not in WLC^0.

LEMMA 8. *Let $q\ge 2$ and $d\ge 1$ be integers. There exists a constant $c>0$ such that if C_{3n} is an $AC^0[q]$ circuit of depth d computing $K^{=3n}$ then the number of wires in C_{3n} is at least $c\,n\,\lambda_{d+1}(n)$.*

PROOF. Consider a directed graph $G=(V,A)$ constructed from $\widetilde{G}(C_{3n})=(\widetilde{V},\widetilde{A})$ as follows: $V=\widetilde{V}\cup\{x_1,x_2,\ldots,x_n,y_1,\ldots,y_n\}$ and $A=\widetilde{A}\cup\{(x_i,(0,Q_{\sigma,3i-2})),((2d,Q_{\sigma,3i-1}),y_i);\ \sigma\in\Sigma,\ 1\le i\le n\}$. Set $IN=\{x_1,x_2,\ldots,x_n\}$ and $OUT=\{y_1,y_2,\ldots,y_n\}$. We claim that G satisfies the $\epsilon',1/2$-DR property and hence it has many edges by Theorem 7.

LEMMA 9. *G has the $\epsilon',1/2$-DR property, for some $\epsilon'>0$.*

PROOF. For every $1\le k\le n$ we define $\mathcal{X}_k$ and $\mathcal{Y}_k$ as follows. Let $l=\lfloor\frac{n}{k}\rfloor$. We split $\{1,2,\ldots,n\}$ into k *regular blocks* $B_{k,i}=\{il+1,il+2,\ldots,il+l\}$, for $0\le i<k$, and one *left-over block* $B_k=\{kl+1,\ldots,n\}$ of size at most $n/2$. For $0<s,r\le l$, let $X_{k,s}=\{s+li;\ 0\le i<k\}$ and $Y_{k,r}=\{r+li;\ 0\le i<k\}$, i.e., every $X_{k,s}$ and $Y_{k,r}$ contain one element from every regular block. Now, $\mathcal{X}_k=\{X_{k,s};\ 0<s\le l\}$, $\mathcal{Y}_k=\{Y_{k,r};\ 0<r\le l\}$, and we pick distribution on $\mathcal{X}_k$ and $\mathcal{Y}_k$ to be uniform. Clearly, for $1<i\le n$, $1/2\cdot\Pr[i\in X]\le k/n$ and $1/2\cdot\Pr[i\in Y]\le k/n$, where $X\in\mathcal{X}_K$ and $Y\in\mathcal{Y}_k$ are chosen at random, since except elements of the left-over block B_k, every i appears in exactly one $X_{k,s}$ and $Y_{k,r}$.

Pick any $k\in\{1,\ldots,n\}$ and consider arbitrary sets $X\in\mathcal{X}_k$ and $Y\in\mathcal{Y}_k$. We want to show that there are at least $\epsilon' k$ vertex disjoint paths between $\{x_i;\ i\in X\}$ and $\{y_i;\ i\in Y\}$ in G, for some $\epsilon'>0$. We will show that there are no small (X,Y)-cuts between the input gates corresponding to X and Y in C_{3n} which will imply the existence of many vertex disjoint (X,Y)-paths between these gates in C_{3n}. By the remark following the definition of $\widetilde{G}(C)$, we will conclude the lemma.

Since input gates in C_{3n} are connected to the output gate via paths of length at most d, there is always at least one path between $\{x_i;\ i\in X\}$ and $\{y_i;\ i\in Y\}$ in G. Let $X'=\{3i-2;\ i\in X\}$ and $Y'=\{3i-1;\ i\in Y\}$. Let $Q_X=\{Q_{\sigma,j};\ \sigma\in\Sigma,\ j\in X'\}$ and $Q_Y=\{Q_{\sigma,j};\ \sigma\in\Sigma,\ j\in Y'\}$.

CLAIM 10. *Every (Q_X,Q_Y)-cut in $G(C_{3n})$ is of size at least $k/2\lceil\log_2 q\rceil$.*

We use Corollary 5 to establish the claim. Consider the following type of inputs $w\in\Sigma^{3n}$ to circuit C_{3n}. $w=w_1w_2\cdots w_{3n}$, where $w_j=b$ for $j\in Z'=\{3i\lfloor\frac{n}{k}\rfloor;\ 1\le i\le k\}$, $w_j=c$ for $j\in\{1,\ldots,3n\}-(X'\cup Y'\cup Z')$, and $w_j\in\{a,c\}$ for $j\in X'\cup Y'$. Notice, between every pair of b's there is one position $j\in X'$ and one position $j'\in Y'$. C_{3n} evaluates to 1 on w iff $w_j\ne w_{j'}$, for every pair $(j,j')\in X'\times Y'$, where $\lfloor j/3\lfloor n/k\rfloor\rfloor=\lfloor j'/3\lfloor n/k\rfloor\rfloor$. Hence, C_{3n} restricted to inputs w computes a bit-wise complement of parts X' and Y' of the input, i.e., a function $f(x,y)$ such that $f(x,y)=1$ iff $x_i\ne y_i$ for all $i\in\{1,\ldots,k\}$. This function essentially corresponds to Equality Function, hence $CC^1(f)=k+1$. Now, the claim follows by Corollary 5.

