Bridges between Algebraic Automata Theory and Complexity Theory*

Pascal Tesson† Denis Thérien‡

Abstract

The algebraic theory of finite automata has been one of the most successful tools to study and classify regular languages. These very same tools can in fact be used to understand more powerful models of computation and we discuss here the impact that semigroup theory can have in computational complexity.

1 Introduction

Many interesting results in the study and classification of regular languages have come from an algebraic perspective pioneered by Eilenberg [23]. As we describe more precisely in the next section, finite automata and finite semigroups and monoids are closely related and one can naturally choose to consider automata as algebraic objects or, conversely, consider finite semigroups and monoids as machines. This connection makes it possible to relate the combinatorial properties of a regular language with the algebraic properties of its minimal automaton. One such famous example is Schützenberger's characterization of star-free regular languages [52]: a language can be described by an extended regular expression without the use of the Kleene star if and only if the semigroup corresponding to its minimal automaton contains no non-trivial group. A number of important classes of regular languages admit similar algebraic characterizations, often yielding decidability results which are not known to be obtainable by other means.

This algebraic perspective conveniently allows one to switch between the language theoretic and semigroup theoretic viewpoints on finite automata.

*Research supported by NSERC and FQRNT.

†Département d'Informatique et de Génie Logiciel, Université Laval, Québec, Canada. pascal.tesson@ift.ulaval.ca

‡School of Computer Science, McGill University, Montréal, Canada. denis@cs.mcgill.ca

In fact, the interaction between automata theory and finite semigroup theory has been so successful that the two fields are now barely distinguishable. As one can witness by comparing the key references on the subject over the last fifty years [23, 45, 46, 2], finite semigroup theory has rapidly evolved into a rich mathematical field with considerable depth that involves category theory, topology and combinatorics.

In this column, we wish to explore the relevance of semigroup theoretic tools in computational complexity. At first glance, regular languages seem to be of little interest to computational complexity theory because they can all be recognized in linear time and constant space but there are other measures of complexity for which regular languages provide a non-trivial case-study of considerable interest: circuit complexity [12, 38], communication complexity [50, 65], property testing [3], etc. In these investigations, the algebraic perspective on regular languages is often a key starting point and can help uncover unexpected similarities between the various models. There is, for instance, a very tight relation between the communication complexity and the circuit complexity of regular languages [38, 66].

By viewing finite semigroups as machines akin to finite automata we can analyze, with the same algebraic perspective, more powerful models of computation such as boolean circuits, branching programs and communication protocols. This approach has been particularly successful in analyzing the computational power of bounded-width branching programs and oolean circuits of shallow depth and has provided algebraic characterizations of important complexity classes such as AC^0, ACC^0 and NC^1 through the "program over monoid" formalism [12] and PSPACE and its main subclasses through the "leaf language" model [33]. In this context, it is sometimes possible to draw interesting parallels between algebraic operations on classes of semigroups, combinatorial operations on classes of regular languages and operations on complexity classes.

We believe that the rich algebraic theory of finite automata partly explains why the study of regular languages has had more impact in computational complexity than the rest of formal language theory. Often the combinatorial characterizations of classes of regular languages are sufficient to derive upper bounds on their complexity whereas the algebraic point of view is most useful when establishing hardness results for languages that do not belong to these same classes. We outline in this column how the interaction between semigroup theory and complexity has developed over the last twenty years and survey what we believe are promising avenues for future research. It should be noted, although this is beyond our present scope, that the computational relevance of algebraic automata theory has in turn introduced new ideas in finite semigroup theory (see the survey [64]).

In the next section, we will review the basics of algebraic automata theory and proceed to survey some applications of these methods in communication complexity, boolean circuit complexity, bounded width branching programs and learning theory.

2 Regular Languages and Finite Monoids

We begin with a brief overview of algebraic automata theory. A more thorough introduction can be found in the excellent survey of Pin [46].

A semigroup is a set S equipped with a binary associative operation which we denote multiplicatively. A monoid M is a semigroup with a distinguished identity element 1_M. Throughout this column S and M denote respectively a finite semigroup and a finite monoid.

The set Σ^* of finite words over the alphabet Σ forms a monoid (the *free monoid over* Σ) under concatenation and with identity ϵ, the empty word. With the exception of the free monoid, all monoids considered in the sequel are finite. If A is a finite automaton with states Q over the alphabet Σ, then a word $w \in \Sigma^*$ induces a mapping t_w from Q to Q. Since Q is finite, the set $T(A) = \{t_w : w \in \Sigma^*\}$ is also finite and because $t_u \circ t_v = t_{uv}$, $T(A)$ forms a monoid under composition, known as the *transformation monoid of* A. The identity element in that case is the identity transformation which corresponds to $w = \epsilon$[1].

For a language $L \subseteq \Sigma^*$ the syntactic congruence $\equiv_L$ on Σ^* is defined by setting $x \equiv_L y$ if and only if $uxv \in L \Leftrightarrow uyv \in L$ for all $u, v \in \Sigma^*$. It is well-known that L is a regular language if and only if $\equiv_L$ has finite index: in this case the syntactic monoid of L defined by $M(L) = \Sigma^*/\equiv_L$ is in fact the transition monoid of L's minimal automaton.

We say that a monoid M divides the monoid N and write $M \prec N$ if M is the homomorphic image of a submonoid of N. A class $\mathbf{V}$ of finite monoids forms a *pseudo-variety* if it is closed under direct product and division.

For a language L and a word u, the left quotient of L by u is the language $u^{-1}L = \{x : ux \in L\}$ and the right quotient Lu^{-1} is defined symmetrically. A class $\mathcal{V}$ of languages forms a *variety of languages* if it is closed under boolean operations, left and right quotients and inverse homomorphisms from one free monoid to another (i.e. if $L \subseteq \Sigma^*$ is in $\mathcal{V}$ and ϕ is a morphism from Γ^* to Σ^* then the language $\phi^{-1}(L)$ must also belong to $\mathcal{V}$). The variety theorem, which we state next, is the cornerstone of algebraic automata theory.

[1]We should mention here that for technical reasons we will always consider monoids rather than semigroups. We refer the interested reader to [46] for a thorough discussion of this technical point.

Theorem 1 (Variety theorem)**.** *If* $\mathbf{V}$ *is a pseudo-variety then the class* $L(\mathbf{V})$ *of languages with syntactic monoids in* $\mathbf{V}$ *is a variety of languages and these syntactic monoids generate* $\mathbf{V}$*. In fact, the correspondence* $\mathbf{V} \to \mathbf{L}(\mathbf{V})$ *defines a natural bijection between pseudo-varieties of finite monoids and varieties of languages.*

Because of this correspondence, we can easily switch our point of view on varieties and describe them either combinatorially as classes of regular languages or algebraically as classes of monoids. In many important cases this bijection has been made explicit. Schützenberger's result mentioned in the introduction and Simon's theorem, giving an algebraic characterization of the variety of piecewise-testable languages [53], are the most famous examples. This has been most fruitful in connection with logic. In that context, one basic question is to decide whether or not a given regular language can be described by a formula of some particular form. More often than not, these languages form a language variety (see [58] for a meta-explanation of this phenomenon) and it becomes possible to use computations on the syntactic monoid in the decision procedure. This approach yields decidability results for various classes of first-order formulas [42, 55, 68] and **FO** formulas augmented with modular quantifiers [57, 59, 60, 66] as well as temporal logics [20, 36, 68, 69]. There have been attempts to use the same approach to decide logically defined classes of regular tree languages although this extension faces a number of technical hurdles (see e.g. [24]).

For two elements a, b of a group G recall that the commutator $[a, b]$ of a and b is the group element $a^{-1}b^{-1}ab$. Clearly $[a, b] = 1_G$ if and only if a and b commute. A group G is *nilpotent of class* 1 if it is Abelian and, inductively is *nilpotent of class* k if the subgroup $[G, G]$ generated by its commutators is nilpotent of class $k - 1$. We denote as $\mathbf{Ab}$, $\mathbf{G_{nil,k}}$ and $\mathbf{G_{nil}}$ respectively the class of Abelian groups, nilpotent groups of class k and nilpotent groups. These classes form pseudo-varieties and there are good descriptions of the corresponding regular languages.

A word $u = a_1 \dots a_t$ with $a_i \in \Sigma$ is a *subword* of $w \in \Sigma^*$ if w can be factorized as $w_0 a_1 w_1 \dots w_{t-1} a_t w_t$ and we denote by $\binom{w}{u}$ the number of such factorizations. We say that a language L *counts subwords of length* k *modulo* m if membership of w in L depends on the values modulo m of $\binom{w}{u_1}, \dots, \binom{w}{u_t}$ for some words u_i each having length at most k. It was shown in [67] that a regular language K belongs to $L(\mathbf{G_{nil,k}})$ if and only if there exists $m \geq 1$ such that L counts subwords of length k modulo m. In particular, if the syntactic monoid of K is an Abelian group then membership of w in K depends solely on the number of occurrences of each letter modulo some integer m. There also exists an explicit, albeit more complex, description of regular languages

whose syntactic monoid is a solvable group but all attempts to provide a useful description of languages corresponding to groups in general have failed because of our very limited understanding of non-solvable groups. For any pseudo-variety $\mathbf{H}$ of groups, we will denote by $\overline{\mathbf{H}}$ the class of monoids whose subgroups lie in $\mathbf{H}$.

