REGULAR LANGUAGES DEFINED WITH GENERALIZED QUANTIFIERS

Howard Straubing*
Computer Science Department
Boston College
Chestnut Hill, Massachusetts
USA 02167

Denis Thérien
School of Computer Science
McGill University
Montréal, Québec
Canada H3A 2K6

Wolfgang Thomas
Institut für Informatik und Praktische Mathematik
Universität Kiel
D-24098 Kiel
Germany

* Research supported by National Science Foundation Grant CCR-9203208

Abstract

We study an extension of first-order logic obtained by adjoining quantifiers that count with respect to an integer modulus. It is shown that the languages definable in this framework are precisely the regular languages whose syntactic monoids contain only solvable groups. We obtain an analogous result for regular ω-languages and establish some connections with complexity theory for fixed-depth families of circuits. An earlier version of this paper appeared in the Proceedings of the 1988 International Colloquium on Automata, Languages and Programming.

Running head: Generalized Quantifiers

Contact author:

Howard Straubing
Department of Computer Science
Boston College
Chestnut Hill, Massachusetts, 02167

LIST OF SYMBOLS

$\neg$

$\exists$

$\forall$

$\wedge$

$\vee$

$<, \leq, >, \geq$

$\times, \Rightarrow, \setminus, \in$

$\subseteq, \equiv, \cong, \models, \#$

$\bigcup, \bigcap, \bigvee$

$\mathcal{AC}, \mathcal{C}, \mathcal{FAC}, \mathcal{FC}$–script AC , C, FAC

$\mathcal{FOL}, \mathcal{M}, \mathcal{Q}, \mathcal{E}$–script FOL,M,Q,E

$\alpha, \beta, \eta, \iota, \mu, \nu$–lower-case Greek alpha,beta,eta,iota,mu,nu

$\pi, \rho, \phi, \psi, \omega$–lower-case Greek pi,rho,phi,psi,omega

Introduction.

In this paper we give a logical characterization of an important family of regular languages (also called *recognizable languages*): Those whose syntactic monoids contain only solvable groups. These languages turn out to be precisely the sets definable in an extension of first-order logic by generalized quantifiers of the form 'there exist exactly r elements modulo q such that...'. We obtain various subfamilies by imposing restrictions on the use of these quantifiers in the defining formulas. This work is simultaneously a generalization of a theorem of McNaughton and Papert [14] characterizing the so-called star-free languages by first-order formulas, and a transfer of the work of Straubing [21] and Thérien [24] from the domain of semigroup theory to logic.

It was shown by Büchi [7] that the regular languages are precisely those definable by sentences in certain monadic second-order theories of linear order. McNaughton [14] considered the star-free regular languages—those that can be built from the letters of the alphabet by repeated application of boolean operations and the concatenation product—and showed that these are precisely the languages defined by first-order sentences. In this context, a word is viewed as a finite ordered model, and properties of words are formalized in the corresponding first-order language. For example, the set of words in which no more than two letters are a is defined by the first-order sentence:

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

In this sentence, the variables x, y and z are interpreted as positions in the word, and $Q_a x$ is interpreted to mean that the letter in position x is a.

Cyclic counting, that is, counting with respect to an integer modulus, cannot be described in first-order logic over ordered models with the unary predicates Q_a. Indeed, the ordered models that have an even number of elements with some given property serve in the literature as a typical example of a class that is not first-order definable. (See, for example, [10],[13],[28],[1].) Such counting does, however, keep us within the family of regular languages, and thus one obtains new families of regular languages by adjoining quantifiers that perform cyclic counting. The same generalized quantifiers are studied, in a different context, by Paris and Wilkie [15].

We are able to precisely describe the class of regular languages definable in this extension of first-order logic—in fact, we can effectively
determine if a given regular language is so definable—by appeal to previous work describing the structure of regular languages in terms of algebraic properties of the syntactic monoid

of the language. (See 1.1 for the definition of the syntactic monoid.) The earliest work in this vein was the theorem of Schützenberger [20] showing that a language is star-free if and only if its syntactic monoid contains no nontrivial groups. Straubing [21] and Thérien [24] studied new operations on languages based on cyclic counting, and showed that the languages that could be built by means of these operations, together with the boolean operations and concatenation, are precisely those whose syntactic monoids contain only solvable groups. Our principal result (Theorem 2 below) shows that these are also the languages obtained with the cyclic counting quantifiers. (In particular, we do not obtain all the regular languages by means of these new quantifiers, since there are regular languages whose syntactic monoids contain non-solvable groups.) This theorem is presented in Section 1.

In Section 2 we extend our results to sets of infinite words. This is based on earlier work of Arnold [2], Perrin [16] and Thomas [25] concerning regular sets of ω-words.

We are particularly interested in these results because of their connection to an algebraic and automata-theoretic approach to boolean circuit complexity, recently studied by Barrington and Thérien [6]. Barrington [3] introduced the complexity class ACC, defined in terms of constant-depth families of circuits with unbounded fan-in in which both boolean gates and cyclic counting gates are allowed to appear. Our result can be viewed as describing a 'very uniform' version of ACC, and may be relevant to Barrington's conjecture that ACC is strictly contained in the complexity class NC^1 of languages accepted by log-depth bounded fan-in boolean circuits. We discuss these connections in Section 3.