By Lemma 6, there are at least $k/(4\lceil\log_2 q\rceil)$ vertex disjoint (Q_X,Q_Y)-paths in C_{3n}. By the remark following the definition of $\widetilde{G}(C)$, we know that there are at least $k/(4\lceil\log_2 q\rceil)$ vertex disjoint paths between $\{0\}\times Q_X$ and $\{2d\}\times Q_Y$ in $\widetilde{G}(C)$. Finally, we set $\epsilon'=1/(20\lceil\log_2 q\rceil)$. It is immediate that there are $\epsilon' k$ vertex disjoint paths between $\{x_i;\ i\in X\}$ and $\{y_j;\ j\in Y\}$ in G, hence the lemma follows. □

Lemma 9 establishes that G has the $\epsilon',1/2$-DR property, for some ϵ'. By Theorem 7, G has at least $c\,n\,\lambda_{d+1}(n)$ wires, for some $c>0$. Observe, $|A|=2d|A(C_{3n})|+2d|V(C_{3n})|+4n$ and $|V(C_{3n})|\le|A(C_{3n})|+1$. Hence, the number of wires in C_{3n} must be at least $c'\,n\,\lambda_{d+1}(n)$, for some $c'>0$.

LEMMA 11. *Let $q\ge 2$, $n,m,d\ge 1$ be integers, where $n=2^m$. If C_{4n+m} is an $AC^0[q]$ circuit of depth d computing P_n then the number of wires in C_{4n+m} is at least $\Omega(n\,\lambda_{d+1}(n))$.*

PROOF. We shall show now that every bounded-depth circuit computing function P_n has a nonlinear number of wires. The proof is essentially the same as the proof of Lemma 8. For the reader's convenience we shall give a complete proof.

We shall use the following property of this function:

For every t, $0\le t\le m$, every $a_0,\ldots,a_{2^t-1}$, $i2^{m-t}\le a_i<(i+1)2^{m-t}$ and every $b_0,\ldots,b_{2^t-1}$, $i2^{m-t}\le b_i<(i+1)2^{m-t}$, we can fix all variables except of $x_{a_0},\ldots,x_{a_{2^t-1}}$ and $y_{b_0},\ldots,y_{b_{2^t-1}}$, so that the restricted function computes $\bigwedge_{i=0}^{2^t-1}x_{a_i}\equiv y_{b_i}$.

It is clear from the definition of the function that it satisfies this condition.

Let C be a depth d circuit computing the function. Let t, $0\le t\le m$, be fixed, and let us denote by $k=2^t$. If $a_0,\ldots,a_{2^t-1}$ and $b_0,\ldots,b_{2^t-1}$ are as above, then, by Corollary 5, every $(\{x_{a_0},\ldots,x_{a_{2^t-1}}\},\{y_{b_0},\ldots,y_{b_{2^t-1}}\})$-cut has at least $k/2$ vertices. By Lemma 6, there are at least $k/4$ vertex disjoint paths connecting the sets $\{x_{a_0},\ldots,x_{a_{2^t-1}}\}$ and $\{y_{b_0},\ldots,y_{b_{2^t-1}}\}$.

Now we would like to derive an ϵ,δ–DR property. What may seem an obstacle is that we can guarantee such paths only for k which is a power of 2. But if k is not of this form, we can take the largest t such that $2^t<k$, ignore subsets of size $k-2^t$ vertices and the number of paths will still be linear in k.

More precisely, we shall show that the graph of the circuit C has the $\epsilon,1$–DR property with $\epsilon=1/8$. To this end we need to define probability distributions on subsets of $\{x_1,\ldots,x_n\}$ and $\{y_1,\ldots,y_n\}$ of size k. Suppose $2^t\le k<2^{t+1}$. The probability distribution for $\{x_1,\ldots,x_n\}$ is the uniform distribution on all subsets X of $\{x_1,\ldots,x_n\}$ of size k which have the property that for every $i=0,\ldots,2^t-1$, X contains at least one vertex of $\{x_j\ ;\ j=i2^{m-t},\ldots,(i+1)2^{m-t}-1\}$. Then every x_i occurs in X with equal probability, namely k/n. We take the same distribution on the subsets of $\{y_1,\ldots,y_n\}$. This shows that $\delta=1$. Finally, we have at least $2^t/4 > k/8$ paths between every pair of such sets, which shows $\epsilon = 1/8$. □

This concludes the proof of Theorem 1.

4. REGULAR LANGUAGES

In this section we provide a precise characterization of regular languages that are computable in WLC^0 and $WLCC^0$. Before stating our results we need some more definitions.

A *monoid* M is a set together with an associative operation that contains a distinguished identity element 1_M. We denote the operation multiplicatively, e.g., for all $m \in M$, $m1_M = 1_M m = m$. Except for the free monoids Σ^*, all the monoids that we consider in this paper are finite. An element $m \in M$ is an *idempotent* if $m = m^2$. Since M is finite, there exists an integer $\omega \geq 1$, the *exponent of* M, such that for every $m \in M$, m^ω is an idempotent. A *group* G is a monoid where for every element $a \in G$ there is an *inverse* $b \in G$ such that $ab = ba = 1_G$. A monoid M is *group-free* if every group $G \subseteq M$ is of size 1. (A group G in a monoid M does not have to be a subgroup of M, i.e., 1_M may differ from 1_G). For a monoid M and $a \in M$, the *a-word problem over* M is the function $f_a : M^* \to \{0,1\}$ such that $f_a(a_1 a_2 \cdots a_n) = 1$ iff $a_1 a_2 \cdots a_n = a$.

