Circuit Complexity of Regular Languages

Michal Koucký

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

Abstract. We survey our current knowledge of circuit complexity of regular languages. We show that regular languages are of interest as languages providing understanding of different circuit classes. We also 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].

Keywords: regular languages, circuit complexity.

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 [6] 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 [8], 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. [17,16]). (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 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. Beside that we provide a proof 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.

S.B. Cooper, B. Löwe, and A. Sorbi (Eds.): CiE 2007, LNCS 4497, pp. 426–435, 2007.

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. We recall few elementary concepts regarding monoids. 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 $m = m^2$. 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 $f : M^* \to M$ 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 $\Pi_p : M^* \to M^*$ 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 $\Pi_s : M^* \to M^*$ 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 concerned about a particular choice of a we will refer to such problems as *word problems over* M.

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, and the associated morphism $\nu_L : \Sigma^* \to M(L)$ we call the *syntactic morphism of* L. The syntactic monoid $M(L)$ of L happens to be 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.)

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. We use similar convention for circuits outputting non-Boolean values.

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 1. *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 2 ([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 3 ([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 [9]. 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 binary numbers.

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

Theorem 4 ([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 [6] 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 3 and 4. 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 5 ([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:

Theorem 6 ([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 [14] 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 $\mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1$.

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 [12]. The construction of Ragde and

Wigderson [12] is based on perfect hashing and it is not known if it could be applied to other regular languages.

Despite that we can 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 $\sqrt{n}$ block products. This construction can be iterated constantly many times to obtain:

Proposition 2. *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 7 ([5]). *Let $g_0(n) = n^{1/4}$ and further for each $d = 0, 1, 2, \ldots$, $g_{d+1}(n) = g_d^*(n)$. 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 8. *Let $g_d(n)$ be as in Theorem 7. 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 3. *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 $|w|$ then $\nu_L(w) \in M$.*

Due to space limitations we omit proofs of Proposition 3 and Theorem 8 from this version of the paper. They can be found in the full version of the paper.

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. (Alternatively, we could use Thérien's characterization of regular languages in ACC^0 [17].)

Theorem 9. *Let $g_i(n)$ be as in Theorem 7. Every regular language L that is computable by ACC^0 circuits is computable by ACC^0 circuits of size $O(n \cdot g_i^2(n))$.*

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 circuits. Let $\epsilon > 0$. Proposition 2 gives us circuits of size $s(n) = O(n^{1+(\epsilon/2)})$ for computing the product over M. By choosing $g(n) = n/2$ we obtain the following proposition.

Proposition 4. *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 $M(L)$.*

Proposition 4 states clearly something non-trivial as a naïve construction of a prefix-product circuit would provide at least quadratic size circuits. By setting $g(n) = g_i^2(n)$ we obtain the following key lemma from the CFL procedure.

Lemma 1. *Let $g_i(n)$ be as in Theorem 7. 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.

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.

These two claims are sufficient to prove Theorem 9. We omit the proof due to space limitations. The proof can be found in the full version of this paper.

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:

Problem 1. 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 3 in terms of wires instead of gates. A close inspection of our arguments in the previous section reveals that our Theorems 8 and 9 also hold in terms of wires. Hence we obtain:

Theorem 10. *Let L be a regular language, $d > 0$ be an integer and functions g_d be as in Theorem 7.*

- *If L is in AC^0 then it is computable by AC^0 circuits with $O(ng_d^2(n))$ wires.*
- *If L is in ACC^0 then it is computable by ACC^0 circuits with $O(ng_d^2(n))$ wires.*

Interestingly enough the wire variant of Problem 1 was answered negatively:

Theorem 11 ([8]). *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 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 12 ([8]). *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.

Problem 2. Are the classes of regular languages computable by AC^0 circuits using linear number of gates and liner number of wires different?

[8] 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 13 ([8]). *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 $x, y, z \in M(L)$.*

Due to space limitations we omit here several other beautiful characterizations of the language classes from the previous theorem [8,15,16].

6 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 current state of the art circuit separations are based on regular languages. Regular languages could still provide enough ammunition to separate say ACC^0 from NC^1. Such a separation is currently a major 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 particularly 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:

Problem 3. 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 [12,7].) 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 [8] however that language is not very natural and was constructed explicitly to provide this separation. 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 reaches somewhat outside of the realm of regular languages.

Problem 4. What is the number of wires in AC^0 and ACC^0 circuits computing Th_k, for $k \in \omega(n)$?

Acknowledgements

Theorem 9 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 grateful to Pascal Tesson and anonymous referees for useful comments that helped to improve this paper. Partially supported by grant GA ČR 201/07/P276 and 201/05/0124.

References

1. Barrington, D.A.: 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. Barrington, D.A.M., Compton, K.J., Straubing, H., Thérien, D.: Regular languages in NC^1. Journal of Computer and System Sciences 44(3), 478–499 (1992)
3. Barrington, D.A.M., Thérien, D.: Finite Monoids and the Fine Structure of NC^1. Journal of ACM 35(4), 941–952 (1988)
4. Chandra, A.K., Fortune, S., Lipton, R.J.: Lower bounds for constant depth circuits for prefix problems. In: Proc. of the 10th ICALP, pp. 109–117 (1983)
5. Chandra, A.K., Fortune, S., Lipton, R.J.: Unbounded fan-in circuits and associative functions. Journal of Computer and System Sciences 30, 222–234 (1985)
6. Furst, M., Saxe, J., Sipser, M.: Parity, circuits and the polynomial time hierarchy. Mathematical Systems Theory 17, 13–27 (1984)
7. Chaudhuri, S., Radhakrishnan, J.: Deterministic restrictions in circuit complexity. In: Proc. of the 28th STOC, pp. 30–36 (1996)
8. Koucký, M., Pudlák, P., Thérien, D.: Bounded-depth circuits: Separating wires from gates. In: Proc. of the 37th STOC, pp. 257–265 (2005)
9. McNaughton, R., Papert, S.A.: Counter-Free Automata. The MIT Press, Cambridge (1971)
10. Pin, J.-E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Chap. 10 in Handbook of language theory, vol. I, pp. 679–746. Springer, Heidelberg (1997)
11. Pudlák, P.: Communication in bounded depth circuits. Combinatorica 14(2), 203–216 (1994)
12. Ragde, P., Wigderson, A.: Linear-size constant-depth polylog-threshold circuits. Information Processing Letters 39, 143–146 (1991)
13. Schwentick, T., Thérien, D., Vollmer, H.: Partially ordered two-way automata: a new characterization of DA. In: Proc. of DLT, pp. 242–253 (2001)
14. Straubing, H.: Families of recognizable sets corresponding to certain varieties of finite monoids. Journal of Pure. and Applied Algebra 15(3), 305–318 (1979)
15. Tesson, P., Thérien, D.: Restricted Two-Variable Sentences, Circuits and Communication Complexity. In: Proc. of ICALP, pp. 526–538 (2005)
16. Tesson, P., Thérien, D.: Bridges between algebraic automata theory and complexity theory. The Complexity Column, Bull. EATCS 88, 37–64 (2006)
17. Thérien, D.: Classification of Finite Monoids: the Language Approach. Theoretical Computer Science 14, 195–208 (1981)
18. Vollmer, H.: Introduction to Circuit Complexity - A Uniform Approach. In: Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (1999)