A monoid is *aperiodic* (or group-free) if it contains no non-trivial group. Groups and aperiodics are in some sense two extreme cases of monoids and they can be used as building blocks to construct all monoids via semidirect products [46].

An element e of M is *idempotent* if $e^2 = e$. For a finite monoid M there is a minimal integer ω, called M's exponent, such that m^ω is idempotent for all $m \in M$. We define the pseudo-variety $\mathbf{DO}$ as the class of finite monoids that satisfy the identity $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$. This pseudo-variety and its subclasses has found considerable importance both in language theory and in applications to complexity [63] and is the focus of the results surveyed in the sequel. The following lemma provides an alternative characterization of $\mathbf{DO}$ which is often more useful.

Lemma 2. *A monoid M does not belong to $\mathbf{DO}$ if and only if*

- *M is divided by the syntactic monoid of $(ab)^*$; or*
- *M is divided by the syntactic monoid of $\{a,b\}^*aa\{a,b\}^*$; or*
- *M is a T_q-monoid for some $q \geq 2$, i.e. there are idempotents $e, f \in M$ such that $(ef)^q e = ef$ but $(ef)^r \neq ef$ when q does not divide r.*

For any pseudo-variety of groups $\mathbf{H}$, it is possible to give a combinatorial description of the regular languages with syntactic monoids in $\mathbf{DO} \cap \overline{\mathbf{H}}$ using unambiguous concatenations. For $a \in \Sigma$ and $L, K \subseteq \Sigma^*$, we say that the concatenation LaK is *perfectly unambiguous* if $L \subseteq (\Sigma - \{a\})^*$ or $K \subseteq (\Sigma - \{a\})^*$. If LaK is perfectly unambiguous then any $w \in LaK$ can be uniquely factorized as $w_L a w_K$ with $w_L \in L$ and $w_K \in K$ since the a can only be the first or last occurrence of a in w.

Lemma 3 ([65])**.** *A language K has its syntactic monoid in $\mathbf{DO} \cap \overline{\mathbf{H}}$ iff it belongs to the smallest language variety closed under perfectly unambiguous concatenations which contains the languages Σ^* for any alphabet Σ and the languages with syntactic monoids in $\mathbf{H}$.*

As a corollary, it can be inferred that the pseudo-variety $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ corresponds to the smallest language variety containing the languages with commutative syntactic monoids and closed under perfectly unambiguous concatenations.

Perfectly unambiguous concatenations are a special case of a more general construction. For languages $K_0, \ldots, K_t \subseteq \Sigma^*$ and letters $a_1, \ldots, a_t \in \Sigma$, we denote as $\binom{w}{K_0 a_1 K_1 \ldots a_t K_t}$ the number of distinct ways in which the word $w \in \Sigma^*$ can be factorized as $w = w_0 a_1 \ldots a_t w_t$ with $w_i \in {}_i$. The concatenation $K_0 a_1 K_1 \ldots a_t K_t$ is said to be *unambiguous* if for all w it holds that $\binom{w}{K_0 a_1 K_1 \ldots a_t K_t}$ is 0 or 1. For any variety $\mathcal{V}$ of languages the *unambiguous polynomial closure of* $\mathcal{V}$, denoted $(\mathcal{V})$, is the variety generated by unambiguous concatenations of languages in $\mathcal{V}$. Similarly, for any prime p, the Mod_p *polynomial closure of* $\mathcal{V}$, denoted $\mathrm{Mod}_p\mathrm{Pol}(\mathcal{V})$, is the variety generated by languages of the form $\{w : \binom{w}{K_0 a_1 K_1 \ldots a_t K_t} \equiv i \pmod p\}$ for $K_i \in \mathcal{V}$. While the operators UPol and $\mathrm{Mod}_p\mathrm{Pol}$ are defined on varieties of languages, they have an algebraic counterpart defined in terms of so-called Mal'cev products of pseudo-varieties (see [46]): if $\mathbf{V}$ denotes the pseudo-variety of monoids corresponding to $\mathcal{V}$, then the pseudo-varieties corresponding to $\mathrm{UPol}(\mathcal{V})$ and $\mathrm{Mod}_p\mathrm{Pol}(\mathcal{V})$ are respectively $\mathbf{LI} \mathbin{Ⓜ} \mathbf{V}$ and $\mathbf{LG_p} \mathbin{Ⓜ} \mathbf{V}$ where $\mathbf{LI}$ and $\mathbf{LG_p}$ respectively denote the classes of locally trivial monoids and local p-groups [48, 71]. While a technical discussion of the Mal'cev product would be tedious, we simply mention that given a list of identities defining a pseudo-variety $\mathbf{V}$ it is often possible to derive a list of identities defining the two pseudo-varieties above [47, 48, 71]. In particular the following lemma can be obtained.

Lemma 4 ([65])**.** *A monoid does not belong to* $\mathbf{LG_p} \mathbin{Ⓜ} \mathbf{Com}$ *if and only if*

- *M is divided by the syntactic monoid of $(ab)^*$; or*
- *M is divided by the syntactic monoid of $\{a,b\}^* aa \{a,b\}^*$; or*
- *M is a T_q-monoid for some $q \geq 2$ which is not a power of p; or*
- *or M contains a subgroup H whose commutator subgroup is not a p-group.*

This lemma provides us with an excellent understanding of the Mod_p-closure of commutative languages. The unambiguous polynomial closure of these languages in fact corresponds to the pseudo-variety $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ which we discussed earlier.

3 Communication Complexity

Communication complexity measures the amount of information that two or more processors need to exchange when computing a function on some input

distributed among them. It was first introduced by Yao [72] in relation to distributed computing but it has become one of the most ubiquitous tools of theoretical computer science with important connections to VLSI design (see [40]), branching program complexity [70], circuit complexity [28, 29, 32, 43, 51], time-space tradeoffs for Turing machines [6], data structures [40] and proof complexity [13]. The monograph of Kushilevitz and Nisan [40] provides a very good overview of the first fifteen years of research on the subject.

In a (very entertaining) paper [5], Babai, Frankl and Simon defined communication complexity analogues of the standard time/space complexity classes P, NP, RP, $\oplus$P, PSPACE and so on and considered a natural notion of reduction in the communication context under which classes like NP, $\oplus$P and PSPACE have complete problems. This provides, on one hand, a powerful framework to compare the power of various extensions of the usual deterministic model while building, on the other hand, a rich structure of classes in which one can hope to gain intuition on the nature of non-determinism, alternation, randomization and so on.

3.1 Two-Party Models

In the simplest model, two parties (Alice and Bob) collaborate to evaluate a function $f : X \times Y \to R$. Alice is given $x \in X$ while Bob receives $y \in Y$ and their goal is to compute $f(x, y)$ while exchanging as few bits as possible. The two parties communicate according to some previously agreed upon protocol which, informally, is a scheme ensuring that Alice and Bob will never speak simultaneously and will be able to make sense of the information they send each other. Formally, one usually defines a communication protocol $\mathcal{P}$ as a binary tree in which edges are labeled 0 or 1, leaves are labeled by some value in R and every inner node s is either labeled by a function $a_s : X \to \{0, 1\}$ or a function $b_s : Y \to \{0, 1\}$. On input (x, y) one can now picture Alice and Bob following a path from the root down the tree: when a node labeled by some a_s is reached, Alice computes $a_s(x)$ sends the result to Bob and the players follow the edge out of s labeled by $a_s(x)$. When a node labeled b_s is encountered Bob similarly computes $b_s(y)$. The output $\mathcal{P}(x, y)$ of the protocol is simply the label of the leaf at the end of the unique path defined by (x, y) and we say that $\mathcal{P}$ computes f if $\mathcal{P}(x, y) = f(x, y)$ for all (x, y). The *cost* of the protocol is the depth of this tree, i.e. the number of bits exchanged by Alice and Bob on the worst-case input.

The *deterministic communication complexity* of f, denoted $D(f)$ is the cost of the cheapest protocol computing f. In general, we are interested in the complexity of functions $f : \Sigma^* \times \Sigma^* \to R$ and will thus consider $D(f)$ as a function of input length and study its asymptotic behavior. Unless specified

otherwise, we will in fact assume that the inputs given to Alice and Bob are a pair of words of equal length over some finite alphabet.