We consider the following classes (*varieties*) of monoids. The variety $\overline{Ab}$ is the class of monoids in which all groups are commutative (*abelian*). The variety DS consists of monoids M that satisfy for every $x, y \in M$, $((xy)^\omega (yx)^\omega (xy)^\omega)^\omega = (xy)^\omega$. The variety $DO \subseteq DS$ is the class of monoids M that satisfy $(xy)^\omega (yx)^\omega (xy)^\omega = (xy)^\omega$. Finally, the variety DA is a subclass of DO and consists of all monoids M, where for every $x, y, z \in M$, $(xyz)^\omega y (xyz)^\omega = (xyz)^\omega$. Note that the group-free monoids in DS are the same as those in DO, and these are precisely those in DA. There is a rich theory of classification of finite monoids, and all these varieties have many other characterizations; see e.g.[1] for more background. We will only use a few of their properties which will be summarized in propositions that follow.

For a finite alphabet Σ, we can view Σ^* as a (free) monoid with operation concatenation. For monoids M, N, a function $\phi : N \to M$ is a *morphism* if for all $u, v \in N$, $\phi(uv) = \phi(u)\phi(v)$. We say that $L \subseteq \Sigma^*$ can be *recognized by* M if there exist a morphism $\phi : \Sigma^* \to M$ and a subset $F \subseteq M$ so that $L = \phi^{-1}(F)$. A trivial variant of Kleene's theorem states that a language L is *regular* iff it can be recognized by some finite monoid. For every such L there is a minimal monoid $M(L)$ that recognizes L, which we call the *syntactic monoid of* L. A regular language L is in AC^0 iff the morphism $\phi : \Sigma^* \to M(L)$ has the property that for every integer $t \geq 0$, the image $\phi(\Sigma^t)$ does not contain any non-trivial group [2].

Let $L_0, \ldots, L_k \subseteq \Sigma^*$ and $a_1, \ldots, a_k \in \Sigma$. We say that a product $L_0 a_1 L_1 a_2 \cdots L_k$ is *unambiguous* if for any word $w \in \Sigma^*$ there is at most one factorization of the form $w = w_0 a_1 w_1 \cdots a_k w_k$, with $w_i \in L_i$ for each i. We say that $L \subseteq \Sigma^*$ is an *unambiguous language* if it is the disjoint union of languages that are unambiguous products of the form $\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots \Sigma_k^*$, where $\Sigma_i \subseteq \Sigma$ for each i.

Recall, that $K = (c^*ac^*b)^*c^*$. We also consider the language $U = A^*ac^*aA^*$ over the alphabet $A = \{a, b, c\}$. It can be proven that U and K are not unambiguous. They are closely related: $K = \overline{A^*ac^*aA^*} \cap \overline{A^*bc^*bA^*} \cap c^*aA^* \cap A^*bc^*$. The following facts are standard (see e.g. [12]):

Proposition 12.

- *$DA \subsetneq DO \subsetneq DS$.*
- *If a monoid is not in DS then it can recognize either the language U or K.*
- *A group-free monoid is in DA iff it can recognize neither the language U nor K.*
- *A regular language is unambiguous iff its syntactic monoid is in DA.*

For technical reasons it is easier to deal with languages $L \subseteq \Sigma^*$ that have a neutral letter (i.e., there is a letter $e \in \Sigma$ such that for any u and v in Σ^*, $uv \in L$ iff $uev \in L$). Standard techniques introduced in [2] allow to extend this restricted case to the general situation. We make the following observation.

Proposition 13. *If a regular language from WLC^0 (LC^0, $WLCC^0$,...) has a neutral letter then for any a, the a-word problem over its syntactic monoid is also in WLC^0 (LC^0, $WLCC^0$,...).*

4.1 Main results on regular languages

The main result of this section is the following theorem:

Theorem 14. *Let L be a regular language with neutral letter. The following equivalences hold.*

- *L is in WLC^0 iff L is unambiguous.*
- *L is in $WLCC^0$ iff the syntactic monoid of L is in $DO \cap \overline{Ab}$.*

[3] considers a somewhat related problem of computing simultaneously membership of all prefixes of a word in a regular language. It provides a characterization of regular languages for which this can be computed in WLC^0. These languages form a proper subclass of unambiguous languages.

To prove our theorem we need to show two upper bounds, namely that unambiguous languages are in WLC^0 and that regular languages with syntactic monoids in $DO \cap \overline{Ab}$ are in $WLCC^0$. Further also, we need to show some lower bounds, namely that languages that are not unambiguous and whose syntactic monoids are outside $DO \cap \overline{Ab}$, are not in WLC^0 and $WLCC^0$, respectively.

To establish the upper bound for unambiguous languages we will use the characterization of these languages in terms of "turtle" programs [11]. A *turtle program* T is a sequence of instructions $T = (d_1, a_1), (d_2, a_2), \ldots, (d_k, a_k)$, where $d_j \in \{L, R\}$ and $a_j \in \Sigma$. The program executes on an input $w \in \Sigma^*$ placed between two end-markers not in Σ, by moving a turtle along the input. If $d_1 = R$, the turtle is initially on the left end-marker (i.e., at position 0), if $d_1 = L$, it is on the right end-marker (i.e., position $n+1$). To *execute an instruction* (d, a), the turtle moves in direction d from its current position to the next position that contains a; if there is no occurrence of a in the given direction, the program stops and *rejects*. The *language* $L(T)$ *recognized by* T is the set of strings for which the program executes all instructions without rejecting. We say that L is a *turtle language*.

Schwentick et al. show the following theorem:

Theorem 15 ([11]). *A language L is unambiguous iff L is a boolean combination of turtle languages.*

An immediate consequence of the following lemma is that unambiguous languages are in WLC^0.

