Preprint, Institute of Mathematics, AS CR, Prague. 2007-12-21

Circuit complexity of regular languages

Michal Koucký*
koucky@math.cas.cz
Mathematical Institute of the Academy of Sciences of Czech Republic
Žitná 25, CZ-115 67 Praha 1, Czech Republic

December 21, 2007

Abstract

We survey our current knowledge of circuit complexity of regular languages and we prove that regular languages that are in AC^0 and ACC^0 are all computable by almost linear size circuits, extending the result of Chandra et. al [5]. As a consequence we obtain that in order to separate ACC^0 from NC^1 it suffices to prove for some $\epsilon > 0$ an $\Omega(n^{1+\epsilon})$ lower bound on the size of ACC^0 circuits computing certain NC^1-complete functions.

Keywords: regular languages, circuit complexity, upper and lower bounds.

1 Introduction

Regular languages and associated finite state automata occupy a prominent position in computer science. They come up in a broad range of applications from text processing to automatic verification. In theoretical computer science they play an important role in understanding computation. The celebrated result of Furst, Saxe and Sipser [7] separates circuit classes by showing that the regular language PARITY is not in AC^0, the class of languages that are computable by bounded-depth polynomial-size circuits consisting of unbounded fan-in And, Or gates and unary Not gates. The result of Barrington [1] shows that there are regular languages that are complete for the class NC^1, the class of languages computable by logarithmic-depth circuits consisting of fan-in two And, Or gates and unary Not gates. Recently in [10], regular languages were shown to separate classes of languages computable by ACC^0 circuits using linear number of gates and using linear number of wires. The ACC^0circuits are similar to AC^0 circuits but in addition they may contain unbounded fan-in Mod-q gates.

There is a rich classification of regular languages based on properties of their *syntactic monoids* (see e.g. [20, 19]). (The syntactic monoid of a regular language is essentially the monoid of transformations of states of the minimal finite state automata for the language. See the next section for precise definitions.) It turns out that there is a close connection

*Partially supported by grants GA ČR 201/07/P276 and 201/05/0124.

between algebraic properties of these monoids and computational complexity of the associated regular languages. In this article we survey our current knowledge of this relationship from the perspective of circuit complexity and we point out the still unresolved open questions. Furthermore, we prove that all regular languages that are in AC^0 and ACC^0 are recognizable by AC^0 and ACC^0 circuits, resp., of almost linear size. As a consequence we obtain that in order to separate ACC^0 from NC^1 it suffices to prove for some $\epsilon > 0$ an $\Omega(n^{1+\epsilon})$ lower bound on the size of ACC^0 circuits computing certain NC^1-complete functions.

2 Preliminaries on monoids

In order to understand regular languages one needs to understand their finite state automata. It turns out that the proper framework for such a study are associated transformation monoids. Hence, to facilitate the algebraic classification of regular languages we need to recall few elementary concepts regarding monoids. We should warn the reader however that the transformation monoids do not give a complete picture of regular languages and that one needs to look at certain other properties of the languages as well.

A *monoid* M is a set together with an associative binary operation that contains a distinguished identity element 1_M. We will denote this operation multiplicatively, e.g., for all $m \in M$, $m1_M = 1_M m = m$. For a finite alphabet Σ, an example of a monoid is the (*free*) monoid Σ^* with the operation concatenation and the identity element the empty word. Except for the free monoid Σ^*, all the monoids that we consider in this paper will be finite.

An element $m \in M$ is called an *idempotent* if $m = m^2$. One can easily verify that since M is finite, there exists the smallest integer $\omega \geq 1$, the *exponent of* M, such that for every $m \in M$, m^ω is an idempotent. A monoid G where for every element $a \in G$ there is an *inverse* $b \in G$ such that $ab = ba = 1_G$ is a *group*. A monoid M is called *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 the *product over* M is the function $f : M^* \rightarrow M$ such that $f(m_1 m_2 \cdots m_n) = m_1 m_2 \cdots m_n$. The *prefix-product over* M is the function $\Pi_p : M^* \rightarrow M^*$ defined as $\Pi_p(m_1 m_2 \ldots m_n) = p_1 p_2 \cdots p_n$, where for $i = 1, \ldots, n$, $p_i = m_1 m_2 \cdots m_i$. Similarly we can define the *suffix-product over* M as a function $\Pi_s : M^* \rightarrow M^*$ defined by $\Pi_s(m_1 m_2 \ldots m_n) = s_1 s_2 \cdots s_n$, where $s_i = m_i m_{i+1} \cdots m_n$. For $a \in M$, the *a-word problem over* M is the language of words from M^* that multiply out to a. When we are not particularly concerned about the choice of a we will often refer to such problems just as *word problems over* M. Notice that all word problems over M are indeed regular languages.