There are a number of important variations on the deterministic model which have been thoroughly studied and we consider here two of them:

- In a *non-deterministic communication protocol* $\mathcal{P}$ a third player, say God, having access to *both* x and y first sends to Alice and Bob a proof π whose length is a function of the length of x and y. Alice and Bob then follow an ordinary deterministic protocol $\mathcal{P}'$ with output in $\{0,1\}$. The protocol $\mathcal{P}$ accepts the input (x,y) if and only if there is some proof π such that the output of the ensuing deterministic protocol $\mathcal{P}'$ outputs 1. The cost of a non-deterministic protocol is the maximum number of bits exchanged in the protocol (*including* the bits of π) for any input (x,y). We denote the non-deterministic communication complexity of a language L as $N^1(L)$.

- A *Mod$_p$-counting communication protocol* $\mathcal{P}$ is syntactically similar to a non-deterministic protocol but it accepts the inputs (x,y) for which the number of proofs leading Alice and Bob to acceptance is *not* divisible by p. We denote by $N^p(L)$ the Mod$_p$-counting communication complexity of L.

Note that in these two models, protocols are only used to compute languages, i.e. functions with a $\{0,1\}$ output.

Definition 5. *Let* $f : \Sigma^n \times \Sigma^n \to \{0,1\}$ *and* $g : \Gamma^{t(n)} \times \Gamma^{t(n)} \to \{0,1\}$ *be two functions. A* rectangular reduction *of length* t *from* f *to* g *is a pair of functions* $A, B : \Sigma^n \to \Gamma^{t(n)}$ *such that* $f(x,y) = g(A(x), B(y))$.

These reductions are very natural in a communication complexity context since the functions A and B can be computed privately by Alice and Bob respectively. Hence, if there is a rectangular reduction of length $t(n)$ from f to g then $D(g)(t(n)) \geq D(f)(n)$.

If f is a function of two strings x, y of length n over Σ then $D(f) \leq (\log \Sigma) \cdot n + 1$ since it is always possible for Alice to send x to Bob and for Bob to return $f(x,y)$. Thus, Babai, Frankl and Simon defined the class P^{cc} as the class of languages with deterministic communication complexity $O(\log^d n)$ and argued that P^{cc} captures languages with efficient protocols just as P captures languages with efficient algorithms. Following their suggestion, we will drop the *cc* superscript for "convenience and added thrill" and define NP and Mod$_p$P as the class of languages with, respectively, non-deterministic and Mod$_p$-counting communication complexity $O(\log^d n)$. There are in fact natural complete problems for NP and Mod$_p$P. For n-bit strings x, y, we

define the disjointness function as $DISJ(x,y) = 1$ if $x_i y_i = 0$ for $1 \leq i \leq n$ and the inner product modulo p as $IP_p(x,y) = 1$ if $\sum_{i=1}^{n} x_i y_i \equiv 0 \pmod{p}$. If we view x, y as encoding subsets of $\{1, \ldots, n\}$ then $DISJ(x,y) = 1$ if and only if $x \cap y = \emptyset$ and $IP_p(x,y) = 1$ if and only if $|x \cap y|$ is divisible by p.

Theorem 6 ([5, 22]). *The complement of $DISJ$ is NP-complete under rectangular reductions of length $O(2^{\log^d n})$.*
For any prime p, IP_p is Mod_pP-complete under rectangular reductions of length $O(2^{\log^d n})$.

It is not hard to show that $D(DISJ) = \Theta(n)$ and so P $\neq$ NP in the world of communication complexity. Furthermore one can show that $N^1(IP_p) = \Theta(n)$ and, for any prime p, $N^p(DISJ) = \Theta(n)$ and $N^p(IP_q) = \Theta(n)$ for any q not a power of p. So the classes NP and Mod_pP are all incomparable and distinct from P [22].

Strictly speaking, $DISJ$ and IP_q are not regular languages but it is easy to see that they are closely related to the regular languages $\{00, 01, 10\}^*$ and $((\{00, 01, 10\}^* 11 \{00, 01, 10\}^*)^q)^*$ respectively. To further explore the relevance of regular languages in this context, we define the communication complexity of a regular language as the complexity of the following task: Alice and Bob respectively receive $a_1, a_3, \ldots a_{2n-1}$ and $a_2, a_4, \ldots, a_{2n}$ where each a_i is either an element of the finite alphabet Σ or the empty word ϵ and they want to determine whether $a_1 a_2 \ldots a_{2n}$ belongs to L. Similarly, the communication complexity of a finite monoid M is the communication complexity of evaluating in M the product $m_1 \cdot m_2 \cdot \ldots \cdot m_{2n}$ where the odd-indexed $m_i \in M$ are known to Alice and the even-indexed m_i are known to Bob (we will refer to this task as the *word problem for M*). The next lemma provides a key tool for studying the communication complexity of regular languages by relating the complexity of a language with its syntactic monoid.

Lemma 7 ([50]). *If L is a regular language with syntactic monoid $M = M(L)$ then $D(L) = \Theta(D(M))$ and $N^p(L) = \Theta(N^p(M))$ for any prime p.*

It is possible to give a complete classification result for the communication complexity of regular languages in a number of models. In order to discuss the proof of the next theorem, we will need the lower bounds stated earlier about disjointness and inner product as well as a simple lower bound about the function Greater Than defined as $GT(x,y) = 1$ iff $x \geq y$. It is known that the deterministic and Mod_p-counting complexity of GT is $\Theta(\log n)$.

Theorem 8. *[65] Let $L \subseteq A^*$ be a regular language with $M = M(L)$. Then*

$$D(L) = \begin{cases} O(1) & \textit{if and only if } M \textit{ is commutative;} \\ \Theta(\log n) & \textit{if and only if } M \textit{ is in } \mathbf{DO} \cap \overline{\mathbf{Ab}} \textit{ but not commutative;} \\ \Theta(n) & \textit{otherwise.} \end{cases}$$

$$N^p(L) = \begin{cases} O(1) & \textit{if and only if } M \textit{ is commutative;} \\ \Theta(\log n) & \textit{if and only if } M \textit{ is in } \mathbf{LG_p} \circledcirc{M} \mathbf{Com} \\ & \textit{but is not commutative;} \\ \Theta(n) & \textit{otherwise.} \end{cases}$$

Proof sketch. We can take advantage of Lemma 7 to conveniently reason either in language-theoretic or algebraic terms and all these classifications fully exploit this duality. The upper bounds are typically derived through our combinatorial understanding of the relevant classes of regular languages while the semigroup-theoretic tools are most crucial to obtain the matching lower bounds.

For example, it is quite easy to see that $D(L) = O(1)$ if $M(L)$ is commutative. If on the other hand, M contains elements a, b that do not commute then there is an obvious exponential length rectangular reduction from GT to the word problem of M. Given n-bit integers x, y, Alice and Bob can form a string of 2^{n+1} monoid elements $m_1 \dots m_{2^{n+1}}$ where $m_{2i} = a$ if $i = x$ but $m_{2i} = 1_M$ otherwise and $m_{2i-1} = b$ if $i = y$ but $m_{2i-1} = 1_M$ otherwise. Clearly Alice and Bob can respectively compute the even and odd indexed m_i's and the resulting product is ab if $x < y$ but ba if $x \geq y$. Thus in any of the models considered, the communication complexity of a non-commutative monoid is at least log of the complexity of GT.

Let us next consider the pseudo-variety $\mathbf{DO} \cap \overline{\mathbf{Ab}}$: recall from the previous section that the variety of languages corresponding to $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ is the smallest language variety containing the commutative languages and closed under perfectly unambiguous concatenations.

Let L_0 and L_1 be languages of $O(\log n)$ deterministic communication complexity and $L_0 a L_1$ be perfectly unambiguous and assume without loss of generality that $L_0 \subseteq (\Sigma - \{a\})^*$. To determine if an input word w belongs to $L_0 a L_1$, it suffices for Alice and Bob to identify the position of the first a in w and then use the logarithmic cost protocols for L_0 and L_1 to check if the prefix and suffix of that position have the required form. Of course, the position of that initial a can be determined at cost $2 \log n$ by simply exchanging the indices of the first a held by each player and so $D(L_0 a L_1) = O(\log n)$. Since commutative languages have constant deterministic communication complexity we obtain the general logarithmic upper bound for any language with a syntactic monoid in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$.

To obtain the upper bound for the Mod_p-counting complexity of lan-

guages with syntactic monoids in $\mathbf{LG_p} \textcircled{M} \mathbf{Com}$, we use the definition of $\mathrm{Mod}_p\mathrm{Pol}(Com)$. Any such language is the boolean combination of languages of the form

$$\{x | \binom{x}{(L_0 a_1 L_1 \ldots a_k L_k)} \equiv j \pmod{p}\}$$

where each $M(L_i)$ is commutative. We sketch an $O(\log n)$-cost protocol to check if a given word $w \in (A \cup \{\epsilon\})^*$ has a number of factorizations as $u_0 a_1 u_1 \ldots a_k u_k$, with $u_i \in L_i$, that is not congruent to 0 modulo p: The proof chosen in the first step of the protocol consists of k $\log n$-bit integers $t_1 < t_2 < \ldots < t_k$. Next, Alice and Bob interpret the t_i's as possible locations for the bookmarks $a_1, a_2, \ldots, a_k$ in w and accept if they correspond to a valid factorization $u_0 a_1 u_1 \ldots a_k u_k$ with $u_i \in L_i$. The latter can be done at constant cost since Alice and Bob need only check that the position t_i really holds the letter a_i and that the segment u_i belongs to L_i, which requires only $O(1)$ bits since $M(L_i)$ is commutative. The cost of the protocol is dominated by the length of the proof which is $O(\log n)$ and the number of proofs accepted by Alice and Bob is exactly the number of legal factorizations so the protocol accepts if and only if it is non-zero modulo p.