LEMMA 16. *If L is a turtle language then $L \in WLC^0$.*

PROOF. Let $(d_1, a_1)(d_2, a_2) \cdots (d_k, a_k)$ be a turtle program for L. For every $n \geq 1$, we will construct an AC^0 circuit C_n of depth $O(k)$ and with $O(kn)$ wires recognizing $L^{=n}$. Recall, that C_n accesses its input $w = w_1 w_2 \cdots w_n \in \Sigma^n$ via gates $Q_{\sigma,i}$, for $\sigma \in \Sigma$ and $n \in \{1, \ldots, n\}$.

The circuit C_n consists of k layers. The j-th layer will determine whether the input is already rejected or not after executing the first j instructions of the program; this information is computed in a boolean variable denoted by o_j. Obviously, o_k will be the output that we wish to compute. In addition the j-th layer will also compute a boolean vector of length $n+2$ denoted by p_j; this vector should be equal to $1^s 0^{n+2-s}$ if $o_j = 1$ and the turtle is at position s after the execution of the first j instructions (if $o_j = 0$ then the value of p_j is irrelevant). We now show, how to compute o_j and p_j from o_{j-1} and p_{j-1} using linear number of wires. The j-th layer computes as follows.

If $d_j = R$

1. $h_i = Q_{a_j,i} \wedge \neg p_{j-1,i-1}$ $\quad 1 \leq i \leq n$
   $h_0 = h_{n+1} = 0$
2. $p'_0 \cdots p'_{n+1} = \text{prefix-OR}(h_0 h_1 \cdots h_{n+1})$
3. $p_{j,i} = \neg p'_i$ $\quad 0 \leq i \leq n+1$
4. $o_j = o_{j-1} \wedge \neg p_{j,n}$

If $d_j = L$

1. $h_i = Q_{a_j,i} \wedge p_{j-1,i}$ $\quad 1 \leq i \leq n$
   $h_0 = h_{n+1} = 0$
2. $p'_0 \cdots p'_{n+1} = \text{suffix-OR}(h_0 h_1 \cdots h_{n+1})$
3. $p_{j,i} = p'_{i+1}$ $\quad 0 \leq i \leq n$
   $p_{j,n+1} = 0$
4. $o_j = o_{j-1} \wedge p_{j,0}$

Here, prefix-OR is the function introduced in [5] which is defined as follows: for $h_0, \ldots, h_{n+1} \in \{0, 1\}$,

$$\text{prefix-OR}(h_0 h_1 \cdots h_{n+1}) = b_0 b_1 \cdots b_{n+1},$$

where $b_i = \bigvee_{i \in \{0,\ldots,i\}} h_i$, and

$$\text{suffix-OR}(h_0 h_1 \cdots h_{n+1}) = b'_0 b'_1 \cdots b'_{n+1},$$

where $b'_i = \bigvee_{i \in \{i,\ldots,n+1\}} h_i$. By [5], prefix-OR as well as suffix-OR are computable by AC^0 circuits with linear number of wires. Hence, every layer of the circuit is computable by an AC^0 circuit with $O(n)$ wires. Thus, C_n is of depth $O(k)$ and has $O(kn)$ wires. □

4.1.1 Easiness of $DO \cap \overline{Ab}$ languages

Languages with syntactic monoids in $DO \cap \overline{Ab}$ can be shown to be disjoint unions of unambiguous products of the form $L_0 a_1 L_1 a_2 \cdots L_k$, where each L_i is recognized by a commutative monoid. An algorithm similar to the one of Lemma 16 can be given to recognize these languages in ACC^0 with linear number of wires. We give a sketch of this argument here.

A *super-turtle program* S over a commutative group G is a sequence of instructions $S = I_1, I_2, \ldots I_k$, where each I_j is either a turtle instruction (d_j, a_j) or a *super-turtle instruction* $(d_j, \Sigma_j, g_j, f_j)$, where $d_j \in \{L, R\}$, $\Sigma_j \subseteq \Sigma$, $g_j \in G$ and $f_j : \Sigma_j \to G$. The program *executes* on an input $w \in \Sigma^*$ placed between two distinguished end-markers by moving a turtle along the input similarly to a turtle program. Each instruction is executed in turn; if the instruction is a turtle instruction then it is executed as usually, if it is a super-turtle instruction (d, Σ', g, f) then it is executed as follows: the turtle moves from its current position (at least one step) in the direction d until it reaches the first position with a symbol *not* in Σ'. The turtle stops at that position which may be a position of an end-marker. Then it verifies that the product of $f(w_i)$ of all the symbols w_i that lie between the new position of the turtle and the position before the beginning of the execution of this instruction is equal to g. (The turtle computes the product of $f(w_i)$ while traveling.) If the product is not equal to g the computation stops and rejects.

The *language $L(S)$ recognized by S* is the set of strings for which the program executes all instructions without rejecting. We say that L is a *super-turtle language* if there is a super-turtle program S over some commutative group G such that $L = L(S)$.

One can establish the following theorem. We include the proof of this statement in the Appendix.

THEOREM 17. *If the syntactic monoid of a language L is in $DO \cap \overline{Ab}$ then L is a boolean combination of super-turtle languages.*

The previous theorem together with Lemma 19 below imply that if a syntactic monoid of L is in $DO \cap \overline{Ab}$ then L is in $WLCC^0$. The following fact will be useful for proving Lemma 19.

PROPOSITION 18. *For any commutative monoid M and any $a \in M$, the a-word problem over M is in $WLC^0[q]$, where q is the exponent of M.*

PROOF. Let M be a finite commutative monoid with exponent q. By commutativity, to determine the product of n elements $m_1, m_2, \ldots, m_n \in M$ we only need to determine for every element $m \in M$ the number of its occurrences among $m_1, m_2, \ldots, m_n$. Since $m^{q+i} = m^{jq+i}$, for any $i \in \{0, \ldots, q-1\}$ and $j \geq 1$, we need to know the exact number of occurrences of m only if the number is less than q. If it is at least q, it suffices to know its remainder when divided by q (together with the fact that it is larger than q).