For monoids M, N, a function $\phi : N \rightarrow M$ 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 $\phi : \Sigma^* \rightarrow M$ 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, and the associated morphism $\nu_L : \Sigma^* \rightarrow M(L)$ we call the *syntactic morphism of* L. It turns out that the syntactic monoid $M(L)$ of L is the monoid of state transformations generated by the minimum state finite automaton recognizing L, i.e. every element of $M(L)$ can be thought of as a map of states of the automaton to itself.

2.1 Boolean circuits

Boolean circuits are classified by their size, depth and type of gates they use. For us the following classes of circuits will be relevant. NC^1 circuits are circuits of logarithmic depth consisting of fan-in two AND and OR gates, and unary NOT gates. Because of the bound on the depth and fan-in, NC^1 circuits are of polynomial size. AC^0, $\mathrm{AC}^0[q]$, ACC^0, TC^0 circuits are all of constant depth and polynomial size. AC^0 circuits consist of unbounded fan-in AND and OR gates, and unary NOT gates whereas $\mathrm{AC}^0[q]$ circuits contain in addition unbounded fan-in MOD-q gates. (A MOD-q gate is a gate that evaluates to one iff the number of ones that are feed into it is divisible by q.) ACC^0 circuits are union of $\mathrm{AC}^0[q]$ circuits over all $q \geq 1$. Finally, TC^0 circuits are circuits consisting of unbounded fan-in AND, OR and MAJ gates, and unary NOT gates. (A MAJ gate is a gate that evaluates to one iff the majority of its inputs is set to one.)

So far we did not say what we mean by the *size of a circuit.* There are two possible measures of the circuit size—the number of gates and the number of wires. As these two measures usually differ by at most a square the difference in these measures is usually not important. As we will see for us it will make a difference. Unless we say otherwise we will mean by the size of a circuit the number of its gates.

Beside languages over a binary alphabet we consider also languages over an arbitrary alphabet Σ. In such cases we assume that there is some fixed encoding of symbols from Σ into binary strings of fixed length, and inputs from Σ^* to circuits are encoded symbol by symbol using such encoding. Similarly, a circuit for a function with non-Boolean output produces a binary encoding of the output symbol.

There is a close relationship between a circuit complexity of a regular language L and the circuit complexity of a word problem over its syntactic monoid $M(L)$. One can easily establish the following relationship.

Proposition 1
1. *If a regular language L is computable by a circuit family of size $s(n)$ and depth $d(n)$ and for some $k \geq 0$, $\nu_L(L^{=k}) = M(L)$ then the product over its syntactic monoid $M(L)$ is computable by a circuit family of size $O(s(O(n)) + n)$ and depth $d(O(n)) + O(1)$.*

2. *If the product over a monoid M is computable by a circuit family of size $s(n)$ and depth $d(n)$ then any regular language with the syntactic monoid M is computable by a circuit family of size $s(n) + O(n)$ and depth $d(n) + O(1)$.*

The somewhat technical condition that for some k, $\nu_L(L^{=k}) = M(L)$ is unfortunately necessary as the language $LENGTH(2)$ of strings of even length does not satisfy the conclusion of the first part of the claim in the case of AC^0 circuits. However, the first part of the proposition applies in particular to regular languages that contain a *neutral letter*, a symbol that can be arbitrarily added into any word without affecting its membership/non-membership in the language. For $L \subseteq \Sigma^*$, $L^{=k}$ means $L \cap \Sigma^k$.

3 Mapping the landscape

It is folklore that all regular languages are computable by linear size NC^1 circuits. Indeed by Proposition 1 it suffices to show that there are NC^1 circuits of linear size for the product of n

elements over a fixed monoid M: recursively reduce computation of a product of n elements over M to a product of $n/2$ elements over M by computing the product of adjacent pairs of elements in parallel. Turning such a strategy into a circuit provides a circuit of logarithmic depth and linear size. Thus we can state:

Theorem 2 *Every regular language is computable by* NC^1 *circuits of linear size.*

Can all regular languages be put into even smaller circuit class? A surprising result of Barrington [1] indicates that this is unlikely: if a monoid M contains a non-solvable group then the word problem over M is hard for NC^1 under projections. Here, a *projection* is a simple reduction that takes a word w from a language L and maps it to a word w' from a language L' so that each symbol of w' depends on at most one symbol of w and the length of w' depends only on the length of w. Thus, unless NC^1 collapses to a smaller class such as TC^0, NC^1 circuits are optimal for some regular languages. The theorem of Barrington was further extended by Barrington et al. [2] to obtain the following statement.

Theorem 3 ([1, 2]) *Any regular language whose syntactic monoid contains a non-solvable group is hard for* NC^1 *under projections.*

An example of a monoid with a non-solvable group is the group S_5 of permutations on five elements. Thus for example the word problem over the group S_5 is hard for NC^1 under projections.