Let us next prove that a monoid containing a non-Abelian group has linear deterministic communication complexity. If G is a non-Abelian subgroup of M there are $a, b \in G$ such that $[a, b] = a^{-1}b^{-1}ab \neq 1_G$. We claim that if $[a, b]$ has order m in G then there is a rectangular reduction of length $O(n)$ from IP_m to the word problem of G. Indeed, if x, y are n-bit vectors Alice and Bob construct for each pair of bits x_i, y_i a four-tuple of group elements $g_{4i-3}g_{4i-2}g_{4i-1}g_{4i}$. Alice sets $g_{4i-3} = a^{-1}$ and $m_{4i-1} = a$ when $x_i = 1$ but $g_{4i-3} = g_{4i-1} = 1_G$ when $x_i = 0$. Bob similarly chooses $g_{4i-2} = b^{-1}$ and $g_{4i} = b$ when $y_i = 1$ but $g_{4i-2} = g_{4i} = 1_G$ when $y_i = 0$. This four-tuple thus evaluates to $[a, b]$ if and only if $x_i = y_i = 1$ and to 1_G otherwise. The product of all such tuples is $[a, b]^r$ with $r = \sum_{1 \leq i \leq n} x_i y_i$ and is therefore equal to 1_G if and only if $IP_m(x, y) = 1$. Thus, in all models considered, the communication complexity of G or any monoid containing G must be at least that of IP_m. Note that for a prime p, the argument actually shows that there exists an m not a power of p such that IP_m reduces to the word problem for G as long as $[G, G]$ is not a p-group. This fact is needed for the linear lower bound in the Mod_p-counting case.

Using the same type of arguments, it is possible to construct a linear length reduction from IP_q to the word problem of any T_q-monoid.

One can also show that there is a length n reduction from Disjointness to the communication problem for the regular language $(ab)^*$. Once again, on input x, y Alice and Bob construct for each pair of bits x_i, y_i of the $DISJ$

instance, a four-tuple $s_{4i-3} \dots s_{4i}$. Alice sets $s_{4i-3} = a$ and $s_{4i-1} = b$ if $x_i = 1$ but $s_{4i-3} = s_{4i-1} = \epsilon$ when $x_i = 0$. Similarly, Bob sets $s_{4i-2} = a$ and $s_{4i} = b$ when $y_i = 1$ but $s_{4i-2} = s_{4i} = \epsilon$ when $y_i = 0$. The resulting four-tuple is $aabb$ if $x_i = y_i = 1$ but otherwise is one of $\epsilon\epsilon\epsilon\epsilon$, $\epsilon a \epsilon b \epsilon$ and $a \epsilon b \epsilon$. It is now clear that the word $s_1 \dots s_{4n}$ belongs to $(ab)^*$ if and only if $DISJ(x, y) = 1$. We can conclude that $D((ab)^*) = \Omega(n)$ and $N^p((ab)^*) = \Omega(n)$ for each prime p. Furthermore, these lower bounds also apply to any monoid divided by the syntactic monoid of $(ab)^*$.

A similar argument provides a linear length reduction from the complement of $DISJ$ to the communication problem associated with the regular language $\{a, b\}^* aa \{a, b\}^*$. If a monoid M is not in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ then either M is a T_p-monoid or it contains a non-Abelian subgroup or it is divided by the syntactic monoid of either $(ab)^*$ or $\{a, b\}^* aa \{a, b\}^*$. In the first two cases we obtain the required lower bound for $D(M)$ through a reduction from IP_p and in the last two cases through a reduction from $DISJ$. ■

In [65], the same approach is used to obtain complete classification results for the two-party communication complexity of regular languages in the probabilistic and non-interactive variants of the two-party model. Interestingly, all the lower bounds are obtained through reductions from four problems (inner product modulo p, disjointness, greater than and indexing) which are among the most-studied examples in communication complexity. In a sense, the classification results retrospectively explain their pivotal importance.

One can also derive the following corollary from elements of the above proof.

Corollary 9. *If L is a regular language then the corresponding communication problem is either in P or hard for one of the classes NP, $co-NP$ or Mod_pP for some prime $p \geq 2$.*

In this sense, regular languages exhibit a very nice behavior with respect to communication complexity and this corollary is reminiscent of results about leaf language classes defined by regular languages [17]. Using an argument closely related to Barrington's theorem (see Section 4) it is also possible to show that there any regular language whose syntactic monoid contains a non-solvable group is complete for the class PSPACE which, in the communication complexity context, is defined using alternation [5].

It can be shown if $\mathcal{V}$ is a variety of languages and p, q are distinct primes then $\mathrm{Mod}_p(\mathcal{V}) \cap \mathrm{Mod}_q(\mathcal{V}) = \mathrm{UPol}(\mathcal{V})$ [46]. This yields:

Corollary 10. *If L is a regular language such that for two distinct primes p and q we have both $N^{MOD_p}(L) = O(\log n)$ and $N^{MOD_q}(L) = O(\log n)$. Then $D(L) = O(\log n)$.*

It is unclear whether this corollary holds only for regular languages: we do not know of any language in $(\mathrm{Mod}_p\mathrm{P} \cap \mathrm{Mod}_q\mathrm{P}) - \mathrm{P}$ and it is quite possible that these classes are in fact equal. For the usual time/space classes one suspects that $\mathrm{Mod}_p\mathrm{P} \cap \mathrm{Mod}_q\mathrm{P} \neq P$ since $\mathrm{Mod}_p\mathrm{P} \cap \mathrm{Mod}_q\mathrm{P}$ at least contains the class UP of unambiguous polynomial time which is believed to be distinct from P. However the communication complexity analogue of UP is in fact equal to P [37] and it has also been shown that $\mathrm{NP} \cap \mathrm{co}\text{–}NP = P$ [40].

It would be very interesting to also obtain a complete classification result for the communication complexity of regular languages in the non-deterministic model. This task however requires the use of more sophisticated algebraic machinery and the key Lemma 7 has to be refined. Indeed, this lemma essentially states that the communication complexity of a regular language in the deterministic or Mod_p-counting models depends on its syntactic monoid. A regular language and its complement share the same syntactic monoid but might have very different non-deterministic communication complexity. To circumvent the problem one needs to consider so-called *ordered syntactic monoids* and *positive varieties* [46]. With the appropriate definitions it is possible to show that the non-deterministic communication complexity of a regular language is exactly that of its ordered syntactic monoid. Let $\mathrm{Pol}(Com)$ denote the class of unions of languages of the form $K_0 a_1 K_1 \dots a_t K_t$, where each K_i is a commutative language. We conjecture that

Conjecture 11. *Let L be a regular language. Then*

$$N^1(L) = \begin{cases} O(1) & \text{if } M(L) \text{ is commutative;} \\ \Theta(\log n) & \text{if } L \in \mathrm{Pol}(Com); \\ \Theta(n) & \text{otherwise.} \end{cases}$$

The $O(\log n)$ upper bound can easily be derived from the definition of the class of languages $\mathrm{Pol}(Com)$. The $\Omega(\log n)$ is also a simple consequence of the non-deterministic complexity of GT. However, the remaining $\Omega(n)$ lower bound has yet to be established and seems quite challenging. It would however provide interesting insight into the non-deterministic communication model.

3.2 Input on Forehead Model

An interesting multiparty extension of the two-party model known as the input on the forehead model was introduced by Chandra, Furst and Lipton in order to prove lower bounds on the length of certain classes of branching programs. In this game, k parties want to compute a function $f(x_1, \dots, x_k)$

and the ith player has access to all inputs *except* x_i so we conveniently picture player i as having x_i written on his forehead. We will denote as $D_k(f)$ the k-party communication complexity of f.

In contrast to the two-party model, the input on the forehead model does not obviously model any sort of real-life situation. Still it has proved to be an extremely useful tool in proving complexity-theoretic lower bounds, most notably for branching programs and boolean circuits. In particular, all languages in the circuit class ACC^0 are known to have polylogarithmic communication complexity when the number of players involved in the game is polylogarithmic. Hence proving sufficiently strong lower bounds in multiparty communication complexity remains an important open problem. Unfortunately, the underlying combinatorics of the model are still poorly understood.