Technique of e.g. [10] allows one to construct a WLC^0 circuit for exact counting up-to *constant* threshold q. The circuit can also determine if the count is more or equal to q. For each element of M, one MOD-q gate can provide the number of occurrences of that element mod q. Hence, for each element of M we can compute the necessary information in $WLC^0[q]$. After collecting this information for all the elements of M, we can easily check whether $m_1 m_2 \cdots m_n = a$. □

LEMMA 19. *If L is a super-turtle language then $L \in WLCC^0$.*

PROOF. Let $I_1, I_2, \ldots, I_k$ be a super-turtle program for L over some group G. Let q be the exponent of G. For every $n \geq 1$, we will construct an $AC^0[q]$ circuit of depth $O(k)$ with $O(kn)$ wires recognizing $L^{=n}$. Recall that C_n accesses its input $w = w_1 w_2 \cdots w_n$ via gates $Q_{\sigma,i}$, for $\sigma \in \Sigma$ and $i \in \{1, \ldots, n\}$.

The circuit C_n computes in essentially the same way as the circuit in the proof of Lemma 16. The circuit is layered and the layer j computes the vector p_j and the bit o_j. The meaning of these values is the same as in the proof of Lemma 16 and their computation proceeds in the same way as in that proof for all turtle instructions I_j. If I_j is a super-turtle instruction $(d_j, \Sigma_j, g_j, f_j)$, then the computation proceeds as follows:

If $d_j = R$

1. $h_i = \left(\bigvee_{a \notin \Sigma_j} Q_{a,i} \right) \wedge \neg p_{j-1,i-1} \qquad 1 \leq i \leq n$
   $h_0 = 0,\ h_{n+1} = 1$
2. $p'_0 \cdots p'_{n+1} = \text{prefix-O}\textsc{r}(h_0 h_1 \cdots h_{n+1})$
3. $p_{j,i} = \neg p'_i \qquad 0 \leq i \leq n+1$
4. $v_i = \neg p_{j-1,i-1} \wedge p_{j,i} \qquad 1 \leq i \leq n$
   $m_i = f'(v_i, w_i) \qquad 1 \leq i \leq n$
5. $o_j = o_{j-1} \wedge \neg p_{j-1,n} \wedge g_j\text{-word}(m_1 m_2 \cdots m_n)$

If $d_j = L$

1. $h_i = \left(\bigvee_{a \notin \Sigma_j} Q_{a,i} \right) \wedge p_{j-1,i} \qquad 1 \leq i \leq n$
   $h_0 = 1,\ h_{n+1} = 0$
2. $p'_0 \cdots p'_{n+1} = \text{suffix-O}\textsc{r}(h_0 h_1 \cdots h_{n+1})$
3. $p_{j,i} = p'_{i+1} \qquad 0 \leq i \leq n$
   $p_{j,n+1} = 0$
4. $v_i = p_{j-1,i} \wedge \neg p_{j,i-1} \qquad 1 \leq i \leq n$
   $m_i = f'(v_i, w_i) \qquad 1 \leq i \leq n$
5. $o_j = o_{j-1} \wedge p_{j-1,0} \wedge g_j\text{-word}(m_1 m_2 \cdots m_n)$

Here, $f'(v_i, w_i)$ is an encoding of 1_G whenever $v_i = 0$ and is an encoding of $f_j(w_i)$ for the i-th input symbol w_i otherwise. The function $g_j\text{-word}(m_1 m_2 \cdots m_n)$ is a boolean function that evaluates to 1 iff $m_1 m_2 \cdots m_k = g_j$ in G. By Proposition 18, the function $g_j\text{-word}(m_1 m_2 \cdots m_n)$ is computable in $WLC^0[q]$. Thus the overall circuit C_n is of depth $O(k)$ and consists of $O(kn)$ wires. $\square$

4.1.2 Lower bounds for regular languages

To prove our lower bounds on (non)computability of certain languages by small AC^0 and ACC^0 circuits we will use the tools developed in Section 3. In particular, we can establish the following claim:

Lemma 20. *Let M be a monoid and $a \in M$. Let $q \geq 2$, $d \geq 1$ and $n > 0$ be integers. Let the a-word problem over M of size n have communication complexity at least $\Omega(n)$. If C_{2n} is an $AC^0[q]$ circuit of depth d computing a-word problem over M of size $2n$, then C_{2n} has $\Omega(n\lambda_{d+1}(n))$ edges.*

Proof. The proof is similar to the proof of Lemma 8. Consider a directed graph $G = (V, A)$ constructed from $\widetilde{G}(C_{2n}) = (\widetilde{V}, \widetilde{A})$ as follows: $V = \widetilde{V} \cup \{x_1, x_2, \ldots, x_n, y_1, \ldots, y_n\}$ and $A = \widetilde{A} \cup \{(x_i, (0, Q_{\sigma,2i-1})), ((2d, Q_{\sigma,2i}), y_i);\ \sigma \in M,\ 1 \leq i \leq n\}$. Set $IN = \{x_1, x_2, \ldots, x_n\}$ and $OUT = \{y_1, y_2, \ldots, y_n\}$. We claim that G satisfies the $\epsilon', 1/2$-DR property and hence it has many edges by Theorem 7.