Chandra, Fortune and Lipton [5] identified a large class of languages that are computable by AC^0 circuits.

Theorem 4 ([5]) *If a language L has a group-free syntactic monoid $M(L)$ then L is in* AC^0.

The regular languages with group-free syntactic monoids have several alternative characterizations. They are precisely the *star-free languages*, the languages that can be described by a regular expression using only union, concatenation and complement operations but not the operation star where the atomic expressions are languages $\{a\}$ for every $a \in \Sigma$. They are also the *non-counting languages*, the languages L that satisfy: there is an integer $n \geq 0$ so that for all words x, y, z and any integer $m \geq n$, $xy^m z \in L$ iff $xy^{m+1}z \in L$.

The proof of Chandra et al. uses the characterization of counter-free regular languages by flip-flop automata of McNaughton and Papert [11]. Using this characterization one only needs to prove that the prefix-product over *carry semigroup* is computable by AC^0 circuits. The carry semigroup is a monoid with three elements P, R, S which multiply as follows: $xP = x$, $xR = R$, $xS = S$ for any $x \in \{P, S, R\}$. The carry semigroup is especially interesting because of its relation to the problem of computing the addition of two numbers represented in binary.

Chandra et al. also prove a partial converse of their claim.

Theorem 5 ([5]) *If a monoid M contains a group then the product over M is not in* AC^0.

Their proof shows how a product over a monoid with a group can be used to count the number of ones in an input from $\{0,1\}^*$ modulo some constant $k \geq 2$. However, by the result of Furst, Saxe and Sipser [7] that cannot be done in AC^0 so the product over monoids containing groups cannot be in AC^0.

There is still an apparent gap between Theorems 4 and 5. Namely, the language $LENGTH(2)$ of words of even length is in AC^0 although its syntactic monoid contains a group. This gap was closed by Barrington et al. [2].

Theorem 6 ([2]) *A regular language is in* AC^0 *iff for every* $k \geq 0$*, the image of* $L^{=k}$ *under the syntactic morphism* $\nu_L(L^{=k})$ *does not contain a group.*

Surprisingly, there is a beautiful characterization of these languages using regular expressions provided by [2]. L is in AC^0 iff it can be described by a regular expression using operations union, concatenation and complement with the atoms $\{a\}$ for every $a \in \Sigma$ and $LENGTH(q)$ for every $q \geq 1$. $LENGTH(q)$ is the language of all words whose length is divisible by q.

The remaining gap between regular languages with group-free monoids and monoids that contain non-solvable groups was essentially closed by Barrington [1]:

Theorem 7 ([1]) *If a syntactic monoid of a language contains only solvable groups then the language is computable by* ACC^0 *circuits.*

An example of such a language is the language PARITY of words from $\{0,1\}^*$ that contain an even number of ones. There is a very nice characterization of also these languages by regular expressions of certain type. For this characterization we need to introduce one special regular operation on languages. For a language $L \subseteq \Sigma^*$ and $w \in \Sigma^*$, let L/w denote the number of initial segments of w which are in L. For integers $m > 1$ and $0 \leq r < m$ we define $\langle L, r, m\rangle = \{w \in \Sigma^*;\ L/w \equiv r \bmod m\}$. Straubing [16] shows that the syntactic monoid of a language contains only solvable groups iff the language can be described by a regular expression built from atoms $\{a\}$, for $a \in \Sigma$, using operations union, concatenation, complement and $\langle La, r, m\rangle$, for any $a \in \Sigma$, $m > 1$ and $0 \leq r < m$.

The above results essentially completely classify all regular languages with respect to their circuit complexity—they are complete for NC^1, they are computable in AC^0 or otherwise they are in ACC^0. It is interesting to note that the class TC^0 does not get assigned any languages unless it is equal either to NC^1 or ACC^0. Proving that a regular language with its syntactic monoid containing non-solvable group is in TC^0 would collapse NC^1 to TC^0. Currently not much is known about the relationship of classes ACC^0, TC^0, and NC^1 except for the trivial inclusions $\mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1$.

In the next section we refine the classification of regular languages even further.

4 Circuit size of regular languages

In the previous section we have shown that all regular languages are computable by linear size NC^1 circuits. Can anything similar be said about regular languages in AC^0 or ACC^0? The answer may be somewhat surprising in the light of the following example. Let Th_2 be the language over the alphabet $\{0,1\}$ of words that contain at least two ones. This is clearly a regular language and it is in AC^0: check for all pairs of input positions whether anyone of them contains two ones. However this gives an AC^0 circuit of quadratic size and it is not at all obvious whether one can do anything better. Below we show a general procedure that produces more efficient circuits. We note here that the language Th_2 as well as all the *threshold languages Th_k* for up-to even poly-logarithmic k are in fact computable by linear size AC^0 circuits [14]. The construction of Ragde and Wigderson [14] is based on perfect hashing and it is not known if it could be applied to other regular languages.

