LOGICS FOR REGULAR LANGUAGES, FINITE MONOIDS, AND CIRCUIT COMPLEXITY

H. STRAUBING
Computer Science Department
Boston College
Chestnut Hill, Mass.
U.S.A. 02167

D. THÉRIEN
School of Computer Science
McGill University
Montréal, Québec
Canada H3A 2K6

and

W. THOMAS
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
D-24098 Kiel
Germany

Abstract. This paper is an introduction to the definability theory of regular sets of words (regular languages), in which definability in formal logic, the algebraic classification via the notion of syntactic monoid, and recognizability by small-depth boolean circuits are connected. First, the description of the regular and the star-free languages in terms of monadic second-order logic and first-order logic over word models is reviewed. The characterization of first-order definable languages by aperiodic monoids, originally due to Schützenberger and McNaughton, is established using the block product decomposition of finite monoids. Then the framework of circuit complexity theory is introduced, focussing on the circuit complexity class NC^1 and several subclasses (such as AC^0, ACC^0, CC^0). Starting from Barrington's Theorem, characterizations of these classes by polynomial length programs over finite monoids (from appropriate monoid varieties) are presented. Finally, the logical approach is taken up again by a description of the considered subclasses of NC^1 in terms of first-order formulas where nonperiodic numerical predicates are admitted.

The survey addresses readers who are familiar with the fundamentals of semigroup theory. Other prerequisites from formal language theory, logic, and complexity theory are shortly recalled.

Key words: Regular languages, star-free languages, first-order logic, monadic second-order logic, modular quantifiers, syntactic monoid, aperiodic monoid, solvable monoid, block product, monoid decomposition, circuit complexity, small-depth circuits, programs over finite monoids.

1. Introduction

In the sixties, the theory of regular languages developed from two principal sources: First, the notion of *syntactic monoid* opened the possibility of studying regular languages and their properties in the framework of semigroup theory. A basis of this approach is Eilenberg's Variety Theorem which sets up a correspondence between classes of regular languages (respecting certain closure properties) and pseudovarieties of finite monoids. A second source was the view that regular languages are the behaviors of finite automata and that these behaviors can be defined naturally in certain systems of symbolic logic. Büchi and Elgot showed that the system of *monadic second-order logic* characterizes the expressive power of finite automata (and thus the class of regular languages). Both aspects were connected in a line of research opened by Schützenberger and McNaughton, and it was subsequently shown that several sublogics of monadic second-order logic, notably systems of first-order logic, correspond to special properties of syntactic monoids.

In the past ten years, surprising connections between this field and the complexity theory of boolean circuits were discovered. The principal aim in this theory is to evaluate the computational power of boolean circuits in terms of their size, depth, and the types of gates used in their construction. Circuit families of particular interest are those with circuits of small depth and of polynomial size (i.e., with polynomially many gates in the number of input ports), consisting of *AND*- and *OR*-gates and possibly some generalizations of these gates. A family of circuits which contains one circuit for each possible number of input ports can be viewed as a definition of a formal language (to which a 0-1-word belongs if it causes output 1 in the circuit with the appropriate number of input ports). Restrictions in the size and depth of circuits thus constitute corresponding classes of formal languages, and these in turn have been characterized by suitable classes of finite monoids or certain logical systems. As a result, all the different aspects of definability—the algebraic, the logical, and the complexity theoretic one—have now been merged into a unified and appealing theory.

The present lecture notes give an introduction to this rapidly developing field and survey the most important results. In Section 2, we shall present the logical framework for the definition of regular languages and introduce some important specializations. Section 3 is devoted to the characterization of classes of regular languages in terms of the corresponding (varieties of) syntactic monoids. In Section 4, a short introduction to circuit complexity theory is given, together with a survey of the algebraic description of circuit complexity classes (based on "polynomial length programs over finite monoids"). Finally, in Section 5, the logical approach is taken up again; here extensions of the logical systems of the first section are used to describe circuit complexity classes.

While the necessary terminology from formal language theory, logic and circuit theory will be shortly recalled, the reader is assumed to have basic knowledge of semigroup theory.

For a more detailed exposition of the material see the forthcoming monograph [29].

2. Logical Description of Regular Languages

2.1. Regular Languages and Monadic Second-Order Logic

Let us first recall the definiton of regular languages in terms of regular expressions and finite automata. *Regular expressions* over the finite alphabet A are the expressions built up from the symbols a in A (denoting the respective singletons $\{a\}$), $\emptyset$ (denoting the empty language), and ϵ (denoting the singleton containing the empty word) by means of the operations + (for union), $\cdot$ (for concatenation of languages), and the Kleene star * (for the iteration of concatenation). Conventions on bracketing and omission of the concatenation dot are assumed as usual, and often we do not distinguish formally between expressions and denoted languages. So, for instance, we speak of the regular language

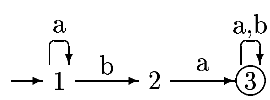
$$L_0 = a^*ba(a+b)^*$$

over the alphabet $A = \{a, b\}$, which contains all words that start with a sequence of letters a followed by a segment ba.

A *(nondeterministic) finite automaton* is a structure $\mathcal{A} = (Q, A, \iota, \Delta, F)$ with a finite set Q of "states", an "initital state" $\iota \in Q$, a "transition relation" $\Delta \subseteq Q \times A \times Q$, and a set $F \subseteq Q$ of "final states". A word $w = a_1 \dots a_n$ is *accepted* by $\mathcal{A}$ if there is a state sequence (a "successful run") $p_1, \dots, p_n, p_{n+1}$ such that

- $p_1 = \iota$,
- $(p_i, a_i, p_{i+1}) \in \Delta$ for $i = 1, \dots, n$,
- $p_{n+1} \in F$.

The language *recognized* by $\mathcal{A}$ consists of the words over A accepted by $\mathcal{A}$. Any finite automaton can be converted to a *deterministic* one (where a function $\delta : Q \times \Sigma \to Q$ replaces the relation Δ) recognizing the same language. By Kleene's Theorem, the languages definable by regular expressions and those recognizable by finite automata coincide. For example, the language L_0 defined by the above mentioned regular expression is recognized by the finite automaton with the states $1, 2, 3$, initial state 1, final state set $\{3\}$, and the transitions $(1, b, 2)$, $(2, a, 3)$ as well as $(1, a, 1)$, $(3, a, 3)$, and $(3, b, 3)$ (see Figure).



A natural alternative for the description of formal languages is to express properties of words in first-order logic. Here one uses variables $x, y, \dots$ that range over positions of letters in a word, and uses formulas such as $Q_a x$ and xSy to express the conditions "position x carries letter a", "position x has position y as successor", respectively. Thus, the language L_0 is defined over the alphabet $A = \{a, b\}$ by the formula

$$\varphi_0 : \ \exists x \exists y (Q_b x \wedge xSy \wedge Q_a y \wedge \forall z (z < x \to Q_a z)).$$

More precisely, we identify a word $w = a_1 \dots a_n$ over A with the corresponding *word model*, namely the relational structure

$$\underline{w} = (\{1, \dots, n\}, S, <, (Q_c)_{c \in A})$$

where $\{1, \dots, n\}$ is the set of "positions", S and $<$ are the usual successor relation and order relation on $\{1, \dots, n\}$, and $Q_c = \{i \in \{1, \dots, n\} \mid a_i = c\}$. The predicates S and $<$, which only depend on the length of w, are called *numerical predicates*, while the unary predicates Q_c are *letter predicates* (coding which positions carry which letters). The corresponding *first-order language* has variables $x, y, z, \dots, x_1, x_2, \dots$

ranging over the positions in word models, and is built up from atomic formulas of the form

$$x = y,\ xSy,\ x < y,\ Q_c x \text{ (for } a \in A)$$

by means of the usual connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and the quantifiers $\exists$ and $\forall$. If $r_1, \ldots, r_n$ are prositions in $\{1, \ldots, n\}$ and $\varphi(x_1, \ldots, x_n)$ is a formula with at most $x_1, \ldots, x_n$ occurring free in φ, then $(\underline{w}, r_1, \ldots, r_n) \models \varphi(x_1, \ldots, x_n)$ means that φ is satisfied in $\underline{w}$ under the mentioned interpretation for S and $<$ and with $r_1, \ldots, r_n$ as interpretation of $x_1, \ldots, x_n$, respectively. A *sentence* is a formula without free variables. The language *defined by* the sentence φ is

$$L(\varphi) = \{w \in A^* \mid \underline{w} \models \varphi\}.$$

Thus, $L(\varphi_0) = L_0$. Let us denote by $FO[S, <]$ the class of languages definable by first-order sentences of this form. Similarly, $FO[S]$ and $FO[<]$ indicate the classes of languages defined by first-order sentences with S, resp. $<$, as the only numerical predicate besides equality.

In defining word properties, it is often convenient to use constants *min* and *max* for the first and last positions of a word. In the sequel we shall allow these constants in order to abbreviate notation. For example, the formulas $Q_a min$ and $xSmax$ are abbreviations for $\exists z(Q_a z \wedge \neg\exists y\ ySz)$ and $\exists z(xSz \wedge \neg\exists y\ zSy)$, respectively. Also the use of S could be eliminated via a definition in terms of $<$, replacing xSy by

$$x < y \wedge \neg\exists z(x < z \wedge z < y).$$

Hence we have $FO[S, <] = FO[<]$. On the other hand, we shall see later that $FO[S]$ forms a proper subclass of $FO[<]$.

We now extend the logical formalism by second-order variables $X, Y, \ldots X_1, \ldots$, which range over sets of positions, together with corresponding atomic formulas $Xx, Xy, \ldots$. Since sets are "monadic second-order objects" , in contrast to relations of higher arity that are polyadic second-order objects, the system is called *monadic second-order logic* (over the signature with the relation symbols $S, <, Q_a$). Note that in this logic, the order relation $<$ becomes definable in terms of successor S: We have (over word models) that $x < y$ is satisfied iff

$$\neg x = y \wedge \forall X(Xx \wedge \forall z \forall z'(Xz \wedge zSz' \rightarrow Xz') \rightarrow Xy).$$

Since $x = y$ is also definable by $\forall X(Xx \leftrightarrow Xy)$, the class $MSO[S]$ of languages definable in monadic second-order logic with S as only numerical predicate is the class of languages definable by monadic second-order sentences including $=$ and $<$. Altogether we obtain

$$FO[S] \subseteq FO[<] \subseteq MSO[S].$$

The key result connecting finite automata with logic states that $MSO[S]$ is the class of regular languages.