Lemma 21. *G has the $\epsilon', 1/2$-DR property, for some $\epsilon' > 0$.*

Proof. For every $1 \leq k \leq n$ we define $\mathcal{X}_k$ and $\mathcal{Y}_k$ as follows. Let $l = \lfloor \frac{n}{k} \rfloor$. We split $\{1, 2, \ldots, n\}$ into k *regular blocks* $B_{k,i} = \{il+1, il+2, \ldots, il+l\}$, for $0 \leq i < k$, and one *left-over block* $B_k = \{kl+1, \ldots, n\}$ of size at most $n/2$. For $0 < s, r \leq l$, let $X_{k,s} = \{s + li;\ 0 \leq i < k\}$ and $Y_{k,r} = \{r + li;\ 0 \leq i < k\}$, i.e., every $X_{k,s}$ and $Y_{k,r}$ contain one element from every regular block. Now, $\mathcal{X}_k = \{X_{k,s};\ 0 < s \leq l\}$, $\mathcal{Y}_k = \{Y_{k,r};\ 0 < r \leq l\}$, and we pick the distribution on $\mathcal{X}_k$ and $\mathcal{Y}_k$ to be uniform. Clearly, for $1 < i \leq n$, $1/2 \cdot \Pr[i \in X] \leq k/n$ and $1/2 \cdot \Pr[i \in Y] \leq k/n$, where $X \in \mathcal{X}_K$ and $Y \in \mathcal{Y}_k$ are chosen at random, since except elements of the left-over block B_k, every i appears in exactly one $X_{k,s}$ and $Y_{k,r}$.

Pick any $k \in \{1, \ldots, n\}$ and consider arbitrary sets $X \in \mathcal{X}_k$ and $Y \in \mathcal{Y}_k$. We want to show that there are at least $\epsilon' k$ vertex disjoint paths between $\{x_i;\ i \in X\}$ and $\{y_i;\ i \in Y\}$ in G, for some $\epsilon' > 0$. By our assumption, there exist $\epsilon > 0$ and $k_0 \geq 0$ such that the communication complexity of the a-word problem over M of size k is at least $\epsilon k + 1$, for all $k > k_0$. If we choose ϵ' so that $\epsilon' \leq 1/k_0$, then for all $k \leq k_0$, there is at least $1 \geq \epsilon' k$ path between $\{x_i;\ i \in X\}$ and $\{y_i;\ i \in Y\}$ in G, since input gates are connected to the output gate of C_{2n} via paths of length at most d. Hence, it remains only to consider the case of $k > k_0$. We will show that in this case there are no small (X, Y)-cuts between the input gates corresponding to X and Y in C_{2n} which will imply the existence of many vertex disjoint (X, Y)-paths between these gates in C_{2n}. By the remark following the definition of $\widetilde{G}(C)$, we will conclude the lemma.

Let $k > k_0$. By our assumption, there is some partition of $\{1, \ldots, k\}$ into two sets I_k and J_k so that for $a_1, a_2, \ldots, a_k \in M$ if Alice and Bob hold $a_i;\ i \in I_k$ and $a_j;\ j \in J_k$, resp., then they have to communicate at least $\epsilon k + 1$ bits in order to decide whether $a_1 a_2 \cdots a_k = a$. Restrict X and Y to blocks of I_k and J_k, respectively, i.e., set $X_I = \{i \in X;\ \exists b \in I_k, i \in B_{k,b}\}$ and $Y_J = \{j \in Y;\ \exists b \in J_k, j \in B_{k,b}\}$. Let $Q_X = \{Q_{\sigma,2i-1};\ \sigma \in M,\ i \in X_I\}$ and $Q_Y = \{Q_{\sigma,2j};\ \sigma \in M,\ j \in Y_J\}$.

Claim 22. *Every (Q_X, Q_Y)-cut in $G(C_{2n})$ is of size at least $\epsilon k / 2\lceil \log_2 q \rceil$.*

We use Corollary 5 to establish the claim. Let e be the neutral element of M. Consider the following type of inputs $w \in M^{2n}$ to circuit C_{2n}: $w = w_1 w_2 \cdots w_{2n}$, where $w_s = e$ for $s \in \{1, \ldots, 2n\} - (\{2i-1;\ i \in X_I\} \cup \{2j;\ j \in Y_J\})$, and $w_s \in M$ for $s \in \{2i-1;\ i \in X_I\} \cup \{2j;\ j \in Y_J\}$. C_{2n} computes the a-word problem of size $2n$, so C_{2n} restricted to these inputs w computes the a-word problem of size k. We can view C_{2n} as computing a function $f : M^{X_I} \times M^{Y_J} \rightarrow \{0, 1\}$ of communication complexity at least $\epsilon k + 1$. The claim follows by Corollary 5.

By Lemma 6, there are at least $\epsilon k / (4\lceil \log_2 q \rceil)$ vertex disjoint (Q_X, Q_Y)-paths in C_{2n}. By the remark following the definition of $\widetilde{G}(C)$, we know that there are at least $\epsilon k / (4\lceil \log_2 q \rceil)$ vertex disjoint paths between $\{0\} \times Q_X$ and $\{2d\} \times Q_Y$ in $\widetilde{G}(C)$. Finally, we set $\epsilon' = \min\{\epsilon / (8|M|\lceil \log_2 q \rceil), 1/k_0\}$. It is immediate that there are $\epsilon' k$ vertex disjoint paths between $\{x_i;\ i \in X_I\}$ and $\{y_j;\ j \in Y_J\}$ in G. Since, $X_I \subseteq X$ and $Y_J \subseteq Y$, the lemma follows. $\square$

Lemma 21 establishes that G has the $\epsilon', 1/2$-DR property, for some ϵ'. By Theorem 7, G has at least $c\, n\, \lambda_{d+1}(n)$ wires, for some $c > 0$. Observe, $|A| = 2d|A(C_{2n})| + 2d|V(C_{2n})| + 4n$ and $|V(C_{2n})| \leq |A(C_{2n})| + 1$. Hence, the number of wires in C_{2n} must be at least $c'\, n\, \lambda_{d+1}(n)$, for some $c' > 0$.