Despite that we can easily reduce the size of constant depth circuits computing regular languages as follows. Assume that a regular language L and the product over its syntactic monoid is computable by $O(n^k)$-size constant-depth circuits. We construct $O(n^{(k+1)/2})$-size constant-depth circuits for product over $M(L)$: divide an input $x \in M(L)^n$ into consecutive blocks of size $\sqrt{n}$ and compute the product of each block in parallel; then compute the product of the $\sqrt{n}$ products obtained in the previous step. Clearly, the computation of the block products can be done in size $O(\sqrt{n} \cdot n^{k/2})$, as computing the product of $\sqrt{n}$ monoid elements requires circuits of size $O(n^{k/2})$. Thus, the total size of the circuit is $O(n^{(k+1)/2})$ and the depth of the circuit only doubles. This construction can be further iterated constantly many times to obtain the following result.

Proposition 8 *Let L be a regular language computable by a polynomial-size constant-depth circuits over arbitrary gates. If the product over its syntactic monoid $M(L)$ is computable by circuits of the same size then for every $\epsilon > 0$, there is a constant-depth circuit family of size $O(n^{1+\epsilon})$ that computes L.*

A substantial improvement comes in the work of Chandra et al. [5] who prove:

Theorem 9 ([5]) *Let $g_0(n) = n^{1/4}$ and further for each $d = 0, 1, 2, \dots$, $g_{d+1}(n) = g_d^*(n)$. Every regular languages L with a group-free syntactic monoid is computable by AC^0 circuits of depth $O(d)$ and size $O(n \cdot g_d^2(n))$, for any $d \geq 0$.*

Here $g^*(n) = \min\{i;\ g^{(i)}(n) \leq 1\}$, where $g^{(i)}(\cdot)$ denotes $g(\cdot)$ iterated i-times. Hence, Chandra et al. prove that almost all languages that are in AC^0 are computable by circuit families of almost linear size. Clearly the same is true for the product over group-free monoids. We generalize this to all regular languages computable in AC^0.

Theorem 10 *Let $g_d(n)$ be as in Theorem 9. Every regular languages L in AC^0 is computable by AC^0 circuits of depth $O(d)$ and size $O(n \cdot g_d^2(n))$, for any $d \geq 0$.*

The proof is a simple extension of the result of Chandra et al. We need to establish the following proposition that holds for all languages in AC^0.

Proposition 11 *Let for every $n \geq 0$, the image of $L^{=n}$ under the syntactic morphism ν_L does not contain any group. Then there is a $k \geq 1$ and a group-free monoid $M \subseteq M(L)$ such that for all $w \in \Sigma^*$, if k divides the length of w then $\nu_L(w) \in M$.*

Proof. For every group G in $M(L)$ let $v_G \in \Sigma^*$ be an arbitrary non-empty word such that $\nu_L(v_G) = 1_G$ if such a word exists. Let k be the product of the lengths $|v_G|$. (Set $k = 1$ if no v_G exists.) Let $M = \{\nu_L(w);\ w \in \Sigma^* \ \&\ k \mid |w|\}$. Clearly, M is a monoid so we only need to argue that it is group-free. Let g be an element of a group G in $M(L)$. Assume that $g \neq 1_G$ and that there is a word w of length divisible by k such that $\nu_L(w) = g$. As $M(L)$ is finite, there exists $\ell \geq 1$ such that $g^\ell = 1_G$. Since $\nu_L(w^\ell) = 1_G$, v_G exists. Clearly, $\{g^1, g^2, \dots, g^\ell\}$ forms a group. This group is the image of words $w^j (v_G)^{(\ell-j)|w|/|v_g|}$ under ν_L, for $j = 1, 2, \dots, \ell$. Since all these words are of the same length, the word w cannot exist. □

Proof of Theorem 10. If L is a regular language in AC^0 then by the previous proposition there is a group-free monoid M and an integer $k \geq 1$ such that all words of length divisible by k are mapped into M by the syntactic morphism of L. Consider an integer $n \geq 1$. It suffices to show that we can compute $\nu_L(w)$ for any word $w \in \Sigma^n$. The word w can be divided into blocks $b_1, b_2, \ldots, b_m$ of length k and one block b of length at most k so that $w = b_1 b_2 \cdots b_m b$. Clearly, $\nu_L(w) = \nu_L(b_1) \cdot \nu_L(b_2) \cdots \nu_L(b_m) \cdot \nu_L(b)$. Hence we can build a circuit that divides its input w into blocks b_i of length k, for each block b_i it computes the mapping $\nu_L(b_i)$ to obtain elements in M, computes the product $m' = \nu_L(b_1) \cdot \nu_L(b_2) \cdots \nu_L(b_m)$ using the circuit of depth $O(d)$ and size $O(n \cdot g_d^2(n))$ guaranteed to exist by Theorem 9, and computes $\nu_L(w) = m' \cdot \nu_L(b)$. As k is a constant the depth of the circuit will be $O(d)$ and size $O(n \cdot g_d^2(n))$. □

Chandra et al. prove actually the even stronger statement that the prefix-problem of these regular languages is computable in that size and using that many wires. We use the technique of Chandra et al. [5] together with the regular expression characterization of languages to show a similar statement for the regular languages in ACC^0 whose syntactic monoid does not contain a non-solvable group. (Alternatively, instead of characterization of regular languages in ACC^0 by regular expressions we could use their similar characterization by Thérien [20].)