Theorem 2.1 (Büchi [10], Elgot [14]) *A language is regular iff it belongs to $MSO[S]$.*

Proof. To show the direction from left to right, let $\mathcal{A} = (Q, A, \iota, \Delta, F)$ be a finite automaton. Assume $Q = \{0, \dots k\}$ where $\iota = 0$. We have to find a monadic second-order sentence that expresses in any given word model $\underline{w}$ (over A) that $\mathcal{A}$ accepts w. Over a word $w = a_1 \dots a_n$, the sentence will state the existence of a successful run $p_1, \dots, p_{n+1}$ of $\mathcal{A}$. We may code such a state sequence up to p_n by a tuple $(X_0, \dots X_k)$ of pairwise disjoint subsets of $\{1, \dots, n\}$ such that X_i contains those positions of w where state i is assumed. From the last state p_n the automaton should be able to reach a final state via the word's last letter a_n. Thus, $\mathcal{A}$ accepts w iff

$$\begin{aligned}\underline{w} \models \exists X_0 \dots \exists X_k \ & (\textstyle\bigwedge_{i \neq j} \neg \exists x (X_i x \wedge X_j x) \\ & \wedge \ X_0 min \\ & \wedge \ \forall x \forall y (xSy \rightarrow \textstyle\bigvee_{(i,a,j) \in \Delta} (X_i x \wedge Q_a x \wedge X_j y) \\ & \wedge \ \textstyle\bigvee_{\exists j \in F : (i,a,j) \in \Delta} (X_i max \wedge Q_a max))\end{aligned}$$

Let us briefly sketch also the proof from right to left. One may proceed in two steps: First we replace first-order variables by a suitable use of second-order variables (which simplifies the logic and hence the translation to automata), and then we construct a finite automaton by induction over these simplified formulas. For the first step, we introduce new types of atomic formulas involving only set variables, namely XSY, $X \subseteq Y$, and $X \subseteq Q_a$ for $a \in A$, with the intended interpretations "X and Y are singletons $\{x\}$ and $\{y\}$ with xSy", "X is subset of Y", "X is subset of Q_a", respectively. It is easy to see that this modified logic, involving only second-order variables and the new atomic formulas, has the same expressive power as the original one. In particular, equality between sets is definable in terms of $\subseteq$, and being a singleton set can be defined by the condition that exactly one proper subset exists. Then first-order variables (and quantifications over them) are captured by variables restricted to singletons and second-order quantifications.

In the second step, we associate to each formula of the modified logic a finite automaton. Since this is done by induction, the case that free set variables occur has to be considered. A formula $\varphi(X_1, \dots, X_m)$ which has free variables $X_1, \dots X_m$ defines a set of words over the extended alphabet $A \times \{0,1\}^m$, where the j-th additional components of the letters determine the interpretation of X_j. (More precisely, the set of positions i of a word model where the j-th additional component is 1 is the interpretation of the set variable X_j.) It is easy to present finite automata that recognize the sets defined by atomic formulas $X_j S X_k$, $X_j \subseteq X_k$, and $X_j \subseteq Q_a$. For the inductive step, it suffices to consider the connectives $\neg, \vee$ and the existential set quantification, since the other connectives and the universal set quantifier are definable in terms of them. This in turn amounts to the proof that the class of regular languages shares certain closure properties, namely closure under complement, under union, and under projection. All these properties are well-known for the class of regular languages. □

¿From the first half of the proof we obtain that regular languages are definable by monadic second-order sentences of a special form, where a prefix of existential second-order quantifiers is followed by a first-order formula. This fragment is called *existential monadic second-order logic*, and *EMSO*$[S]$ is the associated class of definable languages. Combining both parts of the above proof, we obtain $MSO[S] = EMSO[S]$.

In showing Theorem 2.1, the original motivation of Büchi [10] and Elgot [14] was to prove that certain monadic second-order theories are decidable, for example the "weak monadic second-order theory of successor" (which is the set of monadic second-order sentences with S as the only predicate symbol which are true over the domain of the natural numbers, when set variables range only over *finite* sets). By the conversion of formulas into finite automata as described in the proof of the theorem, this decision problem is reduced to the (decidable) emptiness problem for finite automata.

In order to be able to treat more powerful theories, including quantifiers over arbitrary sets of natural numbers, or including a second successor relation (which leads to the infinite binary tree as underlying structure), more general models of finite automata were introduced, working over infinite words or infinite node-labelled trees. Büchi and Rabin succeeded in proving analogues to the theorem above for these generalized finite automata and monadic second-order logic (over the successor structure of the natural numbers, respectively over the binary tree). The nontrivial point here was to establish the necessary closure properties (union, complement, projection) for the automata under consideration, which required intricate constructions. By the same technique, many decidability results for modal logics, temporal logics, and logics of programs have been obtained. For more information on this subject see the survey [38].

If more general ways of quantification or more general underlying structures (than words or trees) are considered, the situation changes radically. Let $SO[S]$ be the class of languages that are definable by general second-order sentences (allowing also quantification over relations of higher arity), and denote by $ESO[S]$ the class of languages definable by existential second-order formulas (where only existential relation quantifiers occur, preceding a first-order formula). As shown by Fagin [15], $ESO[S]$ is equal to the complexity class NP (and hence much larger than the class of regular languages), whereas $SO[S]$ exhausts the polynomial time hierarchy. A similar effect arises even within monadic second-order logic when the underlying structures are no longer words or trees, but finite graphs, i.e., when the successor relation S is replaced by an edge relation E. Already in existential monadic second-order logic one may define graph properties that are NP-complete, for example 3-colorability.

A third way of generalizing Büchi's and Elgot's Theorem is to stay within restricted means of quantification (namely, within first-order logic) and within the domain of word models, but to allow more numerical predicates than S and $<$ in defining formulas. If $\mathcal{P}$ is a class of numerical predicates, then one may consider the class $FO[<, \mathcal{P}]$ of languages which are definable with the numerical predicates in $\mathcal{P}$ besides $<$ and equality. In Section 5 below it will be shown that suitable choices of $\mathcal{P}$ allow us to characterize basic circuit complexity classes.

2.2. First-Order Definability

In this subsection we survey (without proofs) some results on first-order definable languages. We consider the language classes $FO[S]$ and $FO[<]$ and their structure which is derived from a complexity measure of the defining first-order formulas. The measure is based on the prenex normal form of formulas, in which quantifiers of the same type (existential or universal) are grouped into blocks. If there are n such blocks starting with existential quantifiers, we speak of a Σ_n-formula. A Σ_n*-sentence* is thus of the form

$$\exists \overline{x}_1 \forall \overline{x}_2 \ldots \exists/\forall \overline{x}_n \varphi(\overline{x}_1, \ldots, \overline{x}_n)$$

where $\overline{x}_1, \ldots, \overline{x}_n$ are nonempty tuples of variables, the quantifier blocks alternate between existential and universal ones, and φ is quantifier-free. A Π_n*-sentence* has n quantifier blocks starting with a universal one. A boolean combination of Σ_n-formulas will be called a $B(\Sigma_n)$-formula.

In the context of monadic second-order quantifiers and with *first-order* formulas as "kernels" φ, we know from the result $MSO[S] = EMSO[S]$ of the previous section that the Σ_n-form can be reduced to Σ_1-form. Let us consider this question for first-order formulas with numerical predicates S and $<$, respectively.

The situation for first-order logic with successor S is illustrated by the following example sentences, to be interpreted in word models over the alphabet $A = \{a, b\}$:

$$\varphi_1 : \ \forall x \forall y((xSy \wedge Q_a x \wedge Q_b y) \rightarrow \exists z(ySz \wedge Q_b z))$$

$$\varphi_2 : \ \exists x(Q_a x \wedge \forall y(xSy \rightarrow \exists z(ySz \wedge Q_a z)))$$

Converted to prenex normal forms these sentences yield quantifier prefixes of type Π_2 and Σ_3, respectively. It is easy to see that a word satisfies φ_1 iff it does not end with ab and has no segment aba. Similarly, a word satisfies φ_2 iff it has a segment $a * a$, where $*$ indicates a or b, or (in the case that x has no successor) the word ends with a. Such conditions on the existence of segments, suffixes, or prefixes can be formulated directly by $B(\Sigma_1)$-sentences when we allow the constants *min* and *max*. More general conditions that could be formalized by $B(\Sigma_1)$-sentences involve *multiple* occurrences of segments; for example the condition that at least two segments ab occur is formalizable by

$$\exists x \exists y \exists x' \exists y'(xSy \wedge Q_a x \wedge Q_b y \wedge \neg x = x' \wedge x'Sy' \wedge Q_a x' \wedge Q_b y')$$

As it turns out, *all* first-order sentences in the signature of the successor relation S together with *min* and *max* can be reduced to boolean combinations of Σ_1-sentences.

There is a more combinatorial formulation of this reduction result. For a word $w \in A^*$, a segment $\sigma \in A^+$, and a natural number t let

$$\mathrm{occ}_{\sigma,t} := \begin{cases} \text{number of occurrences of } \sigma \text{ in } w & \text{if this number is } < t \\ t & \text{otherwise} \end{cases}$$

Furthermore, let $\mathrm{pref}_d(w)$, resp. $\mathrm{suf}_d(w)$ be the segment of the first, resp. last, d letters of w (or w itself if its length is $\leq d$). Now define, for $u, v \in A^*$ and $d, t \geq 0$,

$$u \sim_{d,t} v$$

iff for all segments σ of length $\leq d$ we have $\mathrm{occ}_{\sigma,t}(u) = \mathrm{occ}_{\sigma,t}(v)$, $\mathrm{pref}_d(u) = \mathrm{pref}_d(v)$, and $\mathrm{suf}_d(u) = \mathrm{suf}_d(v)$. Then $\sim_{d,t}$ is a congruence of finite index, and we say that a language L is *locally threshold testable* if L is a (finite) union of $\sim_{d,t}$-classes for some fixed d and t. Thus membership of a word in a locally threshold testable language depends only on the prefix and suffix of a given length and on the existence of segments of a given length, counted up to a fixed threshold.

Theorem 2.2 (Thomas [35], [36]) *The following are equivalent:*
(a) L is in FO[S],
(b) L is definable by a $B(\Sigma_1)$-sentence of first-order logic with numerical predicate S and the constants min and max.
(c) L is locally threshold testable.

The implications from (c) to (b) and from (b) to (a) are immediate. For the step from (a) to (c), the "Ehrenfeucht-Fraïssé technique" can be applied. For an introduction see e.g. [39].