One of the strongest lower bounds available for this model concerns the generalized inner product mod p. For any k, p, we define $GIP_{k,p}$ as the boolean function over k n-bit strings $x^1, \ldots, x^k$ which is 1 if the number of indices i such that $x_i^1 = x_i^2 = \ldots = x_i^k = 1$ is divisible by p. Note that for $k = 2$, GIP is the usual inner product mentioned earlier. Sophisticated discrepancy techniques show that for any fixed k and any p, the k-party communication complexity of $GIP_{k,p} = \Omega(n)$ [6, 28].

As in the two-party case, we define the k-party communication complexity of a regular language L as the number of bits k parties need to exchange to check if a string $a_1 \ldots a_{kn}$ (with $a_i \in \Sigma \cup \{\epsilon\}$) belongs to L when player i has access to all input letters except those with an index congruent to i mod k. Using an analog of Lemma 7, a number of nice facts about the multiparty communication complexity of regular languages can be established [19, 50, 62] although the picture is not as clear as the one obtained in the two-party case.

Theorem 12 ([19, 50, 62])**.** *Let $L \subseteq \Sigma^*$ be a regular language with syntactic monoid $M = M(L)$. If M lies in $\mathbf{DO} \cap \overline{\mathbf{G_{nil}}}$ then there exists a constant k such that $D_k(L) = O(1)$. Otherwise, we have $D_k(L) = \omega(1)$ for all k.*

If M is a group in $\mathbf{G_{nil,k}}$ then $D_{k+1}(L) = O(1)$. If however M is a group outside $\mathbf{G_{nil,k}}$ then $D_{k+1}(L) = \Omega(n)$.

Proof sketch. Let us first argue the second half of the theorem. We know that if M is a nilpotent group of class k then L counts subwords of length k modulo some integer m. In the $k+1$-party model every input letter is accessible to k players and so any k-tuple is seen entirely by at least one player. Therefore, the players can count modulo m the number of occurrences of a subword of length k with communication at most $(k+1) \cdot \lceil \log m \rceil$, a constant, and so $D_k(L) = O(1)$.

For the matching lower bound we know that in the group M is not nilpotent of class k there are elements $g_1, \ldots, g_{k+1}$ such that the iterated commutator $[\ldots[[g_1, g_2], g_3], \ldots, g_{k+1}]$ has order $q > 1$. An argument similar to the one sketched for Theorem 8 then provides a linear length reduction from $GIP_{k+1,q}$ to the word problem for M.

To provide the first upper bound, it suffices to show that if $K \subseteq (\Sigma - \{a\})^*$ and $L \subseteq \Sigma^*$ have bounded k-party complexity for some k then the perfectly unambiguous concatenation KaL has bounded $k+1$-party complexity. To locate the first occurrence of a, each player identifies the party holding that a. Of course the player actually holding it will be incorrect but all other players will agree. Since $k + 1 \geq 3$, this process allows k of the $k + 1$ parties to identify the location of the first a using only $(k + 1) \cdot \lceil \log(k + 1) \rceil$ bits of communication and they can now simulate the k-party protocols for K and L.

Finally, by Lemma 2, it suffices to establish the $\omega(1)$ lower bound for T_q-monoids and for the languages $\{a, b\}^* aa \{a, b\}^*$ and $(ab)^*$. First one can use the definition of T_q-monoids to build for any fixed k a reduction from $GIP_{k,q}$ to the word problem of a T_q-monoid. In similar fashion, it is possible to reduce the k-party generalization of the disjointness function to $\{a, b\}^* aa \{a, b\}^*$, thus yielding an $\Omega(\log n)$ lower bound for the language. The last bound seems to be the most difficult and is obtained through Ramsey-theoretical arguments. ■

In many multiparty communication complexity upper bounds, the fact that any $k - 1$-tuple of input bits is seen entirely by at least one of the k players is used to construct efficient protocols. The above result outlines the importance of this feature: counting subwords of length k modulo m can be done at constant cost by $k + 1$ players while it requires linear (i.e. maximal) cost for k parties. Suppose now that k parties simply want to test if a certain subword occurs in the input, i.e. the players want to test if their input belongs to $\Sigma^* a_1 \Sigma^* a_2 \ldots a_k \Sigma^*$. For $k+1$ parties, the task is trivial since if the subword occurs at all then at least one player can see it completely. It is however conjectured that for any k this same language has unbounded k-party communication complexity. The result has been established by Pudlák [49] for $k \leq 5$ using Ramsey-theoretical methods but remains open in general.

3.3 Characterizations of Languages of Bounded Complexity

In general, we can define the communication complexity of a (not necessarily regular) language $K \subseteq \Sigma^*$ as the complexity of testing if a word w belongs to

K when the input positions are distributed in the worst possible way among the different players. We should note that this is not quite the definition we used for regular languages since we assumed that each input position was either some letter of Σ or the empty string ϵ. Equivalently, Lemma 7 and Theorems 8 and 12 could be stated as holding for the worst-case partition complexity of regular languages with a *neutral letter*. A letter e is said to be neutral for K if for all $u, v \in \Sigma^*$ it holds that $uv \in K \Leftrightarrow uev \in K$ and this in effect what we are imposing by allowing an input letter to be ϵ.

A deep and beautiful theorem of Szegedy provides a surprising characterization of languages that have bounded two-party communication complexity [61].

Theorem 13. *A language K has bounded worst-case partition two-party communication complexity if and only if K can be recognized by a program over a commutative monoid.*

We will define precisely what is meant by "program" in the next section. An alternative statement would be that every such K is reducible to a commutative regular language via a reduction in which each bit of output depends on a single bit of input. Note that these reductions are rectangular reductions.

Since we have a characterization of regular languages with bounded k-party communication complexity for sufficiently large k, it is natural to wonder whether there exists a multiparty analog of Szegedy's theorem. There is in fact reason to believe [19] that it is not possible to obtain a result quite as general as Theorem 13. Nevertheless, one can prove

Theorem 14 ([19])**.** *Let K be a language with a neutral letter such that, for some constant k, the worst-case partition k-party complexity of K is bounded. Then in fact K is a regular language and its syntactic monoid lies in* $\mathbf{DO} \cap \overline{\mathbf{G_{nil}}}$.

Let us (conveniently) leave aside the neutral letter issue. This theorem in some sense characterizes the type of information that k parties can expect to compute for free and thereby solidifies our understanding of the power and limitations of the multiparty model.

4 Boolean Circuit Complexity and Bounded-Width Branching Programs

Historically, the interaction between algebraic automata theory and complexity theory began with Barrington's insight on bounded-width branching

programs and the ensuing characterization of the circuit complexity class NC^1.

Recall that a boolean circuit C_n with n (boolean) inputs $X_1, \ldots, X_n$ is a directed acyclic graph with three types of nodes (or gates): $2n$ input nodes of in degree 0, a single output node of out-degree 0 and inner nodes with in- and out-degree at least 1. The input nodes are labeled with some input X_i or its complement $\overline{X}_i$ while the inner nodes and output node are labeled with a symmetric boolean function chosen from some predetermined base. The depth of C_n is the length of the longest path from an input node to the output and its size is its number of wires (i.e. edges). Such a circuit naturally computes a function of its n inputs. Since circuits only process inputs of some fixed length we use families of circuits $C = \{C_n\}_{n \geq 0}$ so that circuit C_n processes inputs of length n. The size and depth of C are then functions of n.

We recall the definitions of standard circuit complexity classes. A boolean function $f : \{0,1\}^* \to \{0,1\}$ belongs to the class NC^1 if it can be computed by a family of circuits of depth $O(\log n)$ constructed with AND and OR gates of fan-in 2. Furthermore f belongs to ACC^0 if it can be computed by a family of circuits of polynomial size and constant depth constructed with AND, OR and MOD_m gates of arbitrary fan-in, where a MOD_m gate outputs 1 if the sum of its inputs is divisible by m. The classes AC^0 and CC^0 denote the restrictions of ACC^0 in which we allow respectively only AND, OR gates or only the MOD_m gates.

A branching program (or BP) over the variables $x_1, \ldots, x_n$ is a directed acyclic graph whose inner nodes are labeled with a variable x_i. Each inner node has outdegree two and the two outgoing edges are labeled with 0 or 1. A branching program further has a distinguished start node of indegree 0 and two output nodes corresponding to acceptance and rejection. An n-bit input naturally traces a path from the start vertex to one of the output nodes: when a node labeled x_i is reached we follow the edge out of that node labeled by the value of the ith bit of the input. Branching programs, also known as Branching Decision Diagrams (or BDD's) can be used as data structures to represent boolean functions and have been particularly important in automated software and hardware verification [70].

By definition, a BP only processes inputs of some fixed length and, as for circuits, we consider families of BPs $\{B_n\}_{n \geq 0}$ where B_i handles inputs of length n. There have been extensive efforts to prove lower bounds on the size (i.e. number of nodes) or the length (i.e. maximum length path from input to output) of BPs (or restricted classes of BPs) computing certain boolean functions. Interestingly, the most proficient tool for this task seems to be communication complexity [70].