Theorem 12 *Let $g_i(n)$ be as in Theorem 9. Every regular language L whose syntactic monoid contains only solvable groups is computable by ACC^0 circuits of size $O(n \cdot g_i^2(n))$.*

Assuming that ACC^0 and NC^1 are different the above theorem indeed applies to all regular languages in ACC^0.

The following general procedure that allows to build more efficient circuits for the prefix-product over a monoid M from circuits for the product over monoid M and less efficient circuits for the prefix-product over M is essentially the procedure of Chandra et al. Together with the inductive characterization of regular languages by regular expressions it provides the necessary tools to prove the above theorem. Let $g : \mathcal{N} \to \mathcal{N}$ be a non-decreasing function such that for all $n > 0$, $g(n) < n$, and M be a monoid with the product and prefix-product computable by constant-depth circuits.

CFL procedure:

Step 0. We split the input $x \in M^n$ iteratively into sub-words. We start with x as the only sub-word of length n and we divide it into $n/g(n)$ sub-words of size $g(n)$. We iterate and further divide each sub-word of length $l > 1$ into $l/g(l)$ sub-words of length $g(l)$. Hence, for $i = 0, \ldots, g^*(n)$ we obtain $n/g^{(i)}(n)$ sub-words of length $g^{(i)}(n)$.

Step 1. For every sub-word obtained in Step 0 we compute its product over M.

Step 2. Using results from Step 1 and existing circuits for prefix-product, for each sub-word of length $l > 1$ from Step 0 we compute the prefix-product of the products of its $l/g(l)$ sub-words.

Step 3. For each $i = 1, \ldots, n$, we compute the product of $x_1 \cdots x_i$ by computing the product of $g^*(n)$ values of the appropriate prefixes obtained in Step 2.

Let us analyze the circuit obtained from the above procedure. Assume that we have existing circuits of size $s(n)$ and constant depth d_s for computing product over M and of size

$p(n)$ and constant depth d_p for computing prefix-product over M. Then the above procedure gives a circuits of depth $2d_s + d_p$ and size

$$\sum_{i=0}^{g^*(n)} \frac{n}{g^{(i)}(n)} \cdot s(g^{(i)}(n)) + \sum_{i=0}^{g^*(n)-1} \frac{n}{g^{(i)}(n)} \cdot p\left(\frac{g^{(i)}(n)}{g^{(i+1)}(n)}\right) + n \cdot s(g^*(n)).$$

We demonstrate the use of the above procedure. Let M be a monoid such that the product over M is computable by polynomial size constant-depth circuit family. Choose $\epsilon > 0$. From Proposition 8 we have circuits of size $s(n) = O(n^{1+\epsilon})$ for computing the product over M. By choosing $g(n) = n/2$ and computing the prefix-product of two elements by a trivial circuit we obtain a circuit for the prefix-product over M of constant depth and size $O(n^{1+\epsilon} \log n)$. We can state the following proposition.

Proposition 13 *Let L be a regular language computable by a polynomial-size constant-depth circuits over arbitrary gates. If the product over its syntactic monoid $M(L)$ is computable by similar circuits then for every $\epsilon > 0$, there is a constant-depth circuit family of size $O(n^{1+\epsilon})$ that computes the prefix-product over the monoid $M(L)$.*

The previous proposition states clearly something non-trivial as a naïve construction of a prefix-product circuit would produce at least quadratic size circuits. We can derive also the following key lemma from the CFL procedure.

Lemma 14 *Let $g_0(n) = n^{1/4}$ and further for $i = 0, 1, 2, \ldots$, $g_{i+1}(n) = g_i^*(n)$. Let M be a monoid. If there is a size $O(n \cdot g_{i+1}(n))$ depth d_s circuit family for computing product over M and a size $O(n \cdot g_i^2(n))$ depth d_p circuit family for computing prefix-product over M then there is a size $O(n \cdot g_{i+1}^2(n))$ depth $2d_s + d_p$ circuit family for computing prefix-product over M.*