¿From this theorem we obtain languages which belong to $FO[<]$ but not to $FO[S]$. For example the language

$$a^*ba^*ca^*$$

is clearly first-order definable in terms of $<$ but easily seen to be not locally threshold testable. However, this leaves unsettled the question how to decide *effectively* for a given regular language (presented say by a regular expression) whether it belongs to $FO[S]$ or $FO[<]$. In the next section effective procedures are derived from semigroup theoretical characterizations of $FO[S]$ and $FO[<]$.

We now turn to the class $FO[<]$. As shown by McNaughton and Papert [22], it can be characterized by a variant of the calculus of regular expressions, the "star-free expressions", where the Kleene star * is left out but a symbol $\sim$ for complementation (with respect to A^* for the given alphabet A) is included. As an example over the alphabet $A = \{a, b, c\}$ consider again the language $a^*ba^*ca^*$. It can be defined without use of the star * via a represention of a^* in the form

$$\sim (A^* \cdot b \cdot A^* \ + \ A^* \cdot c \cdot A^*)$$

where A^* stands for the star-free expression $\sim \emptyset$. For technical reasons we will use the concatenation dot only via alphabet letters as in this example, i.e. we allow the formation of an expression $r_0 \cdot a_1 \cdot r_1 \cdot a_2 \cdots a_n \cdot r_n$ from expressions $r_0, \ldots, r_n$ and letters $a_1, \ldots, a_n$. Furthermore, the atomic star-free expressions are restricted to $\emptyset$ alone. A *star-free expression* is any expression obtained from $\emptyset$ by application of the boolean operations $\sim$, $+$ and concatenation via letters. A language over a given alphabet is called *star-free* if it is definable by a star-free expression. (It is not difficult to show that our version of star-free expressions, originating in Straubing [30] and Thérien [33], leads to the same class of languages as the classical version that starts from the atomic expressions $\emptyset$, ϵ, $a \in A$, and includes the boolean connectives $\sim$ and $+$ as well as concatenation products $r_0 \cdot r_1 \cdots r_n$. For instance, concerning the atomic expressions ϵ and a, note that $\{\epsilon\}$ is obtainable by subtracting all languages A^*aA^* from A^*, i.e., is definable by a boolean combination of expressions $\sim \emptyset$ and $\sim \emptyset \cdot a \cdot \sim \emptyset$, and that a language $\{a\}$ is definable via the representation $\{\epsilon\} \cdot a \cdot \{\epsilon\}$.)

Theorem 2.3 (McNaughton, Papert [22]) *A language is star-free iff it belongs to $FO[<]$.*

There is a tight correspondence between first-order formulas with $<$ and star-free expressions, based on the classification of formulas by number of quantifier alternations and the classification of star-free expressions by their "dot-depth".

Define the *dot-depth* $\mathrm{dd}(r)$ of star-free expressions r inductively as follows:

1. $\mathrm{dd}(\emptyset) = 0$
2. $\mathrm{dd}(\sim r) = \mathrm{dd}(r)$
3. $\mathrm{dd}(r + s) = \max\{\mathrm{dd}(r), \mathrm{dd}(s)\}$
4. $\mathrm{dd}(r_0 \cdot a_1 \cdot r_1 \cdots a_m \cdot r_m) = \max\{\mathrm{dd}(r_1), \ldots, \mathrm{dd}(r_m)\} + 1$

The *dot-depth of a star-free language* L is defined to be the smallest dot-depth of a star-free expression defining L. The classes $\mathcal{V}_n$ of star-free languages of dot-depth at most n form the *dot-depth hierarchy* (in the sense of Straubing and Thérien). The example language $a^*ba^*ca^*$ belongs to $\mathcal{V}_2$; note that the definition of a^* given above is of dot-depth 1.

Theorem 2.4 (Straubing [31], Thomas [37], Perrin, Pin [23])

1. *The dot-depth hierachy is strict, i.e., for $n \geq 0$ the class $\mathcal{V}_n$ is properly included in $\mathcal{V}_{n+1}$.*
2. *For $n \geq 1$, a language belongs to $\mathcal{V}_n$ iff it is definable by a $B(\Sigma_n)$-sentence with $<$ as numerical predicate.*

This result shows that, in contrast to the preceding theorems, a reduction of quantifier alternation depth to existential formulas fails for first-order logic with $<$.

There is a similar (and historically earlier) theorem for the version of star-free expressions with atomic expressions including ϵ and letters a and with concatenation products of the form $r_0 \cdot r_1 \cdots r_n$. The resulting hierarchy was introduced by Cohen and Brzozowski in [11] and is often called "Brzozowski hierarchy". The strictness of this hierarchy was shown by Brzozowski and Knast [9], whereas the logical characterization (in fact, in terms of first-order formulas including the constants *min* and *max*) was given by Thomas [36].

In the next section we shall consider languages which are not star-free (i.e., fall outside $FO[<]$) but are still regular. Such examples can be obtained by conditions which involve counting modulo a fixed number (*modular counting* for short). For example, the set of words of even length is regular but not star-free. This motivates the extension of first-order logic by features which allow us to express conditions on modular counting. Two such extended logics are introduced in Examples 3 and 4 below.

3. Regular Languages and Finite Monoids

In this section we shall discuss further classes of regular languages (besides $FO[S]$ and $FO[<]$) and characterizations of language classes by properties of syntactic monoids. The *syntactic monoid* of a language $L \subseteq A^*$ is the quotient monoid $M(L) = A^*/\sim_L$ where $\sim_L$ is the *syntactic congruence* over A^* defined by

$$w \sim_L w' \quad \text{iff} \quad \forall u, v \in A^*(uwv \in L \Leftrightarrow uw'v \in L).$$

The *syntactic morphism* of L is the projection μ_L of A^* onto $M(L)$.

We begin with some examples.

3.1. Examples

Example 1. Let

$$L_1 = (a + b + c)^* aa(a + b + c)^*.$$

which is defined by the first-order sentence

$$\exists x \exists y (xSy \wedge Q_a x \wedge Q_a y).$$

and hence belongs to $FO[S]$. A simple computation shows that the syntactic monoid of this language has six elements, consisting of the identity, a zero (which is the image of a^2 under the syntactic morphism) and a four-element regular $\mathcal{J}$-class with three idempotents (the images of b, ab and ba) and a null element (the image of a).

Example 2. Let

$$L_2 = a^* ba^* ca^*.$$

(considered already after Theorem 2.2) which is defined by the sentence

$$\exists x \exists y ((x < y \wedge Q_b x \wedge Q_c y) \wedge \forall z ((\neg z = x \wedge \neg z = y) \rightarrow Q_a z)).$$

This sentence uses order and equality as the only numerical predicates; thus $L_2 \in FO[<]$. The syntactic monoid consists of an identity element, which is the image of the letter a, together with a four-element nilpotent semigroup, whose nonzero elements are the images of b, c and bc.

Example 3. Let

$$L_3 = \{w \in \{a, b\}^* : |w| \equiv 0 \pmod 2\}.$$

We can write a first-order sentence that defines L_3 by introducing numerical predicates

$$x \equiv 0 \pmod k,$$

for $k > 1$. This predicate is interpreted to mean, 'x is a position divisible by k', where the positions are numbered from left to right, beginning with 1. L_3 is then defined by the sentence

$$\forall x (\neg \exists y (x < y) \ \rightarrow \ (x \equiv 0 \pmod 2)),$$

which says, in effect, that if there is a final position in the string, then that position has an even number. We denote by $FO[<, \{x \equiv 0 \pmod k\}]$ the family of languages defined by first-order sentences in which we admit both the ordering and these new predicates as numerical predicates. Hence $L_3 \in FO[<, \{x \equiv 0 \pmod k\}]$. The syntactic monoid of L_3 is the cyclic group of order 2; all words of even length are mapped by the syntactic morphism to the identity of the group, and all words of odd length are mapped to the generator.

Example 4. Let L_4 be the set of words over the alphabet $\{a, b\}$ in which there are an even number of occurrences of the letter a. As in the preceding example, the syntactic monoid is the group of order 2, however the syntactic morphism is different. In the present example the letter a is mapped to the generator of the group and the letter b to the identity. We define L_4 by the sentence

$$\exists^{(2,0)} x Q_a x.$$

The symbol $\exists^{(2,0)}$ is a *modular quantifier.* It is interpreted to mean 'there are exactly 0 mod 2 positions such that...' Thus the sentence says precisely that the number of positions with the letter a is even. We will use the notation $MOD_k[...]$ analogously to $FO[...]$ to denote the classes of languages defined by sentences with modular quantifiers of modulus k with specified numerical predicates. We also use $(FO + MOD_k)[..]$ to denote the classes defined when both the ordinary quantifiers and modular quantifiers are available. Thus $L_4 \in MOD_2[\emptyset]$, since we did not use any numerical predicates in the defining sentence.

What are the inclusion relations among these language classes, and how can we effectively decide to which classes a given regular language belongs? First, it is straightforward to verify that we have the inclusion chain

$$FO[S] \subseteq FO[<] \subseteq FO[<, \{x \equiv 0 \pmod k\}] \subseteq (FO + MOD_k)[<] \subseteq MSO[S].$$

To answer the question on the strictness of these inclusions and to obtain effective tests for determining if a given regular language belongs to any of these classes, we shall present semigroup theoretic characterizations of all these classes.

3.2. The Key Theorem

In the next two sections we will outline a proof of the following theorem.

Theorem 3.1 *Let $L \subseteq A^*$. $L \in FO[<]$ if and only if L is regular and $M(L)$ is aperiodic.*

In particular, this theorem implies that the languages L_3 and L_4 of the examples in the preceding section are not in $FO[<]$, since their syntactic monoids are groups.

This theorem is actually a consequence of two different results: One, due to McNaughton and Papert [22] and mentioned above (Theorem 2.3), characterizes $FO[<]$ as the family of *star-free* regular languages—those that can be expressed using only boolean operations and concatenation— and the other, due to Schützenberger [26], characterizes the star-free regular languages as those with aperiodic syntactic monoids.

We will use some heavy algebraic machinery to prove this theorem. There are, to be sure, simpler proofs than the one we outline here, but the argument we give has the advantage of generalizing to treat different classes of quantifiers and numerical predicates.