We say that a BP has width k if for every i there are no more than k nodes reachable from the start in i steps. A family of branching programs is said to have bounded-width if there is a k such that all B_i have width k. If we tolerate a moderate blow-up in size, we can assume that the graph underlying a BP of width k consists of ℓ levels each consisting of exactly k nodes and that the different nodes of one level all query the same input bit; we will assume this normal form. The arrows labeled 0 (resp. labeled with 1) between levels i and $i+1$ can simply be thought of as a mapping t_0 (resp. t_1) from $[k]$ to $[k]$. Which of these two transformations is applied of course depends on the value of the input bit queried at that level. Any input to the branching program also naturally defines a mapping from the k points of the first level to the k points of the last level, which is simply the composition of the transformations chosen at each level.

This simple observation naturally leads to an algebraic interpretation of bounded-width BPs known as *non-uniform automata* or *programs over monoid*. An n-input program of length ℓ over M is a sequence of instructions $\phi_n = (i_1, f_1), \ldots, (i_\ell, f_\ell)$ where $1 \leq i_j \leq n$ is the index of some input bit and f_j is a function from $\{0,1\}$ to M. On a given n-bit input $x = x_1 \ldots x_n$, the output of ϕ_n is $\phi_n(x) = f_1(x_{i_1}) \cdot f_2(x_{i_2}) \cdot \cdots \cdot f_\ell(x_{i_\ell})$ where the product is taken in M. The input x is accepted if $\phi_n(x)$ belongs to some target subset $T \subseteq M$. Any branching program of width k can trivially be converted into a program over the finite monoid of transformations on k points. However, the program over monoid formalism allows a finer analysis since the chosen monoid places restrictions on the transformations t_0, t_1 used in the BP.

It was originally believed that any family of bounded width BPs computing the AND of its input bits required exponential or at least super-polynomial length and that polynomial length BPs of width k were strictly less powerful than polynomial length BPs of width $k+1$. Barrington, however proved this intuition incorrect:

Theorem 15 ([7]). *A boolean function can be computed by a family of bounded-width BPs of polynomial length if and only if it belongs to* NC^1 *if and only if it can be computed by a polynomial-length program over any non-solvable group.*

If G is a non-trivial group such that $[G, G] = G$ then it must be simple, non-Abelian and hence non-solvable. The key step in the trickiest part of the above theorem involves showing through induction on depth that for any NC^1 circuit C of depth d and any element $g \in G$, there is a program $\phi_{C,g}$ of length $O(4^d)$ over G whose output is g on inputs accepted by the circuit and 1_G on rejected inputs. Suppose for simplicity that there are elements g_1, g_2 such that $g = [g_1, g_2]$ and that the output gate of the circuit is an AND gate.

The two inputs of this gate are computed by circuits C_1, C_2 of depth $d-1$ and the induction hypothesis gives us programs $\phi_{C_1,g_1}, \phi_{C_2,g_2}$ where $\phi_{C_i,g_i}(x) = g_i$ if C_i accepts x and 1_G otherwise. One can now easily show that the program $\phi_{C,g} = [\phi_{C_1,g_1}, \phi_{C_2,g_2}]$ has the required property.

Since S_5 is non-solvable, it follows that polynomial length BPs of width 5 are as powerful as polynomial length BPs of any fixed width and capture exactly NC^1. Interestingly, some of the most studied subclasses of NC^1 also admit similar algebraic characterizations [9, 10, 12, 41]. In particular, AC^0 , CC^0 and ACC^0 are precisely captured by polynomial length programs over respectively finite aperiodic monoids, finite solvable groups and finite solvable monoids, i.e. monoids whose subgroups are solvable [12]. Recently, the class TC^0 of functions computed by polynomial-size threshold circuits of bounded depth has also been shown to admit an algebraic characterization using infinite groups [39].

These algebraic characterizations can provide useful intuition when reasoning about circuits and were, for instance, instrumental in understanding the computing power of cylindrical circuits [31, 30]. The fact that computing the sum of n bits modulo p requires exponential-size AC^0 circuits [54] is probably the most celebrated result in circuit complexity. The algebraic characterization of AC^0 in terms of group-free monoids was not known when Furst, Saxe and Sipser originally showed a super-polynomial lower bound for AC^0 circuits computing PARITY [25] but in retrospect, this language seems like an ideal candidate since its syntactic monoid is the two-element group.

Many of the open questions about the relative power of these classes can thus be recast in algebraic terms [12, 41, 56]. It is for instance conjectured that CC^0 and ACC^0 are strict subsets of NC^1 and this is equivalent to postulating that polynomial length programs over respectively solvable groups and solvable monoids cannot compute the word problem of a non-solvable group. One advantage of this point of view is that it proposes meaningful intermediate steps towards a proof of such separations. For instance, it has been shown that programs over a nilpotent group cannot compute the n-bit AND, regardless of the program length [11]. The communication complexity results of the previous section already indicated that nilpotent groups are a particularly weak class of solvable groups so such a result is not a real surprise. Much less trivial is the fact that computing AND using programs over the pseudo-variety $\mathbf{G_p} * \mathbf{Ab}$ (generated by the semidirect products of a p-group with an Abelian group) requires exponential length. One can hope to slowly develop lower bound techniques for larger and larger classes of solvable groups to finally obtain the separation of CC^0 and NC^1 . There are a number of natural candidates for the next class of solvable groups for which one might hope to prove program-length lower bounds, including supersolvable

groups and the pseudo-variety $\mathbf{G_p} * \mathbf{G_{nil}}$ generated by semidirect products of p-groups with Abelian groups.

Results on the communication complexity of classes of finite monoids can in fact be used to provide length lower bounds for programs over the same classes. For instance one can show that $\mathbf{G_p} * \mathbf{Ab}$ is a subclass of $\mathbf{LG_p}$ Ⓜ $\mathbf{Com}$ and so any group G in $\mathbf{G_p} * \mathbf{Ab}$ has logarithmic two-party Mod_p-counting communication complexity. It follows that any function computed by a program of length $t(n)$ over G has Mod_p-counting complexity at most $O(\log t(n))$ and in particular, exponential length programs are required to compute functions like IP_q (for q not a power of p) which have linear complexity in this model. It can similarly be shown that a group in $\mathbf{G_p} * \mathbf{G_{nil}}$ has $O(\log n)$ Mod_p-counting k-party communication complexity for some k and so polynomial length programs over these groups can only compute functions also having logarithmic Mod_p-counting k-party complexity. Unfortunately there are currently no known superlogarithmic bounds known for an explicit function in this model.

A slightly different perspective on the algebraic characterizations of subclasses of NC^1 can be given. Using a simple divide and conquer strategy, it is not hard to see that every regular language lies in NC^1 . The surprise is that any regular language whose syntactic monoid contains a non-solvable group is in fact NC^1 -hard under polynomial length projections which are simply reductions in which every bit of output depends on a single bit of input. In that sense, many results about the computing power of polynomial length programs over various classes of monoids can be recast as results on the circuit-complexity of the corresponding classes of regular languages. It should be noted that, as in the communication complexity context, upper bounds invariably follow from our combinatorial understanding of regular languages whereas hardness results or lower bounds usually stem from the algebraic perspective. A number of basic results on the circuit complexity of regular languages can be found in [8, 12, 21].

Simply saying that there are ACC^0 circuits to recognize any regular language with a solvable syntactic monoid or AC^0 circuits for any starfree language is rather vague and it is natural to ask, for instance, how small these circuits can actually be. A general upper bound due to [18] shows that any regular language in AC^0 or ACC^0 can in fact be computed by circuits with only $O(ng^{-1}(n))$ wires for any primitive recursive g.

There are in fact regular languages like the AND or the PARITY function which can be recognized by circuits with a single gate and linearly many wires. Recently, Koucký, Pudlák and Thérien [38] gave a complete characterization of regular languages recognized by AC^0 and ACC^0 circuits with linearly many wires when it is assumed that the language contains a neutral

letter. Once again, the answer turns out to be algebraic.

Theorem 16. *Let L be a regular language with a neutral letter: L can be recognized by an* ACC^0 *circuit with linearly many wires iff $M(L) \in \mathbf{DO} \cap \overline{\mathbf{Ab}}$.*

Proof sketch. As always, the upper bound comes from the combinatorial description of languages with syntactic monoids in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$. It is a simple exercise to show that commutative languages can be recognized with linearly many wires and, once again, the upper bound boils down to showing that if $K \subseteq (\Sigma - \{a\})^*$ and L can be recognized by such circuits then so can the perfectly unambiguous concatenation KaL.

It is of course crucial to locate the first occurrence of a in the input. A nice construction of Bilardi and Preparata [16] gives an AC^0 circuit with linearly many wires with n inputs and n outputs which simultaneously computes for each $i = 1, \ldots, n$ the AND of the first i variables. The combination of this circuit with the linear wire-size circuits computing K and L provide the desired circuit for KaL.