Proof. Use the CFL procedure and choose $g(n) = g_i^2(n)$. The size of the resulting circuit can be bounded by

$$\begin{aligned} & \sum_{i=0}^{g^*(n)} \frac{n}{g^{(i)}(n)} \cdot g^{(i)}(n) \cdot g_{i+1}(g^{(i)}(n)) + \sum_{i=0}^{g^*(n)-1} \frac{n}{g^{(i)}(n)} \cdot \frac{g^{(i)}(n)}{g^{(i+1)}(n)} \cdot g_i^2\left(\frac{g^{(i)}(n)}{g^{(i+1)}(n)}\right) \\ & +n \cdot g^*(n) \cdot g_{i+1}(g^*(n)) \\ \leq\ & g^*(n) \cdot n \cdot g_{i+1}(n) + g^*(n) \cdot n + n \cdot g^*(n) \cdot g_{i+1}(n). \end{aligned}$$

Since $g_i(n) \leq n^{1/4}$, $g_i^2(g_i^2(n)) \leq g_i(n)$. Thus $g^*(n) = (g_i^2)^*(n) \leq 2g_i^*(n) = 2g_{i+1}(n)$. □

It is trivial that if we can compute the prefix-product over some monoid M by $O(n \cdot g_i^2(n))$ circuits then we can also compute the product by the same size circuits. The above lemma provides essentially the other direction, i.e., building efficient circuits for the prefix-product from circuits for the product.

Since we will need to go back and forth from a language to the product over its syntactic monoid and Proposition 1 is not applicable to some languages without a neutral letter we will extend every language by a neutral letter. For a language $L \subseteq \Sigma^*$ and $\epsilon \notin \Sigma$ we define the *extension L_ϵ of L by the neutral letter ϵ* to be the set of words $w \in (\Sigma \cup \{\epsilon\})^*$ such that if we remove all occurrences of ϵ from w we get a word from L. One can easily verify that if we take a regular expression R for a language L that uses operations union, complement, concatenation and $\langle La, r, m\rangle$-operation, and we replace in R every occurrence of each atom $\{a\}$ by a regular expression for $\{a\}_\epsilon$ we get a regular expression that describes precisely L_ϵ. It is clear that if L_ϵ is computable by circuits of some particular size and depth then L is computable by essentially the same circuits as $L = L_\epsilon \cap \Sigma^*$.

Proof of Theorem 12. We prove the theorem by an induction on the depth of the regular expression describing L. Indeed, we will prove that L_ϵ is computable by constant-depth circuits of size $O(n \cdot g_i^2(n))$. The base case is simple as the languages $\{a\}$ and $\{a\}_\epsilon$ are recognized by linear size circuits of constant depth.

For the induction step let L be obtained from L_1 via complement or the $\langle L, r, m\rangle$-operation or from L_1 and L_2 via union or concatenation. By induction hypothesis, for all $i \geq 0$, $L_{1\epsilon}$ and $L_{2\epsilon}$ are computable by constant-depth ACC^0 circuits of size $O(n \cdot g_i^2(n))$. Thus, if L is obtained via union or complement then clearly L as well as L_ϵ are computable by ACC^0 circuits of size $O(n \cdot g_i^2(n))$.

If we can argue that for all $i \geq 0$, the computation of the membership in $L_{1\epsilon}$ of all prefixes of an input w and of the membership in $L_{2\epsilon}$ of all suffixes of w can be done in parallel by ACC^0 circuits of size $O(n \cdot g_i^2(n))$ then we will be done with the proof as such an information can be processed by a simple circuitry to obtain the answer whether w is in L_ϵ.

We claim that the membership of all prefixes of an input w in $L_{1\epsilon}$ can be computed in parallel by ACC^0 circuit of size $O(n \cdot g_i^2(n))$, for all $i \geq 0$. Since $L_{1\epsilon}$ has a neutral letter we can apply Propositions 1 and 13 to conclude that the prefix-product over $M(L_{1\epsilon})$ can be computed by constant-depth ACC^0 circuits of size $O(n^{1+1/2}) = O(n \cdot g_0^2(n))$. By induction hypothesis the product over $M(L_{1\epsilon})$ is computable by ACC^0 circuits of size $O(n \cdot g_i^2(n))$, for all $i \geq 0$. Thus by repeated use of Lemma 14, the prefix-product over $M(L_{1\epsilon})$ is computable by constant-depth circuits of size $O(n \cdot g_i^2(n))$, for any $i \geq 0$. The claim follows.