3.3. The Block Product

Let M and N be monoids. The *block product* of N and M, denoted $N \;\square\; M$, is the set $N^{M \times M} \times M$, together with a multiplication given by

$$(F_1, m_1)(F_2, m_2) = (F, m_1 m_2),$$

where for all $m, m' \in M$,

$$F(m, m') = F_1(m, m_2m')F_2(mm_1, m').$$

It is straightforward to verify that this multiplication is associative, and that $(I, 1)$, where $I(m, m') = 1$ for all $m, m' \in M$, is the identity element. Thus $N \,\square\, M$ is itself a monoid. The map

$$(F, m) \mapsto m,$$

is a homomorphism from the block product onto M. It is not hard to show that if G is a group contained in $N \,\square\, M$, then the kernel of the restriction of this homomorphism to G is isomorphic to a group in the direct product of $|M|^2$ copies of N. In particular, *the block product of two aperiodic monoids is aperiodic.* By the way, the block product is *not* an associative operation, and it most certainly is not a commutative operation.

The definition of the block product is a bit obscure, so let us try to provide a way to understand what it's all about. Let us say that a homomorphism $\phi : A^* \to K$, where K is a monoid, *decomposes* with respect to M and N if there exist homomorphisms

$$\alpha : A^* \to M,$$

and

$$\beta(M \times A \times M)^* \to N$$

such that the value of ϕ at a word $a_1 \cdots a_r$ is determined by

$$\alpha(a_1 \cdots a_r)$$

and

$$\beta(1, a_1, \alpha(a_2 \cdots a_r))\beta(\alpha(a_1), a_2, \alpha(a_3 \cdots a_r)) \cdots \beta(\alpha(a_1 \ldots a_{r-1}), a_r, 1).$$

One can then show that if ϕ decomposes with respect to M and N, then ϕ factors through a homomorphism $\psi : A^* \to N \,\square\, M$, and that if ϕ factors through a homomorphism into the block product, then ϕ decomposes with respect to M and $N^{M\times M}$. It is this property of the block product that we will use in the analysis of logical formulas.

The *Krohn-Rhodes Theorem* is a general decomposition theorem for finite monoids. Although it is usually stated in terms of the wreath product, there is a version for block products that we will use in the next subsection:

Theorem 3.2 *Every homomorphism ϕ from A^* into a finite monoid K factors through an iterated block product*

$$M_r \,\square\, (M_{r-1} \,\square\, \cdots \,\square\, (M_2 \,\square\, M_1) \cdots),$$

where each M_i is either a simple group that divides K or the monoid $U_1 = \{0, 1\}$.

The block product, along with this version of the Krohn-Rhodes theorem, is described in Rhodes and Tilson [25].

3.4. How the Key Theorem is Proved

We will merely sketch the main points of the argument. If $M(L)$ is aperiodic, then by the Krohn-Rhodes Theorem, the syntactic morphism μ_L factors through a block product

$$U_1 \ \Box \ (U_1 \ \Box \ \cdots \ \Box \ (U_1 \ \Box \ U_1)\cdots).$$

Let us denote this iterated block product by $U_1^{[r]}$. It follows from our remarks concerning decomposition, and the fact that U_1 is idempotent and commutative, that there is a homomorphism

$$\alpha : A^* \to U_1^{[r-1]}$$

such that the image of μ_L at $w \in A^*$ is determined by $\alpha(w)$ and by the set of triples

$$\{(\alpha(w'), a, \alpha(w'')) : w = w'aw''\}.$$

This is used to prove by induction on r that for all homomorphisms $\theta : A^* \to U_1^{[r]}$, the sets $\theta^{-1}(k)$, $k \in U_1^{[r]}$ are in $FO[<]$, and thus L, which is a finite union of such sets, is in $FO[<]$.

For the converse, we must show that if $L \in FO[<]$, then $M(L)$ is aperiodic. We will interpret formulas with free variables in $\{x_1, \ldots, x_k\}$ in *word structures,* which we view as words over the extended alphabet $A \times 2^{\{x_1,\ldots,x_k\}}$. A formula θ with free variables thus defines a language over this extended alphabet. For example, if θ is $Q_a x_1$, then L_θ is the language

$$A^*(a, \{x_1\})A^*.$$

If θ is $x_1 < x_2$, then L_θ is

$$A^*(A \times \{\{x_1\}\})A^*(A \times \{\{x_2\}\})A^*.$$

We show by induction on the construction of a formula θ of $FO[<]$ that $M(L_\theta)$ is aperiodic. This is easy to show for the atomic formulas, and it is also easy to show that aperiodicity is preserved under boolean operations. The heart of the argument is the proof that aperiodicity is preserved under quantification: Let θ be a formula with free variables in $\{x, x_1, \ldots, x_k\}$, and suppose that $M(L_\theta)$ is aperiodic. Let ζ be the formula $\exists x\theta$. We denote by μ_θ and μ_ζ the syntactic morphisms of L_θ and L_ζ, respectively. Now μ_ζ decomposes with respect to $M(L_\theta)$ and U_1, since we can take $\alpha = \mu_\theta$, and set

$$\beta(m, (a, X), m') = 0$$

if and only if

$$m \cdot \alpha(a, X \cup \{x\}) \cdot m' \in \mu_\theta(L_\theta).$$

Thus μ_ζ factors through $N \ \Box \ M(L_\theta)$, where N is a direct product of copies of U_1. It follows from the inductive hypothesis and our remarks concerning groups in block products that $M(L_\zeta)$ is aperiodic.

3.5. Characterizations of Other Language Classes

The preceding result can be extended in a number of different ways. The first extension characterizes the language classes defined with modular quantifiers and ordering:

Theorem 3.3 *Let $L \subseteq A^*$ be a regular language. Then $L \in MOD[<]$ if and only if $M(L)$ is a solvable group, and $L \in (FO + MOD)[<]$ if and only if every group in $M(L)$ is solvable.*

This result is due to the authors [32]. The argument used to prove this is virtually identical to the one used to prove the characterization of $FO[<]$. The crucial point is that modular quantification with modulus q is equivalent to formation of the block product with $\mathbf{Z}_q$ on the left, in the same sense that existential quantification is equivalent to forming the block product with U_1 on the left. The Krohn-Rhodes Theorem implies that every homomorphism into a monoid containing only solvable groups factors through a block product whose factors are copies of U_1 and cyclic groups, and this is used to show that if $M(L)$ contains only solvable groups then $L \in (FO{+}MOD)[<]$. Conversely, if $L \in (FO{+}MOD)[<]$, then we use the properties of groups contained in block products to conclude that every group in $M(L)$ is solvable. For the characterization of $MOD[<]$ we need the additional facts that the block product of two groups is a group, and that a homomorphism into a group G factors through an iterated block product all of whose factors are the simple composition factors of G.

Quite similar techniques are used to show the following two theorems, due to Barrington, Compton, Straubing and Thérien [3]:

Theorem 3.4 *Let $L \subseteq A^*$ be a regular language. Then*

$$L \in FO[<, \{x \equiv 0 \pmod{k} : k > 1\}]$$

if and only if for all $t > 0$, $\mu_L(A^t)$ (the image of A^t by syntactic morphism μ_L of L) contains no nontrivial group.

This shows that the language L_4 of the examples in 1.1 is not in

$$FO[<, \{x \equiv 0 \pmod{k} : k > 1\}].$$

There is also a modular version of this theorem:

Theorem 3.5 *Let $L \subseteq A^*$ be a regular language. Then*

$$L \in (FO + MOD_q)[<, \{x \equiv 0 \pmod{k} : k > 1\}]$$

if and only if for all $t > 0$, every group in $\mu_L(A^t)$ is solvable, and has cardinality that divides a power of q.

Somewhat different arguments are used to characterize the classes in which successor and equality are the only numerical predicates:

Theorem 3.6 *Let $L \subseteq A^*$ be regular. Then $L \in FO[S]$ if and only if $M(L)$ is aperiodic, and for all $e, s, f, t, u \in \mu_L(A^+)$, with e, f idempotent, $esfteuf = euftesf$.*

This result was first observed by Beauquier and Pin [7] in the context of infinite words. It follows from a Theorem 2.2, which characterizes $FO[S]$ as the class of 'locally threshold testable' languages, and the algebraic characterization of this class, which is a result of Thérien and Weiss [34]. A corollary is that the language L_2 (from Example 2 of Section 3.1) is not in $FO[S]$, because the identity element of $M(L_2)$ is an idempotent of $\mu_{L_2}(A^+)$, and we get a contradiction to the condition in the theorem by taking $e = f = t = 1$, $s = \mu_{L_2}(b)$ and $u = \mu_{L_2}(c)$. In particular, the order relation cannot be defined by a first-order formula using only successor and equality.

In fact, there are effective algebraic characterizations for all the classes formed by various combinations of first-order and modular quantifiers and the numerical predicates discussed here. For details, see the forthcoming book by Straubing [29].

3.6. A Question

Up until now the numerical predicates we have considered give rise only to regular languages when used in defining sentences. Of course one can consider sentences with such numerical relations as '$y = 2x$', or 'y is the Gödel number of a Turing machine that halts when started on the binary representation of x'. These will give rise to nonregular languages (in the first instance) and nonrecursive languages (in the second). So let us ask the question posed at the end of 1.1 in this wider setting: Are there *any* numerical predicates we can introduce so that L_4, the set of strings with an even number of occurrences of the letter a, is first-order definable? We shall see in Section 5 that the answer to this very difficult question is 'No', and that this reveals interesting connections between logic, the theory of finite automata, and circuit complexity.

4. Circuit Complexity and Finite Monoids

Boolean circuits are now extensively studied as a model for parallel computations. In recent years, deep connections have been found between certain classes of circuits and computations taking place in finite monoids. In this section, we will present some results in this area: we assume that the reader has more knowledge of algebra than of computational complexity.

4.1. Computational Complexity

For both practical and theoretical reasons, computer scientists are interested in classifying problems according to the amount of computing resources required to get their solution. The classical model for sequential computations is the Turing machine and the usual complexity measures are time and space. A language L is recognizable in time $t(n)$ if there exists a Turing machine that determines membership in L using at most $t(n)$ steps on any input of length n; L is recognizable in space $s(n)$ if there exists a Turing machine that determines membership in L while visiting at most $s(n)$ memory cells on any input of length n. Two typical complexity classes definable in this framework are $LOGSPACE$, the family of languages recognizable in deterministic logarithmic space, and $PTIME$, the family of languages recognizable in deterministic polynomial time: an important open question is to determine if the easily shown inclusion $LOGSPACE \subseteq PTIME$ is strict or not. A standard reference for this material is [18].