The algebra comes into play for the proof of the matching lower bound. Using results on so-called superconcentrators it is possible to show that if L is a regular language with a neutral letter and deterministic two-party communication complexity $\Omega(n)$ then L cannot be computed using $O(n)$ wires. The theorem can thus be obtained as a corollary to theorem 8. ■

Alternatively, one can measure the size of circuits by counting gates rather than wires and ask what languages can be computed with a linear number of gates. Surprisingly there exist CC^0 circuits with a linear number of gates recognizing any regular language whose syntactic monoid is a solvable group. We conjecture that these are in fact the only regular languages with this property. To establish this fact, it would suffice to show that computing AND in CC^0 requires $\omega(n)$ gates but no such lower bound is known, even though it is suspected that a much stronger bound holds. For AC^0 the most intriguing open question is whether or not the language $\{a, b\}^* aa \{a, b\}^*$ can be computed with $O(n)$ gates when a neutral letter is introduced. It can be shown that this is equivalent to deciding whether or not addition of two n-bit integers can be done in AC^0 with $O(n)$ gates.

5 Learning

We sketch in this section some ideas that seem to be interesting in the development of an algebraic framework for computational learning theory. We will consider the model of exact learning introduced by Angluin [4] although part of our discussion also applies to the classical PAC-learning model of

Valiant. In Angluin's model, a learning algorithm must identify some unknown function f by asking queries to a teacher. The teacher knows f and we assume that he answers queries honestly although, potentially, in adversarial fashion when there is more than one correct answer to a query. There are typically two types of queries considered: *evaluation queries* (the learner asks the value of f on some specific input w and the teacher returns $f(w)$) and *equivalence queries* (the learner proposes a function g as an hypothetical candidate for f and the teacher returns, if possible, an argument w and the value $f(w) \neq g(w)$ thus proving that $f \not\equiv g$). The complexity of the learning algorithm is primarily measured as the number of queries made to the teacher although the time needed for the algorithm to construct these queries is also a concern.

A considerable amount of research has been devoted to the complexity of learning a boolean function taken from some specific class of boolean functions. The difficulty of the task depends of course on the class of functions being learned, the representation of these functions and the types of queries allowed. One of the key open questions in the field is to determine whether or not boolean functions represented in disjunctive normal form can be learned efficiently using evaluation and equivalence queries.

It is not immediately clear whether there is any way to use algebraic automata theory to enlighten this area of research. But since almost all positive results in exact learning involve subclasses of NC^1 (because it seems hopelessly difficult to learn in richer classes) one can consider the problem of learning boolean functions that are represented by an n-input program of polynomial length over some fixed monoid M. The complexity of the learning task clearly depends on the algebraic structure of M. The most obvious benefit of this approach is its ability to analyze algebraically the power and limitations of some broad algorithmic techniques found in the existing literature and to offer a new perspective on these islands of tractability for exact learning. Moreover we can hope to gain, as in the communication complexity setting, some algebraic intuition about what makes a learning task difficult.

Let us first consider the case where only evaluation queries are allowed. If M is not a group or if M is a non-solvable group then for any $w \in \{0,1\}^n$ there exists a polynomial length program over M computing the singleton language $\{w\}$. By a standard adversarial argument we can conclude that 2^n queries are necessary to identify a function represented by a polynomial-length program over M.

We can therefore focus on the case of solvable groups. If G is a nilpotent group then by results of [44] any program over G accepting at least one n-bit string must in fact accept one of Hamming weight at most some constant k_G depending on G which leads to the following simple result.

Lemma 17. *If P and Q are programs over a nilpotent group G that agree on every input of Hamming weight less than k_G then P and Q compute the same boolean function.*

This clearly leads to a learning strategy to identify a boolean function f represented by a program over G using evaluation queries alone: it suffices to ask for the value of f on all $O(n^{G_k})$ inputs of small weight.

In the same spirit, it is possible to show that a program of length m over a group G in $\mathbf{G_p} * \mathbf{Ab}$ cannot compute the AND of more than $O(\log m)$ variables [11]. By an argument similar to the one above, we can therefore identify a function f represented by a program of length m over such groups using only $O(n^{\log m})$ evaluation queries. Note however that although these queries suffice to identify f there is no known way of efficiently constructing a representation of f which would allow one to subsequently evaluate it on some arbitrary input.

One of the most powerful paradigms of exact learning uses so-called *multiplicity automata* or MAs. A multiplicity automaton $\mathcal{A}$ over a ring R and an alphabet Σ is a non-deterministic automaton in which each transition is assigned some ring value. The value of a path in the automaton is the product of the labels on its edges. For any $w \in \Sigma^*$ the value of $\mathcal{A}$ on w is the sum of the values of all paths that the automaton can visit on input w. A powerful result of [14] shows that a function computed by an MA can be learned efficiently using a combination of evaluation and equivalence queries. This algorithm unified a number of previously existing techniques and, since its discovery, has been the starting point for a number of positive results.

In the algebraic context, MAs have been used to learn functions $f : G^n \to G$ which are represented by expressions over G (i.e. programs where an instruction querying the ith element of the input $w \in G^n$ always returns w_i) when G has a normal cyclic group isomorphic to $\mathbb{Z}_p$ and $G/\mathbb{Z}_p \cong \mathbb{Z}_q$ [26]. It would be interesting to further understand which boolean functions represented by programs over a monoid M can be learned using the MA paradigm: preliminary investigations indicate that, once again, the pseudovariety $\mathbf{LG_p} \circledcirc{M} \mathbf{Com}$ is relevant in this context.

The argument that programs over non-groups can recognize arbitrary singletons in $\{0,1\}^n$ does not apply to expressions computing functions $f : M^n \to M$ so an exponential lower bound for learning functions represented by such expressions using only evaluation queries cannot be readily concluded in general. In fact expressions over monoids in $\mathbf{DA}$, the intersection of $\mathbf{DO}$ with aperiodics, can be identified with polynomially many evaluation queries, again using a strategy that focuses on inputs of small Hamming weight [26]. Within aperiodics, the syntactic monoids of $(ab)^*$ and $\{a,b\}^*aa\{a,b\}^*$ are

minimal (with respect to division) outside of **DA** and it is possible to show that learning expressions over either monoid will generally require an exponential number of queries.

Coming back to the boolean case, it can be shown that programs over **DA** can compute precisely the functions which are represented by decision trees of bounded rank [27]. These functions are known to be learnable with a polynomial number of both equivalence and evaluation queries. Functions represented by programs over the syntactic monoid of $(ab)^*$ can also be learned with a polynomial number of queries of the two types. Programs over the syntactic monoid of $\{a,b\}^*aa\{a,b\}^*$ on the other hand can represent DNF formulas with no blow-up in size and so the question of whether such programs can be learned efficiently is one of the central questions of learning theory. A precise characterization of the classes of monoids over which programs are efficiently learnable is yet to come but it is reasonable to think that it would shed interesting light on the exact learning model.

6 Conclusion

Investigations in the complexity of regular languages are greatly facilitated in contexts where the complexity of a language is determined by the algebraic structure of its syntactic monoid. In such cases, communication complexity and circuit complexity being potent examples, it is possible to use the rich tools of algebraic automata theory to make progress. By taking the combinatorial point of view on languages one can usually obtain upper bounds whereas lower bounds and hardness results come from the algebraic perspective.

Applying the semigroup-theoretic approach to study different models of computation has the advantage of uncovering similarities between the strengths and weaknesses of apparently unrelated models. For instance, we saw that pseudo-varieties like **DO** are involved in the dividing line between easy and hard problems in communication complexity, circuit complexity and learning theory.

This approach has been particularly successful in connection with circuit complexity and it should be mentioned that there is a very nice interplay between the logical descriptions of regular languages, the logical description of subclasses of NC^1 and programs [9, 55, 66]. This provides multiple points of view on circuit complexity classes and the interaction of logic and algebra clearly helps in building our intuition about boolean circuits.

There are two obvious ways to extend the program over monoid formalism: by considering programs over more general algebras or by considering

a more powerful mechanism then programs. In one direction, programs over groupoids can be used to capture circuit-complexity classes beyond NC^1 [15]. In the second, the leaf-language framework [1] can provide algebraic characterizations of subclasses of PSPACE [33] or P [35].

Many other cases of applications of finite algebraic structures can be found in computational complexity, including a very powerful approach to understand the complexity of constraint satisfaction problems [34] and algebraic methods have always been part of the complexity theory toolbox. We believe that in many cases semigroup theory can be an elegant and powerful element of this tool set.

Acknowledgements: We want to thank Mark Mercer and Arkadev Chattopadhyay for their comments on a draft of this survey.

References

[1] Leaf language home page. `http://www.thi.uni-hannover.de/forschung/leafl/index.en.php`.

[2] J. Almeida. *Finite Semigroups and Universal Algebra*. Series in Algebra, Vol 3. World Scientific, 1994.