One can argue similarly for the membership of suffixes of w in $L_{2\epsilon}$ since computing a suffix-product over $M(L_{2\epsilon})$ can be done by a procedure similar to the CFL procedure. □

5 Wires vs. gates

It is a natural question whether all languages that are in AC^0 and ACC^0 could be computed by AC^0 and ACC^0 circuits, resp., of linear size. This is not known, yet:

Open problem 15 *Is every regular language in* AC^0 *or* ACC^0 *computable by linear-size* AC^0 *or* ACC^0 *circuits?*

One would be tempted to conjecture that this must be the case as $O(n \cdot g_d(n))$ may not look like a very natural bound. However, as we shall see further such an intuition fails when considering the number of wires in a circuit. As we mentioned earlier, Chandra et al. in fact proved Theorem 4 in terms of wires instead of gates. A close inspection of our arguments in the previous section reveals that our Theorems 10 and 12 also hold in terms of wires. Hence we can strengthen the results of the previous section as follows:

Theorem 16 *Let L be a regular language and functions g_d be as in Theorem 9.*

- *If L is in* AC^0 *then for every $d \geq 0$ it is computable by* AC^0 *circuits using $O(ng_d^2(n))$ wires.*

- *If L is in* ACC^0 *and it is not hard for* NC^1 *then for every $d \geq 0$ it is computable by* ACC^0 *circuits using $O(ng_d^2(n))$ wires.*
- *If L is in* ACC^0 *then for every $\epsilon > 0$ it is computable by* ACC^0 *circuits using $O(n^{1+\epsilon})$ wires.*

Interestingly enough the wire variant of Problem 15 was answered negatively in [10].

Theorem 17 ([10]) *There is a regular language in* AC^0 *that requires* AC^0 *and* ACC^0 *circuits of depth $O(d)$ to have $\Omega(n \cdot g_d(n))$ wires.*

The language from the theorem is the following simple language $U = c^*(ac^*bc^*)^*$. Although we have described it by a regular expression using the star-operation it is indeed in AC^0. What is really interesting about this language is that it is computable by ACC^0 circuits using a linear number of gates.

Theorem 18 ([10]) *The class of regular languages computable by* ACC^0 *circuits using linear number of wires is a proper subclass of the languages computable by* ACC^0 *circuits using linear number of gates.*

It is not known however whether the same is true for AC^0.

Open problem 19 *Are the classes of regular languages computable by* AC^0 *circuits using linear number of gates and liner number of wires different?*

The proof of Theorem 17 in [10] shows that any circuit computing U must contain within a weak type of *super-concentrator*, a graph that is known to require $\Omega(n \cdot g_d(n))$ of edges [6, 13]. A similar type of argument already appeared in [4] however in a substantially simpler setting of multi-output functions such as Integer Addition.

[10] provides a precise characterization of regular languages with neutral letter that are computable by AC^0 and ACC^0 circuits using linear number of wires.

Theorem 20 ([10]) *Let L be a regular language with a neutral letter.*

- *L is computable by* AC^0 *circuits with linear number of wires iff the syntactic monoid $M(L)$ satisfies the identity $(xyz)^\omega y(xyz)^\omega = (xyz)^\omega$, for every $x, y, z \in M(L)$.*
- *L is computable by* ACC^0 *circuits with linear number of wires iff the syntactic monoid $M(L)$ contains only commutative groups and $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$, for every $x, y, z \in M(L)$.*

The class of regular languages that are computable by AC^0 circuits with linear number of wires has several other characterizations. It is the class of *unambiguous* languages, languages that are recognized by a two-way finite state automata whose graph of transitions is a directed acyclic graph except for the self-loops (see [15] for more definitions). It is also the class of regular languages that can be described by boolean combinations of so called *turtle programs.* A turtle program is a sequence of simple instructions of the type "go left until you find letter a", or go "right until you find letter c". If on an input one never falls of either end of the input word when following the instructions while starting at the left or right end of the word, the word is accepted by the program. A similar turtle programs can also be defined for languages in ACC^0, where the turtle also evaluates some product in a commutative group as it moves along the word (see [10, 18] for more details).

6 Applications to circuit complexity

In the previous sections we have shown that all regular languages that have constant-depth circuits of a particular type have actually constant-depth circuits of the same type and of almost linear size (Proposition 8 and Theorem 16). This is true regardless of whether we count the number of wires or gates. Thus we get:

Theorem 21 *If for some $\epsilon > 0$ a regular language L does not have* ACC^0 *circuits of size $n^{1+\epsilon}$ then* $\mathrm{ACC}^0 \neq \mathrm{NC}^1$.

A consequence of a Barrington's result in Theorem 3 is that there are regular languages that are complete for NC^1 under projections, e.g., word problems over the permutation group S_5. If a language L has NC^1 circuits of depth d, $d \in O(\log n)$, then by the theorem of Barrington [1] it is reducible by projections to a word problem over S_5 of size 4^d. A typical proof that a language belongs to a certain circuit class is constructive, i.e., typically if we know that L is in NC^1 then we can actually determine d. Hence the following claim could be useful.

Theorem 22 *If a language (function) L is computable by* NC^1 *circuits of depth d, $d \in O(\log n)$ but not by* ACC^0 *circuits of size $O(4^{d+\epsilon})$ for some $d, \epsilon > 0$ then* $\mathrm{ACC}^0 \neq \mathrm{NC}^1$.