The development of parallel technology has led to the introduction of new computing models. A formalism commonly considered for parallel computations is that of boolean circuit. An n-input boolean circuit C_n is given by a vertex-labeled directed acyclic graph where

- vertices have fan-in 0 or 2
- vertices of fan-in 0 (the input gates) have their labels in $\{1, 0, X_1, \ldots, X_n, \overline{X_1}, \ldots, \overline{X_n}\}$
- vertices of fan-in 2 (the internal gates) have their labels in $\{AND, OR\}$
- there is a unique vertex of fan-out 0 (the output gate)

Such an object computes a function $C_n : \{0,1\}^n \to \{0,1\}$ in a natural way. Given $x \in \{0,1\}^n$, an input gate labeled $X_i(\overline{X_i}, 1, 0)$ returns the value of the i^{th} bit of x (the complemented value of the i^{th} bit of x, 1, 0); an AND gate returns 1 iff both edges entering it carry the value 1; an OR gate returns 1 iff at least one edge entering it carries the value 1; finally the value of $C_n(x)$ is the bit returned by the output gate.

To recognize subsets of $\{0,1\}^*$, we consider families of circuits, $C = (C_n)_{n \geq 0}$, and the words of length n accepted by C are precisely those satisfying $C_n(x) = 1$. We say that L is recognized in depth $s(n)$ iff there exists a circuit family $C = (C_n)_{n \geq 0}$ accepting L such that the number of vertices in C_n is $s(n)$; we say that L is recognized in size $d(n)$ iff there exists a circuit family $C = (C_n)_{n \geq 0}$ accepting L such that the length of the longest path in C_n is $d(n)$.

Circuits are meant to simulate parallel computations since all gates at a given level operate simultaneously. Thus circuit depth measures parallel time. Also we identify parallel space with circuit size. It is interesting to note that, with the above definitions, sequential space is polynomially related to parallel time and sequential time is polynomially related to parallel space. For a discussion of this fact and of circuits in general, the reader may consult [12].

We end this introduction with some remarks.

1. What we have defined here is non-uniform families of circuits, that is no condition on the algorithmic definability of $C = (C_n)_{n \geq 0}$ is imposed. In particular the model allows non recursively enumerable languages to be recognized. The remedy is to require some condition on how to construct the graph C_n; for example, it may be asked that, given 1^n, some log-space bounded Turing machine is able to produce a description of C_n. Barrington, Immerman and Straubing [4] offer a detailed study of various uniformity criteria; our presentation will concentrate exclusively on the non-uniform model.
2. One can consider circuits over an arbitrary input alphabet A by allowing input gates to be labeled by "$X_i = a$" for any $a \in A$.
3. We do not distinguish between a class $\mathcal{C}$ of circuits and the class of languages recognizable by circuits in $\mathcal{C}$. The context makes clear which is intended.
4. By allowing circuits to have several output gates, we can investigate computation

of functions instead of simply recognition of languages.

4.2. NC^1 AND ITS SUBCLASSES

The circuit class NC^1 that we will be interested in consists, by definition, of those circuit families $C = (C_n)_{n\geq 0}$ of $O(\log n)$ depth (we have $d(n) = O(\log n)$ iff there is some constant c such that for all n $d(n) \leq c \log n$); note that such circuits necessarily have polynomial size since only binary gates are used. It is an easy exercise to show that, with the proper uniformity condition, $NC^1 \subseteq LOGSPACE$; it is an open question to determine if the inclusion is proper or not.

An example of a non-trivial function computable in NC^1 is iterated addition, which is defined as follows:

input: n n-bit integers

output: the $n + \log n$-bit sum of the inputs

The trick is to repeatedly replace each group of three integers of length m (say $X = x_1 \ldots x_m, Y = y_1 \ldots y_m$ and $Z = z_1 \ldots z_m$) by two integers of length $m+1$ (say $C = 0c_1 \ldots c_m$ and $D = d_0 \ldots d_{m-1}0$) such that $X + Y + Z = C + D$. This can be done by letting

$$c_i = 1 \text{ iff } x_i + y_i + z_i \text{ is odd (for } i = 1, \ldots, m)$$
$$d_i = 1 \text{ iff } x_{i+1} + y_{i+1} + z_{i+1} \geq 2 \text{ (for } i = 1, \ldots, m)$$

In $O(\log n)$ stages, each of constant depth, we thus produce two integers, whose sum is equal to that of the original n input numbers. We leave as an exercise to show how these two remaining integers can be added up with $O(\log n)$ levels of binary gates. This example implies that the language

$$MAJORITY(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum x_i \geq n/2 \\ 0 & \text{otherwise} \end{cases}$$

and the language

$$MOD_q(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum x_i \equiv 0 \bmod q \\ 0 & \text{otherwise} \end{cases}$$

are in NC^1. (Here we identify a language $L \subseteq \{0,1\}^*$ with its characteristic function, mapping $(x_1, \ldots, x_n)$ to 1 iff $x_1 \ldots x_n \in L$.)

We next define certain subclasses of NC^1 by restricting the depth of our circuit families $C = (C_n)_{n\geq 0}$ to be constant while allowing more powerful gates than simply binary AND and OR. The class of constant-depth circuit families constructed with binary gates (usually denoted NC^0) is not very interesting: if $C = (C_n)_{n\geq 0}$ is such a circuit family, there is a fixed bound t on the number of bits that can influence the value of $C_n(x)$, and thus C cannot compute the AND function since it depends on all bits.

A more interesting case is obtained by considering circuit families of constant-depth, polynomial size using AND gates and OR gates of arbitrary fan-in. This class of languages is denoted by AC^0. The bound on the size implies that all gates have polynomial fan-in and thus each one can be replaced by an $O(\log n)$-depth tree of binary gates. Hence $AC^0 \subseteq NC^1$. A deep result, independently proved by Furst, Saxe and Sipser [16] and Ajtai [1], says that the inclusion is proper; any MOD_q language belongs to NC^1 but not to AC^0.

In a similar vein, we define, for any $q \geq 2$, $CC^0(q)$ as the class of circuit families $C = (C_n)_{n \geq 0}$ having constant depth, polynomial size, constructed with binary AND and OR gates, and MOD_q gates of arbitrary fan-in. Since MOD_q is computable in NC^1, it follows that $CC^0(q) \subseteq NC^1$. We will write CC^0 for $\bigcup_q CC^0(q)$. Finally $ACC^0(q)$ is similarly obtained by allowing AND, OR and MOD_q gates of arbitrary fan-in: ACC^0 will stand for $\bigcup_q ACC^0(q)$ and we have $ACC^0 \subseteq NC^1$. It can be shown that if p and q have the same prime divisors then $CC^0(p) = CC^0(q)$ and $ACC^0(p) = ACC^0(q)$. If p and q are distinct primes, Smolensky has shown in [27] that $MOD_p \notin ACC^0(q)$, hence that $ACC^0(q) \nsubseteq NC^1$ in that case. The status of the inclusion when q is composite remains an open question.

4.3. Programs over finite monoids

The connection between circuits and monoids is based on the following definition. Let M be a finite monoid; an n-input M-program ϕ_n (over input alphabet A) is a sequence $\phi_n = \nu_1 \dots \nu_l$ of instructions, where ν_j has the form (i_j, f_j) for some $1 \leq i_j \leq n$, $f_j : A \to M$. On input $x = x_1 \dots x_n \in A^n$, $\phi_n(x)$ returns the monoid element which is the product of $f_1(x_{i_1}) \dots f_l(x_{i_l})$. As in the circuit case, functions from A^* into M are obtained by considering families of M-programs $\phi = (\phi_n)_{n \geq 0}$. Fixing an accepting subset $F \subseteq M$, ϕ then recognizes the language L, where $L \bigcap A^n = \{x \in A^n : \phi_n(x) \in F\}$. We will denote by $\mathcal{B}(M)$ the class of languages that can be thus recognized by families of M-programs $\phi = (\phi_n)_{n \geq 0}$, with the added constraint that ϕ_n has polynomial length, i.e. ϕ_n contains at most n^c instructions for some constant c. We will consider only the non-uniform version of this definition, but as in the circuit case, uniformity criteria on the algorithmic definability of our programs can be imposed.

Such computing devices were originally introduced, in a different but equivalent form, under the name bounded-width branching program (Borodin et al. [8]). They observed the following

Lemma 4.1 *For any M, $\mathcal{B}(M) \subseteq NC^1$.*

Proof. Suppose we have a family $\phi = (\phi_n)_{n \geq 0}$ of M-programs, where ϕ_n has length $l = n^c$ and $L \bigcap A^n = \{x : \phi_n(x) \in F\}$. We define an n-input circuit C_n that recognizes the same set in depth $O(\log n)$ by the following process.

Given $x = x_1 \dots x_n \in A^n$:

- in constant-depth (in fact using wires only) produce for each instruction $\nu_j = (i_j, f_j)$ the binary encoding of $f_j(x_{i_j})$;
- in parallel multiply the l monoid elements two by two, i.e. produce $l/2$ binary encodings corresponding to the products of pairs of elements; this can be done in constant depth;
- repeat the previous step until the product of the l elements, i.e. the value of $\phi_n(x)$, is obtained; this will require $\log l = O(\log n)$ stages, each of constant depth;

– test, in constant depth, if $\phi_n(x)$ belongs to F or not.

□

Borodin et al. [8] conjectured that $NC^1 \subseteq \bigcup_M \mathcal{B}(M)$ was false. Indeed, it seems that the fixed-size memory of the program should not allow computing $MAJORITY$, i.e. counting if the number of bits that are on is at least $n/2$. It came as quite a surprise when Barrington disproved the conjecture in [2], using a clever but simple trick available in simple non-abelian groups. The deep relationship between small-depth circuits and computations realized by M-programs was further strengthened by Barrington and Thérien [6] and Barrington, Straubing and Thérien [5] where it was shown that natural subclasses of NC^1 correspond exactly to natural algebraic restrictions on the monoids that can be used.

Before presenting some of these results, we introduce a notion of reducibility between languages which arises naturally in this context and which will make some of the arguments to come easier to describe. Let A, B be finite alphabets and consider n-input programs for which the instructions have the form (i, f) for some $1 \leq i \leq n, f : A \to B$. The program thus induces a map $\phi_n : A^n \to B^*$. Let $L \subseteq A^*, K \subseteq B^*$; we say that L is reducible to K, denoted $L \leq K$, iff there is a family $\phi = (\phi_n)_{n \geq 0}$ of programs as above such that the length of ϕ_n is at most n^c for some constant c and $x \in L \bigcap A^n$ iff $\phi_n(x) \in K$. It is easily seen that $\leq$ is transitive and that any class $\mathcal{B}(M)$ is closed under this reducibility.