Communication complexity of monoids was previously studied in [12]. They prove the following.

Theorem 23 ([12]). *Let M be a monoid. If any of the following conditions holds:*

1. M contains a non-commutative group, or

2. M is in $DS \setminus DO$,

then there exists $a \in M$ such that for every integer $n \geq 1$, the a-word problem over M of size n has communication complexity $\Omega(n)$.

The previous two statements imply that languages whose syntactic monoid contains a non-commutative group or is in $DS \setminus DO$ are not computable in $WLCC^0$. We have seen in Section 3 that K is not in $WLCC^0$. It is straightforward to verify that language U cannot be in $WLCC^0$, either. By Propositions 12 and 13, all languages whose syntactic monoids are outside of DS are not in $WLCC^0$. Hence, together with the upper bounds we obtain Theorem 14.

5. CONCLUDING REMARKS

The most interesting open problems related to the results of this paper are (1) to decide if ADDITION can be computed by bounded-depth circuits with linear number of gates and (2) to decide if regular languages in AC^0 can be computed by bounded-depth circuits with linear number of gates. The currently known lower bound techniques do not seem to be applicable to these problems. The bounds on "generalized bounded-depth super-concentrators" produce only lower bounds on the number of wires, as the proof of $P_n \in LC^0 \setminus WLCC^0$ shows. The switching lemma and the approximation techniques always give bounds that are much larger than the known upper bounds for these problems. Thus to prove such lower bounds one would need to develop a new proof technique, or at least a completely new way of applying the known techniques.

The two problems seem to be, in fact, very much related. For example we can show that the regular language U reduces to integer ADDITION, using bounded-depth circuits with a linear number of gates. Thus our lower bound showing that $U \notin WLCC^0$ gives (the well known result) that ADDITION can not be computed by such circuits.

The difficulty of proving lower bounds on ADDITION is also apparent from the following observation. While we know that ADDITION cannot be computed by WLC^0 circuits, each single bit of the sum can be computed by such circuits. This can be shown using the WLC^0 circuits for the prefix-OR function.

6. REFERENCES

[1] J. Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, 1994.

[2] D. A. M. Barrington, K. J. Compton, H. Straubing, and D. Thérien. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44(3):478–499, 1992.

[3] G. Bilardi and F. P. Preparata. Characterization of associative operations with prefix circuits of constant depth and linear size. *SIAM J. Computing*, 19(2):246–255, 1990.

[4] H. Buhrman. *Personal communication*. Dagstuhl, 2004.

[5] A. K. Chandra, S. Fortune, and R. J. Lipton. Lower bounds for constant depth circuits for prefix problems. In *Proc. of the 10th Intl. Colloquium on Automata, Languages and Programming*, pages 109–117, 1983.

[6] A. K. Chandra, S. Fortune, and R. J. Lipton. Unbounded fan-in circuits and associative functions. *Journal of Computer and System Sciences*, 30:222–234, 1985.

[7] D. Dolev, C. Dwork, N. Pippenger, and A. Wigderson. Superconcentrators, generalizers and generalized connectors with limited depth (preliminary version). In *Proceedings of STOC*, pages 42–51, 1983.

[8] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[9] P. Pudlák. Communication in bounded depth circuits. *Combinatorica*, 14(2):203–216, 1994.

[10] P. Ragde and A. Wigderson. Linear-size constant-depth polylog-threshold circuits. *Information Processing Letters*, 39:143–146, 1991.

[11] T. Schwentick, D. Thérien, and H. Vollmer. Partially ordered two-way automata: a new characterization of DA. In *Proceedings of DLT*, pages 242–253, 2001.

[12] P. Tesson and D. Thérien. Complete classifications for the communication complexity of regular languages. In *Proceedings of STACS*, pages 62–73, 2003.

APPENDIX

A. PROOF OF THEOREM ??

We include here the proof of Theorem 17. We use techniques of [12, 11] to establish the claim. For a commutative group G, integer $k \geq 0$, [12] defines a congruence $\sim_k^G$ on Σ^* as follows. For a word $u \in \Sigma^*$, we denote by $\alpha(u)$ the set of letters occurring in u. For $a \in \alpha(u)$, the *a-left*, resp. *a-right decomposition* of u is the unique factorization $u = u_0 a u_1$ with $a \notin \alpha(u_0)$, resp. $a \notin \alpha(u_1)$, where $u_0, u_1 \in \Sigma^*$. The congruence $\sim_k^G$ is defined iteratively as follows. First, $x \sim_0^G y$ if and only if $\alpha(x) = \alpha(y)$ and for all morphisms $f : \Sigma^* \to G$, $f(x) = f(y)$. Furthermore, for any $k \geq 0$ and the empty word λ, $\lambda \sim_k^G \lambda$. Next, we let $x \sim_k^G y$ if and only if all the following is satisfied:

1. $x \sim_{k-1}^G y$.
2. $\alpha(x) = \alpha(y)$.
3. For any $a \in \alpha(x)$, if $x = x_0 a x_1$ and $y = y_0 a y_1$ are the a-left decompositions of x and y then $x_0 \sim_k^G y_0$ and $x_1 \sim_{k-1}^G y_1$.
4. For any $a \in \alpha(x)$, if $x = x_0 a x_1$ and $y = y_0 a y_1$ are the a-right decompositions of x and y then $x_0 \sim_{k-1}^G y_0$ and $x_1 \sim_k^G y_1$.