Hence instead of proving super-polynomial circuit lower bounds, for separating ACC^0 from NC^1, polynomial lower bounds should suffice. This may also partially explain the obstacles that one encounters when proving $\Omega(n^c)$ lower bounds, most of the currently known circuit lower bounds are well below n^4.

7 Conclusions

We have demonstrated that regular languages are very low on the ladder of complexity—they are computable by almost linear size circuits of different types. Still they provide important examples of explicit languages that separate different complexity classes. It is not much of an exaggeration to say that the state of the art circuit separations that we currently have are based on regular languages. Regular languages could still provide enough ammunition to separate for example ACC^0 from NC^1 which is currently a big open problem.

Several other questions that may be more tractable remain also open. We already mentioned the one whether all languages that are in AC^0 and ACC^0 are computable by linear size constant-depth circuits. The language U defined in the previous section is in particular interesting as it is the essentially simplest regular language not known to be computable by linear size AC^0 circuits. It is also closely related to Integer Addition: if two binary represented numbers can be summed up in AC^0 using linear size circuits then U is computable by linear size circuits as well. We can state the following open problem of wide interest:

Open problem 23 *What is the size of* AC^0 *and* ACC^0 *circuits computing Integer Addition?*

(Previously an unsupported claim appeared in literature that Integer Addition can be computed by linear size AC^0 circuits [14, 9].) If U indeed is computable by linear size AC^0 circuits then it presents an explicit language that separates the classes of languages computable in AC^0 using linear number of gates and using linear number of wires. Such an explicit language is already known [10] however that language is not very natural and was constructed explicitly to provide this separation. The language U would be a more natural explicit example. If U is not computable by AC^0 circuits of linear size then neither is Integer Addition.

We conclude with yet another very interesting problem that rather reaches somewhat outside of the realm of regular languages.

Open problem 24 *What is the number of wires in* AC^0 *and* ACC^0 *circuits computing* Th_k*, for* $k \in \omega(n)$*?*

Acknowledgements

Theorem 12 was obtained jointly with Denis Thérien. I would like to thank him for allowing me to publish the proof here and for sharing with me his insights on regular languages. I am greatful to Pascal Tesson and anonymous referees for useful comments that helped to improve this paper.

References

[1] D. A. Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[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] D. A. M. Barrington and D. Thérien. Finite Monoids and the Fine Structure of NC^1. *Journal of ACM*, 35(4):941–952, 1988.

[4] A. K. Chandra, S. Fortune, and R. J. Lipton. Lower bounds for constant depth circuits for prefix problems. In *Proc. of the 10th ICALP*, pp. 109–117, 1983.

[5] 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.

[6] D. Dolev, C. Dwork, N. Pippenger, and A. Wigderson. Superconcentrators, generalizers and generalized connectors with limited depth (preliminary version). In *Proc. of 15th STOC*, pp. 42–51, 1983.

[7] M. Furst, J. Saxe, and M. Sipser. Parity, circuits and the polynomial time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[8] J. Håstad. Almost optimal lower bounds for small depth circuits. In *Proc. of the 18th STOC*, pp. 6–20, 1986.

[9] S. Chaudhuri and J. Radhakrishnan. Deterministic restrictions in circuit complexity. In *Proc. of the 28th STOC*, pp. 30–36, 1996.

[10] M. Koucký, P. Pudlák, and D. Thérien. Bounded-depth circuits: Separating wires from gates. In *Proc. of the 37th STOC*, pp. 257–265, 2005.

[11] R. McNaughton and S.A. Papert. Counter-Free Automata. The MIT Press, 1971.

[12] J.-E. Pin. Syntactic semigroups. Chap. 10 in *Handbook of language theory, Vol. I.* G. Rozenberg and A. Salomaa (ed.), Springer Verlag, pp. 679–746, 1997.

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

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

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

[16] H. Straubing. Families of recognizable sets corresponding to certain varieties of finite monoids. *Journal of Pure and Applied Algebra*, 15(3):305–318, 1979.

[17] P. Tesson and D. Thérien. Complete classifications for the communication complexity of regular languages. *Theory of Computing Systems*, 38(2):135–159, 2005.

[18] P. Tesson and D. Thérien. Restricted Two-Variable Sentences, Circuits and Communication Complexity. In *Proc. of ICALP*, pp. 526–538, 2005.

[19] P. Tesson and D. Thérien. Bridges between algebraic automata theory and complexity theory. In *The Complexity Column, Bull. EATCS*, 88:37–64, 2006.

[20] D. Thérien. Classification of Finite Monoids: the Language Approach. *Theoretical Computer Science*, 14:195–208 1981.

[21] H. Vollmer. Introduction to Circuit Complexity - A Uniform Approach. *Texts in Theoretical Computer Science. An EATCS Series.* Springer Verlag, 1999.