Recall that a language $K \subseteq B^*$ is regular iff there exists a morphism $\phi : B^* \to M$, where M is some finite monoid, such that $K = \phi^{-1}\phi(K)$. The interest of our reducibility notion is that the classes $\mathcal{B}(M)$ are exactly the closure under $\leq$ of the regular languages recognized by M via morphisms.

Lemma 4.2 *$L \in \mathcal{B}(M)$ iff $L \leq K$ for some regular language K recognized by M via a morphism.*

Proof. If $L \in \mathcal{B}(M)$ we can view the family of M-programs $\phi = (\phi_n)_{n \geq 0}$ as producing a string in M^* instead of an element of M. Letting $\eta_M : M^* \to M$ be the canonical morphism and $K = \eta_M^{-1}(F)$, we have that $L \leq K$. For the converse, suppose ϕ is the reduction from L to K and $\psi : B^* \to M$ is the morphism recognizing K. Composing ϕ and ψ gives us a family of M-programs that recognizes L, using $\psi(K)$ as accepting set. □

4.4. Barrington's theorem

We now prove the theorem of Barrington ([2]). Consider S_5, the group of permutations on five points: let e be the identity and σ be any 5-cycle. We define $\mathcal{B}_\sigma(S_5)$ to be the class of languages L for which there is a polynomial length family of S_5-programs ϕ such that

$$\phi(x) = \begin{cases} \sigma & \text{if } x \in L \\ e & \text{otherwise} \end{cases}$$

Lemma 4.3 *For any two 5-cycles σ and τ $\mathcal{B}_\sigma(S_5) = \mathcal{B}_\tau(S_5)$.*

Proof. We have that σ and τ are conjugates in S_5, say $\tau = \theta^{-1}\sigma\theta$. Suppose $\phi_n = \nu_1 \ldots \nu_l$ is such that $\phi_n(x) = \sigma$ if $x \in L, e$ otherwise. We modify ϕ_n as follows:

the first instruction is changed to produce $\theta^{-1} f_1(x_{i_1})$ instead of $f(x_{i_1})$ and the last one is changed to produce $f(x_{i_l})\theta$ instead of $f(x_{i_l})$. Then the new program produces $\theta^{-1}\phi_n(x)\theta$ instead of $\phi_n(x)$, i.e. it produces τ iff the original one was yielding σ and e otherwise. Note that the modification preserves the length of the program. □

Lemma 4.4 *$\mathcal{B}_\sigma(S_5)$ is closed under complement*

Proof. As in the previous proof, we modify each program by changing the last instruction so that it produces $f_l(x_{i_l})\sigma^{-1}$ instead of $f(x_{i_l})$. The new program produces $\phi_n(x)\sigma^{-1}$ instead of $\phi_n(x)$, i.e. it produces σ^{-1} when the original one was yielding e and e when the original one was yielding σ. Hence $\overline{L}$ is in $\mathcal{B}_{\sigma^{-1}}(S_5) = \mathcal{B}_\sigma(S_5)$. Once again, the modification preserves the length. □

Lemma 4.5 *Let $\sigma = (12345), \tau = (13542)$. Then $\theta = \sigma\tau\sigma^{-1}\tau^{-1}$ is a 5-cycle.*

Proof. $(12345)(13542)(54321)(24531) = (13254)$. □

Theorem 4.6 $NC^1 = \bigcup_M \mathcal{B}(M)$.

Proof. The theorem is proved by showing that an n-input circuit, which we can assume to be a tree, of depth d can be simulated by an n-input S_5-program of length 4^d which will yield some 5-cycle if the input is accepted by the circuit and the identity otherwise. By induction on d;

$d = 0$ The circuit has the form $X_i, \overline{X_i}, 1$ or 0. In each case, a single instruction can appropriately simulate the circuit.

$d > 0$ By Lemma 5.2, we can assume that the output gate is an AND, i.e. that $C = AND(C_1, C_2)$ where each of C_1, C_2 have depth $d-1$. By the induction hypothesis, there exist ϕ_1 via which the language accepted by C_1 is in $\mathcal{B}_\sigma(S_5)$, ϕ_2 via which the language accepted by C_2 is in $\mathcal{B}_\tau(S_5)$, ϕ_3 via which the language accepted by C_1 is in $\mathcal{B}_{\sigma^{-1}}(S_5)$, ϕ_4 via which the language accepted by C_4 is in $\mathcal{B}_{\tau^{-1}}(S_5)$, each of which has length 4^{d-1}. Let $\phi = \phi_1\phi_2\phi_3\phi_4$; then for any $x \in \{0,1\}^n$,

$$\phi(x) = \begin{cases} \sigma\tau\sigma^{-1}\tau^{-1} = \theta & \text{if } C(x) = 1 \\ e & \text{otherwise} \end{cases}$$

□

We add the following remarks.

- The theorem in effect proves that $NC^1 = \mathcal{B}_\sigma(S_5)$. In fact it is true that $NC^1 = \mathcal{B}(M)$ for any monoid M containing a simple non-abelian group.
- In terms of the reducibility notion introduced in the last section, the theorem says that for any language L in NC^1, we have that $L \leq K$ where K is the word problem of S_5.
- An identical trick was used in a different context by Maurer and Rhodes in [20]. Indeed, the theorem of this section follows directly from their result.

4.5. Algebraic descriptions of some subclasses of NC^1

In the last section, it was shown that $\mathcal{B}(M) = NC^1$ for any monoid containing a simple non-abelian group. A natural question is to investigate the computing power of monoids that do not have this property (they are called solvable monoids). One nice feature of the relationship via programs between circuits and monoids is the fact that natural subclasses of NC^1 (such as AC^0, CC^0, ACC^0) can be put in correspondence with natural restrictions on the monoids (respectively aperiodic monoids, solvable groups, solvable monoids).

These results are easier to prove using reductions to regular languages: we then use the nice combinatorial descriptions of the languages recognized via morphisms into aperiodic monoids (Schützenberger [26]), solvable groups (Straubing [30], Thérien [33]) and solvable monoids ([33]).

We will now give in details the proof for aperiodic monoids. Consider the following hierarchy of languages over some alphabet A:

- $\mathcal{H}_1$ = boolean closure of languages of the form A^*aA^* with $a \in A$
- $\mathcal{H}_k$ = boolean closure of $\mathcal{H}_{k-1}$ and languages of the form $L_0a_1L_1 \dots a_rL_r$ with $L_i \in \mathcal{H}_{k-1}, a_i \in A$.

This is the dot-depth hierarchy as introduced in Section 2.2, modified slightly at the first level $\mathcal{H}_1$. A language L is star-free, that is can be described by a regular expression not using the $*$ operator (but allowing complement) iff it belongs to some $\mathcal{H}_k$. ¿From Sections 2 and 3 we know that L is star-free iff it can be recognized via a morphism into an aperiodic monoid. Note as a consequence of this result that the language MOD_q cannot be described by a star-free expression since any monoid recognizing MOD_q via a morphism must contain the cyclic group of order q.

We are now ready to prove our algebraic characterization of the circuit class AC^0.

Theorem 4.7 (Barrington, Thérien [6]) $AC^0 = \bigcup_M \mathcal{B}(M)$, *where the union ranges over all aperiodic monoids.*

Proof. To show the inclusion from right to left we use the fact that $L \in \mathcal{B}(M)$ for some M aperiodic iff $L \leq K$ for some star-free K. Let $\phi = (\phi_n)_{n\geq 0}$ be the reduction from L to K. We construct an appropriate circuit as follows. Given $x = x_1 \dots x_n \in A^n$:

- in constant depth, in fact using wires only, produce the binary encoding of $w = \phi_n(x)$;
- determine if $w \in K$; a union operation translates into a binary OR, a complement translates into a negation (which can be pushed to the input level using de Morgan's laws); a concatenation $K_0b_1K_1 \dots b_rK_r$ translates into an OR of fan-in $\binom{n}{r}$ of ANDs of fan-in $2r+1$ (the subcircuit has to check every set of r positions of the input to verify if these positions contain $b_1, \dots, b_r$ and if the intermediate segments belong to $K_0, \dots, K_r$).

Note that when K belongs to $\mathcal{H}_k$, the circuit will have k levels of unbounded gates.

For the converse, we assume that our constant depth circuit $C = (C_n)_{n\geq 0}$ is over the binary alphabet and that it has k alternating levels of OR gates and AND gates. We say that a circuit is an OR-circuit (AND-circuit) if the output gate is an OR (AND). We show by induction on k that $L \leq K_k$ for some star-free $K_k \subseteq B_k^* = \{a, b, c_1, \dots, c_{k-1}\}^*$.

$k=1$: Suppose $C_n = OR(X_{i_1}, \ldots, X_{i_s}, \overline{X_{j_1}}, \ldots, \overline{X_{j_t}})$. We let $\phi_n = (i_1, f) \ldots (i_s, f), (j_1, \overline{f}), \ldots, (j_t, \overline{f})$, where $f(0) = \overline{f}(1) = a, f(1) = \overline{f}(0) = b$. Clearly $\phi_n(x) \in B_1^* b B_1^*$ iff $C_n(x) = 1$. If C_n is an AND gate instead, we have $\phi_n(x) \in \overline{B_1^* a B_1^*}$ iff $C_n(x) = 1$. Letting $S_1 = B_1^* b B_1^*$ and $P_1 = \overline{B_1^* a B_1^*}$, we notice that both S_1, P_1 are recognized via a morphism into U_1, the 2-element semilattice.

$k>1$: We suppose inductively that there exist S_{k-1}, P_{k-1}, each recognized via a morphism into an aperiodic monoid U_{k-1}, such that for any n-input OR-circuit C (AND-circuit D) of depth $k-1$ and size n^c, there exists an n-input program ϕ having the property that $\phi(x) \in S_{k-1}(P_{k-1})$ iff $C(x) = 1 (D(x) = 1)$ and whose length is a polynomial with the degree depending only on k and c. Let now $C = OR(D_1, \ldots, D_s)$ be an OR-circuit of depth k and size n^c. We let $\phi = (1, f)\phi_1(1, f) \ldots \phi_s(1, f)$, where $f(0) = f(1) = c_{k-1}$, and $\phi_1, \ldots, \phi_s$ are the programs recognizing the languages of the AND-circuits $D_1, \ldots, D_s$ respectively. It is easily seen that $\phi(x) \in B_k^* c_{k-1} P_{k-1} c_{k-1} B_k^*$ iff $C(x) = 1$ since the only way the program can produce a segment in P_{k-1} between two markers c_{k-1} is to have some D_i being 1. Moreover the condition on the length of ϕ is obviously satisfied. If C is an AND-circuit instead, we see that $\phi(x) \in (c_{k-1} S_{k-1})^* c_{k-1}$ iff $C(x) = 1$. Let S_k, P_k be the two languages just introduced. Consider U_{k-1} as a transformation monoid and let $\overline{U_{k-1}}$ be obtained from U_{k-1} by adding the constant transformations. Then one can show that S_k, P_k are both recognized by the wreath product $U_k = U_1 \circ \overline{U_{k-1}}$; the idea is that the markers c_{k-1} act as resets to the identity of U_{k-1} in the front monoid and the end copy of U_1 is used to detect if what happens between two consecutive markers belongs to the appropriate set. Note that U_k is aperiodic, hence S_k, P_k are star-free.