[12] establishes the following theorem.

THEOREM 24 ([12], THEOREM 2). *A syntactic monoid of L is in $DO \cap \overline{Ab}$ iff for some commutative group G and integer $k \geq 0$, L is a union of equivalence classes of $\sim_k^G$.*

By a technique similar to [11] (Lemma 3.3) we will show that every equivalence class of $\sim_k^G$ is a boolean combination of super-turtle languages. That will imply the Theorem 17

Given two super-turtle programs P_1 and P_2 we define $L(P_1, P_2)$ to be the set of words $w \in L(P_1) \cap L(P_2)$ such that the turtle ends the execution of P_1 on w to the left from the position on which it ends when it executes P_2 on w. We do not make a distinction between an instruction I and a program consisting of a single instruction I.

LEMMA 25. *Let P be a super-turtle program and $I = (d, a)$ be a super-turtle instruction. Then both $L(I, P)$ and $L(P, I)$ can be expressed as a boolean combination of super-turtle languages.*

PROOF. Let I^{-1} stand for the *inverse* of I, i.e., the instruction (L, a) if $d = R$ and (R, a) if $d = L$. It is straightforward to verify that if $d = R$ then the following hold for every string w:

$$\begin{array}{rcl} w \in L(I, P) & \iff & w \in L(PI^{-1}). \\ w \in L(P, I) & \iff & w \in L(PI) \cap \overline{L(PII^{-1})}. \end{array}$$

Similarly, if $d = L$ then for every string w:

$$\begin{array}{rcl} w \in L(I, P) & \iff & w \in L(PI) \cap \overline{L(PII^{-1})}. \\ w \in L(P, I) & \iff & w \in L(PI^{-1}). \end{array}$$

□

LEMMA 26. *Let Σ be an alphabet, $k \geq 0$ be an integer and G be a commutative group. If $y \in \Sigma^*$ then $L = \{x \in \Sigma^*;\ x \sim_k^G y\}$ is a boolean combination of super-turtle languages.*

PROOF. We prove the claim by induction on k and the size of $\alpha(y)$. If $k = 0$ the proof is straightforward as

$$\{x \in \Sigma^*;\ x \sim_0^G y\} = \bigcap_{f:\alpha(y)\to G} L((R, \alpha(y), f(y), f)) \cap \bigcap_{a\in\alpha(y)} L((R, a)) \cap \overline{\bigcup_{a\notin\alpha(y)} L((R, a))},$$

where $f(y)$ is the image of y under extension of f into the morphism $f : \alpha(y)^* \to G$.

Let $k > 0$. If $|\alpha(y)| = 0$ the the claim follows as

$$\{x \in \Sigma^*;\ x \sim_k^G y\} = \{\lambda\} = \overline{\bigcup_{a\in\Sigma} L((R, a))}$$

So assume that y is such that $|\alpha(y)| > 0$. Pick $a \in \alpha(y)$. Let y_0ay_1 be the a-left decomposition of y. We show next that the language of words $w = w_0aw_1$, with $w_0 \sim_k^G y_0$ and $w_1 \sim_{k-1}^G y_1$ can be expressed as a boolean combination of super-turtle languages. As the a-right decomposition can be handled analogously, the statement of the lemma follows. $\{x \in \Sigma^*;\ x \sim_k^G y\}$ is just the intersection of all boolean combinations obtained.

First we show how the language $L_0 = \{w \in \Sigma^*;\ w = w_0aw_1\ \&\ w_0 \sim_k^G y_0\}$ can be expressed. By induction $|\alpha(y_0)| < |\alpha(y)|$, so $\{w_0 \in \Sigma^*;\ w_0 \sim_k^G y_0\}$ can be expressed as a boolean combination of super-turtle languages. Let $L(P)$ be a language occurring in this characterization. Let $I = (R, a)$. If the first instruction of P is an *R-instruction*, i.e. instruction that moves the turtle to the right, then

$$w_0 \in L(P) \iff w \in L(P) \cap \bigcap_{P'\sqsubseteq P} L(P', I).$$

Here, $P' \sqsubseteq P$ denotes that P' is a non-empty prefix of P. If the first instruction of P is an *L-instruction*, i.e. instruction that moves the turtle to the left, then

$$w_0 \in L(P) \iff w \in L(IP) \cap \bigcap_{P'\sqsubseteq P} L(IP', I).$$

It is straightforward to verify these equivalences. By Lemma 25 it follows that the languages of the form $L(P', I)$ and $L(IP', I)$ can be expressed as boolean combinations of super-turtle languages.

By replacing each $L(P)$ in the characterization of $\{w_0 \in \Sigma^*;\ w_0 \sim_k^G y_0\}$ with the corresponding right hand expression we obtain a characterization of L_0.

Finally, we show that the language $L_0 = \{w \in \Sigma^*;\ w = w_0aw_1\ \&\ w_1 \sim_{k-1}^G y_1\}$ can be expressed. Again, by induction the equivalence class of y_1 can be expressed as a boolean combination of super-turtle languages. Let $L(P)$ be a language in this characterization. If P starts with a R-instruction then

$$w_1 \in L(P) \iff w \in L(IP) \cap \bigcap_{P'\sqsubseteq P} L(I, IP').$$

If P starts with a L-instruction then

$$w_1 \in L(P) \iff w \in L(P) \cap \bigcap_{P'\sqsubseteq P} L(I, P').$$

Again the languages on the right hand side can be expressed as boolean combinations of turtle languages and we obtain the desired characterization of L_1. This completes the proof of the lemma. □