[3] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000.

[4] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[5] L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory. In *Proc. 27th IEEE Symp. on Foundations of Comp. Sci. (FOCS'86)*, pages 337–347, 1986.

[6] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *J. Comput. Syst. Sci.*, 45(2):204–232, 1992.

[7] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. Preliminary version appeared in STOC'86.

[8] D. A. M. Barrington, K. J. Compton, H. Straubing, and D. Thérien. Regular languages in nc^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.

[9] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within nc. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

[10] D. A. M. Barrington and H. Straubing. Superlinear lower bounds for bounded-width branching programs. *J. Comput. Syst. Sci.*, 50(3):374–381, 1995.

[11] D. A. M. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, 1990.

[12] D. A. M. Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35(4):941–952, Oct. 1988.

[13] P. Beame, T. Pitassi, and N. Segerlind. Lower bounds for lovász-schrijver systems and beyond follow from multiparty communication complexity. In *Proc. 32nd Int. Conf. on Automata, Languages and Programming (ICALP'05)*, pages 1176–1188, 2005.

[14] F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

[15] J. Berman, A. Drisko, F. Lemieux, C. Moore, and D. Thérien. Circuits and expressions with non-associative gates. In *IEEE Conference on Computational Complexity*, pages 193–203, 1997.

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

[17] B. Borchert. On the acceptance power of regular languages. *Theor. Comput. Sci.*, 148(2):207–225, 1995.

[18] A. K. Chandra, S. Fortune, and R. J. Lipton. Unbounded fan-in circuits and associative functions. *J. Comput. Syst. Sci.*, 30(2):222–234, 1985.

[19] A. Chattopadhyay, A. Krebs, M. Szegedy, P. Tesson, and D. Thérien. Functions with bounded multiparty communication complexity. submitted for publication, 2006.

[20] J. Cohen, D. Perrin, and J.-E. Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993.

[21] K. J. Compton and H. Straubing. Characterizations of regular languages in low level complexity classes. In *Current Trends in Theoretical Computer Science*, pages 235–246. 2001.

[22] C. Damm, M. Krause, C. Meinel, and S. Waack. On relations between counting communication complexity classes. *J. Comput. Syst. Sci.*, 69(2):259–280, 2004.

[23] S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[24] Z. Ésik and P. Weil. Algebraic recognizability of regular tree languages. *Theor. Comput. Sci.*, 340(1):291–321, 2005.

[25] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. Preliminary version appeared in FOCS'81.

[26] R. Gavaldà, P. Tesson, and D. Thérien. Learning expressions and programs over monoids. *Information and Computation*, 2006. To appear.

[27] R. Gavaldà and D. Thérien. Algebraic characterizations of small classes of boolean functions. In *Proc. of Symp. on Theoretical Aspects of Comp. Sci. (STACS'03)*, 2003.

[28] V. Grolmusz. Separating the communication complexities of MOD m and MOD p circuits. In *Proc. 33rd IEEE FOCS*, pages 278–287, 1992.

[29] V. Grolmusz. The BNS lower bound for multi-party protocols in nearly optimal. *Information and Computation*, 112(1):51–54, 1994.

[30] K. A. Hansen. Constant width planar computation characterizes acc^0. In *Proc. 21st Symp. on Theoretical Aspects of CS (STACS'04)*, pages 44–55, 2004.

[31] K. A. Hansen, P. B. Miltersen, and V. Vinay. Circuits on cylinders. In *14th Symp. on Fundamentals of Comp. Theory (FCT'03)*, pages 171–182, 2003.

[32] J. Håstad and M. Goldmann. On the power of small-depth threshold circuits. In *Proc. 31st IEEE FOCS*, pages 610–618, 1990.

[33] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. Wagner. On the power of polynomial time bit-reductions. In *Conf. on Structure in Complexity Theory*, 1993.

[34] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

[35] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 242–254. IEEE Computer Society Press, 1994.

[36] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Berkeley, 1968.

[37] M. Karchmer, I. Newman, M. E. Saks, and A. Wigderson. Non-deterministic communication complexity with few witnesses. *J. Comput. Syst. Sci.*, 49(2):247–257, 1994.

[38] M. Koucký, P. Pudlák, and D. Thérien. Bounded-depth circuits: separating wires from gates. In *Proc. 37th ACM Symp. on Theory of Computing (STOC'05)*, pages 257–265, 2005.

[39] A. Krebs, K.-J. Lange, and S. Reifferscheid. Characterizing tc^0 in terms of infinite groups. In *Proc. 22nd Symp. on Theoretical Aspects of CS (STACS'05)*, pages 496–507, 2005.

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

[41] P. McKenzie, P. Péladeau, and D. Thérien. NC^1: The automata theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991.

[42] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, Cambridge, Mass., 1971.

[43] N. Nisan. The communication complexity of treshold gates. In *Combinatorics, Paul Erdös is Eighty, Vol. 1*, pages 301–315, 1993.

[44] P. Péladeau and D. Thérien. Sur les langages reconnus par des groupes nilpotents. *C.R. Acad. des Sci. Paris Sér. I Math.*, 306(2):93–95, 1988. English translation by A. Russell and S. Russell appears as TR01-040 of ECCC.

[45] J.-E. Pin. *Varieties of formal languages*. North Oxford Academic Publishers Ltd, London, 1986.

[46] J.-E. Pin. Syntactic semigroups. In G. R. et A. Salomaa, editor, *Handbook of language theory*, volume 1, chapter 10, pages 679–746. Springer Verlag, 1997.

[47] J. E. Pin and P. Weil. Profinite semigroups, Mal'cev products, and identities. *Journal of Algebra*, 182:604–626, 1996.

[48] J. E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory Comput. Systems*, 30:383–422, 1997.

[49] P. Pudlák. An application of Hindman's theorem to a problem on communication complexity. To appear in *Combinatorics, Probability and Computing*, 2003.

[50] J.-F. Raymond, P. Tesson, and D. Thérien. An algebraic approach to communication complexity. *Lecture Notes in Computer Science (ICALP'98)*, 1443:29–40, 1998.

[51] R. Raz and P. McKenzie. Separation of the monotone NC hierarchy. In *Proc. 38^{th}th IEEE FOCS*, 1997.

[52] M. P. Schützenberger. On finite monoids having only trivial subgroups. 8(2):190–194, 1965.

[53] I. Simon. Piecewise testable events. In *Proc. 2nd GI Conf.*, pages 214–222, 1975.

[54] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proc. 19^{th} ACM STOC*, pages 77–82, 1986.

[55] H. Straubing. *Finite Automata, Formal Logic and Circuit Complexity*. Boston: Birkhauser, 1994.

[56] H. Straubing. When can one monoid simulate another? In *Algorithmic Problems in Groups and Semigroups*, pages 267–288. Birkhäuser, 2000.

[57] H. Straubing. Languages defined by modular quantifiers. *Information and Computation*, 166:112–132, 2001.

[58] H. Straubing. On the logical description of regular languages. In *Proc. of the 5th Latin American Theoretical Informatics Conference (LATIN '02)*, 2002.

[59] H. Straubing and D. Thérien. Regular languages defined by generalized first-order formulas with a bounded number of bound variables. In *Proc. of 18th Symp. on Theoretical Aspects of Comp. Sci. conference*, pages 551–562, 2001.

[60] H. Straubing, D. Thérien, and W. Thomas. Regular languages defined by generalized quantifiers. *Information and Computation*, (118):289–301, 1995.

[61] M. Szegedy. Functions with bounded symmetric communication complexity, programs over commutative monoids, and ACC. *J. Comput. Syst. Sci.*, 47(3):405–423, 1993.

[62] P. Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, McGill University, 2003.

[63] P. Tesson and D. Thérien. Diamonds are forever: the variety **DA**. In G. Gomez, P. Silva, and J-E.Pin, editors, *Semigroups, Algorithms, Automata and Languages*. WSP, 2002.

[64] P. Tesson and D. Thérien. Monoids and computation. *International Journal on Algebra and Computation*, 14(5–6):801–816, 2004.

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

[66] P. Tesson and D. Thérien. Restricted two-variable sentences, circuits and communication complexity. In *Proc. 32nd Int. Conf. on Automata, Languages and Programming (ICALP'05)*, pages 526–538, 2005.

[67] D. Thérien. Subword counting and nilpotent groups. In L. Cummings, editor, *Combinatorics on Words: Progress and Perspectives*, pages 195–208. Academic Press, 1983.

[68] D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 256–263, 1998.

[69] D. Thérien and T. Wilke. Nesting until and since in linear temporal logic. *Theory Comput. Syst.*, 37(1):111–131, 2004.

[70] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[71] P. Weil. Closure of varieties of languages under products with counter. *J. Comput. Syst. Sci.*, 45:316–339, 1992.

[72] A. C. Yao. Some complexity questions related to distributive computing. In *Proc. 11^{th} ACM STOC*, pages 209–213, 1979.