□

We have used a different alphabet B_k for circuits of depth k to simplify the proof. A more careful argument can be given that works with only one marker for arbitrary depth. Finally, define AC_k^0 as those constant depth polynomial size families of circuits $C = (C_n)_{n \geq 0}$ satisfying the condition that, for any n, any path in C_n contains at most k gates of fan-in greater than 2 (for an unimportant technical reason, we will suppose that all gates on the first level have fan-in greater than 2; this can always be achieved by duplicating inputs); it is possible to specialize the above theorem to each individual class AC_k^0. This will be stated in full in the next subsection.

Similar characterizations exist for the classes CC^0 and ACC^0 in terms of solvable groups and solvable monoids respectively. One way to prove these is to use the combinatorial descriptions of languages recognize via morphisms into such monoids ([30], [33]), descriptions which are based in part on a modular counting version of concatenation.

Theorem 4.8 [5] $CC^0 = \bigcup_M \mathcal{B}(M)$, *where the union ranges over all solvable groups;* [6] $ACC^0 = \bigcup_M \mathcal{B}(M)$, *where the union ranges over all solvable monoids.*

Once again this general theorem specializes to take into consideration the exact depth of the circuits.

4.6. Conclusion

We call variety a class $\mathbf{V}$ of finite monoids closed under division (i.e. morphic image of submonoid) and finite direct product. It has clearly emerged over the years that varieties are the proper level at which to classify languages recognized via morphisms into finite monoids (see [13], [24]).

The results of the last subsection show that many interesting subclasses of NC^1 can be algebraically characterized using polynomial length programs. It turns out that varieties seem to play a key role in this setting as well and McKenzie, Péladeau and Thérien [21] explore this point of view in details. Define $\mathcal{B}(\mathbf{V}) = \bigcup_{M \in \mathbf{V}} \mathcal{B}(M)$; let $\mathbf{M}$ stand for the variety of all monoids, $\mathbf{A}$ for the variety of aperiodic monoids, $\mathbf{Gsol}$ for the variety of solvable groups and $\mathbf{Msol}$ for the variety of solvable monoids. We thus have $NC^1 = \mathcal{B}(\mathbf{M})$, $AC^0 = \mathcal{B}(\mathbf{A})$, $CC^0 = \mathcal{B}(\mathbf{Gsol})$ and $ACC^0 = \mathcal{B}(\mathbf{Msol})$. The separation of circuit classes is thus, in all those cases, equivalent to separating the computing power of the corresponding varieties. The problem is that contrary to what happens for morphisms, distinct varieties $\mathbf{V}$ and $\mathbf{W}$ can give rise to the same class $\mathcal{B}(\mathbf{V}) = \mathcal{B}(\mathbf{W})$. We give two examples:

Example 1 It follows from Barrington's theorem that for any variety $\mathbf{V}$ containing a simple non-abelian group we have $\mathcal{B}(\mathbf{V}) = \mathcal{B}(\mathbf{M})$.

Example 2 Let $\mathbf{J_1}$ be the variety of idempotent and commutative monoids, and for $k \geq 2$, let $\mathbf{J_k}$ be the variety generated by monoids of the form $U_1 \circ \overline{M_{k-1}}$ where $M_{k-1} \in \mathbf{J_{k-1}}$. For $k \geq 1$, let $\mathbf{H_k}$ be the smallest variety containing enough monoids to recognize all languages in $\mathcal{H}_k$; we have $\mathbf{H_1} = \mathbf{J_1}$ but $\mathbf{J_k} \not\subseteq \mathbf{H_k}$ for all $k > 1$. On the other hand a careful inspection of the proof of Theorem 4.7 shows that $AC^0_k = \mathcal{B}(\mathbf{J_k}) = \mathcal{B}(\mathbf{H_k})$.

In [21], it is shown that $\mathcal{B}(\mathbf{V}) = \mathcal{B}(\mathbf{W})$ iff the two classes contain the same regular languages: more precisely $\mathcal{B}(\mathbf{V}) \subseteq \mathcal{B}(\mathbf{W})$ iff any word problem of any monoid in $\mathbf{V}$ can be computed by a polynomial length family of programs over some monoid in $\mathbf{W}$. Thus the same claim applies to corresponding circuit classes. This paper also offers a general conjecture about the computing power of various varieties contained in $\mathbf{Msol}$ which, if proved, would give as corollaries virtually all the known results and conjectures believed to be true about the internal structure of NC^1.

5. Tying in Logic Again

5.1. Circuit Complexity and Logic

Let us return to the question asked at the end of Section 3: Is there *any* numerical predicate we can introduce so that the set of strings over $\{a, b\}$ with an even number of occurrences of a is first-order definable?

The answer is 'No'. This follows from the circuit lower bounds of the last lecture, and the next theorem. Let us denote by N the class of all numerical predicates, so that $FO[N]$ denotes the family of languages definable by first-order sentences in which arbitrary numerical predicates are admitted.

Theorem 5.1 $FO[N] = AC^0$.

This theorem appears in Immerman [19] and Gurevich and Lewis [17]. By the results of [1] and [16], cited in Section 4, the language MOD_2 is not in AC^0, so the theorem immediately gives the answer to our question.

Theorem 5.1 is one of a collection of results that characterize computational complexity classes in terms of formal logic. The first of these is a theorem of Fagin [15] that the class NP of languages recognizable in nondeterministic polynomial time are precisely those definable by existential second-order sentences with successor. Immerman [19] gives a large number of such results for various complexity classes, including Theorem 5.1.

The theorem is quite surprising at first glance, because the polynomial size bound in the definition of AC^0 appears nowhere in the definition of $FO[N]$. In fact, the theorem is not all that difficult to prove. We will sketch here a quick proof of the theorem that uses the characterization of AC^0 in terms of programs over finite monoids (see Theorem 4.6). First, suppose L is defined by a first-order sentence ϕ. Beginning with the outermost quantifier, we replace

$$\exists x \theta(x)$$

by

$$\bigvee_{i=1}^{n} \theta(i),$$

and similarly replace universal quantifiers by AND-gates. When we reach the atomic formulas we replace sentences of the form

$$\rho(i_1, \ldots, i_m),$$

where ρ is a numerical predicate by a constant 1 or 0, depending on whether the sentence is true or false, and $Q_1 i, Q_0 i$ by X_i and $\bar{X}_i$, respectively. The resulting expression is a circuit that recognizes the set of strings of length n in L. The depth of this circuit is equal to the depth k of nesting of the quantifiers in ϕ, and the size is bounded by a polynomial in n of degree k. Thus $L \in AC^0$.

Conversely, if $L \in AC^0$, then L is recognized by a family of programs of length n^k over a finite aperiodic monoid M. That is, an input string $w = a_1 \cdots a_n$ over $\{0, 1\}$ is translated to a string

$$\pi(w) = b_1 \cdots b_{n^k} \in M^*.$$

Since M is aperiodic, it follows from the results cited in Section 3 that w is accepted if and only if $\pi(w)$ satisfies a sentence θ of $FO[<]$. We will use θ to construct a sentence of $FO[N]$ that defines L. We can encode each position in $\pi(w)$ by a k-tuple of positions in w. We then replace each quantifier

$$\exists x \psi(x), \forall x \psi(x),$$

in θ by

$$\exists x_1 \cdots \exists x_k \psi(x_1, \ldots, x_k), \forall x_1 \cdots \forall x_k \psi(x_1, \ldots, x_k).$$

The numerical relation $<$ on positions in $\pi(w)$ is replaced by a $2k$-ary numerical predicate expressing the corresponding relation on positions of w (this will depend on the encoding). The atomic formula $Q_m x$, where $m \in M$ is replaced by

$$(R^0_m(x_1, \ldots, x_k, y, z) \wedge Q_0 y) \vee (R^1_m(x_1, \ldots, x_k, y, z) \wedge Q_1 y),$$

where $R^j_m(x_1, \ldots, x_k, y, z)$ is interpreted to mean: 'in the program for inputs of length z, the x^{th} instruction, where x is encoded by $(x_1, \ldots, x_k)$, reads input bit y and emits m if this bit is equal to j'. This is a numerical predicate that depends on the choice of encoding. When this translation is complete we have the promised sentence of $FO[N]$ that defines L.

The same techniques can be used to give characterizations of other circuit complexity classes in terms of modular quantifiers:

Theorem 5.2 *Let $q > 0$. Then*

$$ACC^0(q) = (FO + MOD_k[N]),$$

and

$$CC^0(q) = MOD_k[N].$$

5.2. Regular Languages in Circuit Complexity Classes

The following theorem, due to Barrington, Compton, Straubing and Thérien [3], characterizes the regular languages in AC^0.

Theorem 5.3 *Let $L \subseteq \{0,1\}^*$ be a regular language. $L \in AC^0$ if and only if*

$$L \in FO[<, \{x \equiv 0 \pmod{k} : k > 1\}].$$

The 'if' direction of this theorem follows immediately from Theorem 5.1. For the only if direction, we use the algebraic characterization of the class $FO[<, \{x \equiv 0 \pmod{k} : k > 1\}]$ from Section 3: If $L \in AC^0$ is regular and $L \notin FO[<, \{x \equiv 0 \pmod{k} : k > 1\}]$, then there exist $t > 0$, $q > 1$ such that $\mu_L(A^t)$ contains a cyclic group of order q. We use this together with the circuits for L to construct an AC^0 family of circuits that counts the number of occurrences of 1 in the input modulo q. This contradicts the results of [1] and [16] cited in Section 4.

5.3. A General Principle?

We will say that a numerial predicate is *regular* if it can be expressed as a first-order formula in the atomic formulas $x < y$ and $x \equiv 0 \pmod{k}$. The terminology is justified by the fact that any second-order monadic sentence that uses these numerical predicates is regular, while any nonregular numerical predicate can be used to define a nonregular language. For example,

$$x + y \equiv z \pmod{7}$$

is a regular numerical predicate, while

$$x + y = z$$

is not. Let us denote by Reg the class of all regular numerical predicates, and by $\mathcal{L}_{Reg}$ the class of all regular languages. The last theorem can then be restated as

$$FO[N] \cap \mathcal{L}_{Reg} = FO[Reg].$$

Let us see what this means. It may happen, of course, that a sentence using nonregular numerical predicates defines a regular language. This occurs, for instance, with the sentence

$$\forall x \exists y (x | y \wedge Q_0 y).$$

Here '|' has the usual meaning 'divides'. The language defined is the regular language $0(0+1)^*$. Now in this case we could just as well have written

$$\forall x \exists y (y \leq x \wedge Q_0 y),$$

which uses only regular numerical predicates. The theorem says that this phenomenon is general: Whenever we define a regular language by a first-order sentence using nonregular numerical predicates we can define the same language by a sentence that contains only regular numerical predicates.

Thus Theorem 5.3, which at first glance concerns lower bounds for boolean circuits, is equivalent to a very simple and natural-looking principle about the definability of regular languages in first-order logic. Is there a direct proof of this principle? If we could prove

$$FO[N] \cap \mathcal{L}_{Reg} = FO[Reg]$$

using only algebraic and automaton-theoretic considerations, then we would obtain as a consequence a new proof that the language MOD_2 is not in AC^0.

But there is more: There is considerable evidence that the analogous principle holds for modular quantifiers, and these are provably equivalent to *open* problems about circuit complexity. For example, we have:

Theorem 5.4 *The following are equivalent:*
(a) $ACC^0 \neq NC^1$.
(b) $(FO + MOD)[N] \cap \mathcal{L}_{Reg} = (FO + MOD)[Reg]$.

Let us see why this is so. If $ACC^0 = NC^1$, then ACC^0 contains all regular languages; in particular, ACC^0 contains regular languages with nonsolvable syntactic monoids. Thus

$$ACC^0 \cap \mathcal{L}_{Reg} = (FO + MOD)[N] \cap \mathcal{L}_{Reg}$$

is strictly larger than $(FO + MOD)[Reg]$, which contains only languages with solvable syntactic monoids. Conversely, if $(FO + MOD)[Reg]$ is strictly contained in $(FO+MOD)[N] \cap \mathcal{L}_{Reg}$, then ACC^0 contains a regular language with a nonsolvable syntactic monoid. It follows from the results in Section 4 on complete problems for NC^1 that ACC^0 contains every language in NC^1.

There are analogous equivalences for CC^0 and for the classes $ACC^0(q)$ and $CC(q)$ for a fixed modulus $q > 1$:

Theorem 5.5 *The following are equivalent:*
(a) CC^0 *does not contain the* AND *function.*
(b) $MOD[N] \cap \mathcal{L}_{Reg} = MOD[Reg]$.

Theorem 5.6 *Let* $q > 1$. *The following are equivalent:*
(a) $ACC^0(q)$ *does not contain the language*

$$\{a_1 \cdots a_n \in \{0,1\}^* : \sum_{i=1}^{n} a_i \equiv 0 \pmod{p}\},$$

where p *is any prime that does not divide* q.
(b) $(FO + MOD_q)[N] \cap \mathcal{L}_{Reg} = (FO + MOD_q)[Reg]$.

Theorem 5.7 *The following are equivalent:*
(a) $CC^0(q)$ *contains neither the* AND *function nor the language*

$$\{a_1 \cdots a_n \in \{0,1\}^* : \sum_{i=1}^{n} a_i \equiv 0 \pmod{p}\},$$

where p *is any prime that does not divide* q.
(b) $MOD_q[N] \cap \mathcal{L}_{Reg} = MOD_q[Reg]$.

Moreover, in the last two theorems, the pairs of equivalent statements are known to be true when q is prime, because we possess direct proofs of the statements concerning circuits. We regard this as evidence that the equivalent pairs of statements are true in general. We thus conjecture that for any set $\mathcal{Q}$ of ordinary and modular quantifiers, $\mathcal{Q}[N] \cap \mathcal{L}_{Reg} = \mathcal{Q}[Reg]$. Again, this is a general principle concerning the logical definability of regular languages, equivalent to a quite different-looking principle concerning constant-depth circuits. The cases where it is known to be true are proved using the equivalent circuit formulations. We would like to see a direct proof of the general statement, one that brings to the fore the automaton-theoretic and algebraic considerations. When N is replaced by the union of the regular numerical predicates and the class of all monadic numerical predicates, the conjecture is known to be true ([28]).

References

1. M. Ajtai, Σ_1^1 formulae on finite structures, *Annals of Pure and Applied Logic* **24** (1983), pp. 1–48.
2. D.A.M. Barrington, Bounded width branching programs recognize exactly those languages in NC^1, *J. Computer and Systems Science* **38** (1989), pp. 150–164.
3. D. A. M. Barrington, K. Compton, H. Straubing and D. Thérien, Regular languages in NC^1, *J. Comp. Syst. Sci.* **44** (1992), pp. 478-499.
4. D.A.M. Barrington, N. Immerman and H. Straubing, On uniformity within NC^1, *J. Computer and Systems Science* **41** (1990), pp. 274–306.
5. D.A.M. Barrington, H. Straubing and D. Thérien, Non-uniform automata over groups, *Information and Computation* **89** (2) (1990), pp. 109–132.
6. D.A.M. Barrington and D. Thérien, Finite monoids and the fine structure of NC^1, *J. of the Association for Computing Machinery* **35** (1988), pp. 941–952.

7. D. Beauquier and J. E. Pin, Factors of words, *Proc. 16th ICALP,* Springer Lecture Notes in Computer Science **372** (1989), pp. 63–79.
8. A. Borodin, D. Dolev, F.E. Fich and W. Paul, Bounds for width two branching programs, *Proc. of the 15 ACM Symp. on the Theory of Computing* (1983), pp. 87–93.
9. J.A. Brzozowski, R. Knast, The dot-depth hierarchy of star-free languages is infinite, *J. Comput. System Sci.* **16** (1978), pp. 37-55.
10. J.R. Büchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundl. math.* **6** (1960), pp. 66-91.
11. R.S. Cohen, J.A. Brzozowski, Dot-depth of star-free events, *J. Comput. System Sci.* **5** (1971), pp. 1-15.
12. S.A. Cook, A taxonomy of problems with fast parallel solutions, *Information and Computation* **64** (1985), pp. 2–22 .
13. S. Eilenberg, *Automata, Languages and Machines,* Vol. B, Academic Press, New York 1976.
14. C.C. Elgot, Decision problems of finite automata design and related arithmetics, *Trans. Amer. math. Soc.* **98** (1061), pp. 21-52.
15. R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in R. Karp, ed., *The Complexity of Computation,* SIAM-AMS Proceedings, vol. 7, American Mathematical Society, Providence, Rhode Island, 1974.
16. M.L. Furst, J.B. Saxe and M. Sipser, Parity, circuits, and the polynomial-time hierarchy, *Math. Syst. Theory* **17** (1984), pp. 13–27.
17. Y. Gurevich and H. Lewis, A logic for constant-depth circuits, *Information and Control,* **61** (1984), pp. 65–74.
18. J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley (1979).
19. N. Immerman, Languages that capture complexity classes, *SIAM J. Computing* **16** (1987), pp. 760–778.
20. W. Maurer and J. Rhodes, A property of finite simple non-abelian groups, *Proc. Amer. Math. Soc.* **16** (1965), pp. 552–554.
21. P. McKenzie, P. Péladeau and D. Thérien, NC^1: the automata-theoretic viewpoint, *Computational Complexity* **1** (1991), pp. 330–359.
22. R. McNaughton and S. Papert, *Counter-Free Automata*, MIT Press, Cambridge, Mass. 1971.
23. D. Perrin, J.E. Pin, First-order logic and star-free sets, *J. Comput. System Sci.* **32** (1986), pp. 393-406.
24. J.-E. Pin, *Varieties of Formal Languages,* Plenum Press (1986).
25. J. Rhodes and B. Tilson, The kernel of monoid morphisms, *J. Pure and Applied Algebra* **62** (1989), pp. 227–268.
26. M. P. Schützenberger, On finite monoids having only trivial subgroups, *Information and Control* **8** (1965), pp. 190–194.
27. R. Smolensky, Methods in the theory of lower bounds for boolean circuit complexity, *Proc. of the 19 ACM Symp. on the Theory of Computing* (1987), pp. 77–82.
28. H. Straubing Circuit complexity and the expressive power of generalized first-order formulas, in *Proc. 19th ICALP,* Springer Lecture Notes in Computer Science **623** (1992), pp. 16–27.
29. H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity,* Birkhäuser, Boston, 1994.
30. H. Straubing, *Varieties of recognizable sets whose syntactic monoids contain solvable groups*, Ph. D. Thesis, UC Berkeley, 1978.
31. H. Straubing, A generalization of the Schützenberger product of finite monoids, *Theoretical Computer Science* **13**, pp. 107-110.
32. H. Straubing, D. Thérien and W. Thomas, Regular languages defined with generalized quantifiers, in *Proc. 15th ICALP,* Springer Lecture Notes in Computer Science **317** (1988), pp. 561–575.
33. D. Thérien, Classification of finite monoids: the language approach, *Theoretical Computer Science* **14** (1981), pp. 195–208.
34. D. Thérien and A. Weiss, Graph congruences and the wreath product, *J. Pure and Applied Algebra* **36** (1985), pp. 205–215.
35. W. Thomas, The theory of successor with an extra predicate, *Math. Annalen* **237** (1978), pp. 121–132.
36. W. Thomas, Classifying regular events in symbolic logic, *J. Comput. System Sci.* **25** (1982),

pp. 360-376.

37. W. Thomas, An application of the Ehrenfeucht-Fraïssé game in formal language theory, *Bull. Soc. Math. de France* **16** (1984), pp. 11-21.
38. W. Thomas, Automata on infinite objects, in: *Handbook of Theoretical Computer Science* (J. v. Leeuwen, ed.), Vol. B, Elsevier Sci. Publ., Amsterdam 1990, pp. 133-191.
39. W. Thomas, The Ehrenfeucht-Fraïssé game in theoretical computer science, in: TAPSOFT 93 (M.C. Gaudel, J.P. Jouannaud, eds.), Springer Lecture Notes in Computer Science **669**, pp. 559-568.