In Section 4 we review related work and list some open questions.

In a preliminary version of this article [23] we proved the main theorems in Sections 1 and 2 by working directly with the generalized first-order formulas and finding a kind of normal form for these formulas. In this we followed earlier work of Thomas [26] on the logical characterization of the dot-depth hierarchy of regular languages. Our paper contained several errors, and while these can easily be repaired, we thought it better to present an alternative approach. We give a very different proof of the main theorem, using arguments that are far more algebraic in character. The method used owes much to the work of Rhodes and Tilson [19] on finite categories, although we do not explicitly mention categories here. Our new proof also uses an idea of Perrin and Pin [17], who treat formulas with free variables as defining languages over an extended alphabet.

1. The Characterization Theorem.

1.1 The families of languages.

Let A be a finite alphabet. A^* denotes the set of all words over the alphabet A. Algebraically, A^* is a *monoid;* that is, it has an associative multiplication (concatenation of words) and an identity element (the empty word). If $L, L' \subseteq A^*$, $a \in A$, and $0 \leq r < q$, then (L, a, L', q, r) denotes the set of words w in A^* such that the number of factorizations $w = vav'$, with $v \in L$, $v' \in L'$, is congruent to r modulo q. LaL' denotes the set of words $w = vav'$, where $v \in L$, and $v' \in L'$. Let $\mathcal{AC}$ be the smallest family of languages in A^* that contains the empty set and is closed under boolean operations and the operations

$$(L, L') \mapsto (L, a, L', q, r),$$

and

$$(L, L') \mapsto LaL'.$$

Let P be a set of primes, and let $\mathcal{AC}(P)$ be the smallest family of languages containing the empty set and closed under the operations above, with the restriction $q \in P$.
Let $\mathcal{C}$ be the smallest family of languages closed under boolean opearations and the first of the two operations above; $\mathcal{C}(P)$ is defined similarly, with the restriction $q \in P$. (In these notations, $\mathcal{A}$ stands for 'aperiodic', and $\mathcal{C}$ for 'counting'. Observe that $\mathcal{AC}(\emptyset)$ is the family of star-free languages, and that $\mathcal{AC} = \mathcal{AC}(P)$, where P is the set of all primes.)
If $L \subseteq A^*$, an equivalence relation $\equiv_L$ on A^* is defined as follows: $u \equiv_L v$ if and only if

$$\{(x, y) : x, y \in A^*, xuy \in L\} = \{(x, y) : x, y \in A^*, xvy \in L\}.$$

This equivalence relation is a congruence on A^* (that is, it is compatible with the concatenation of words) and thus the quotient forms a monoid, called the *syntactic monoid* of L, denoted $M(L)$. The map sending a word in A^* to its equivalence class is then a homomorphism of monoids, called the *syntactic morphism* of L, and denoted μ_L. (By definition, a homomorphism of monoids both preserves the multiplication and maps the identity element of the domain monoid to the identity of the codomain.) A language is *regular* (also called *recognizable*) if and only if the syntactic monoid is finite. (See Eilenberg [9], or Pin [17] for a full account.) All the language families defined in the preceding paragraph are contained in the family of regular languages in A^*, and all can be characterized in terms of the syntactic monoids of the languages they contain. The characterizations are given by Proposition 1 below, whose proof may be extracted from [24] , where the results are stated in terms of congruences rather than in terms of operations on languages. See also [21], where an almost identical theorem is proved using the unary operation $L \mapsto (L, a, A^*, q, r)$ in place of the binary operation $(L, L') \mapsto (L, a, L', q, r)$.

Proposition 1.

(a) $L \in \mathcal{AC}$ if and only if $M(L)$ is finite and every group in $M(L)$ is solvable.

(b) $L \in \mathcal{AC}(P)$ if and only if $M(L)$ is finite, every group in $M(L)$ is solvable, and every prime dividing the order of these groups is in P.

(c) $L \in \mathcal{C}$ if and only if $M(L)$ is a solvable group.

(d) $L \in \mathcal{C}(P)$ if and only if $M(L)$ is a solvable group, and every prime dividing the order of $M(L)$ is in P.

1.2 Syntax of logical formulas.

We build formulas using

variables: $x, x_0, x_1, ...; y, y_0, y_1, ..., etc.$

boolean connectives: $\wedge, \neg$. ($p \vee q$ is, as usual, an abbreviation for $\neg(\neg p \wedge \neg q)$.)

non-logical symbols: $<, Q_a (a \in A)$. ($x = y$ is an abbreviation for $\neg((x < y) \vee (y < x))$.)

existential quantifier: $\exists$ ($\forall x \phi$ is an abbreviation for $\neg \exists x \neg \phi$.)

modular quantifiers: $\exists^{(q,r)}$ with $0 \leq r < q$.

The usual syntactic rules for the formation of formulas apply. Q_a is treated as a unary predicate. If the variables occurring free in a formula ϕ are in $\{x_1, ..., x_n\}$ we sometimes write $\phi(x_1, ..., x_n)$. If ϕ has no variables occurring free it is a *sentence.*

We can now define the following formula classes: $\mathcal{FAC}$ denotes the family of all formulas; $\mathcal{FC}$ denotes the family of formulas in which only modular quantifiers (that is, no ordinary existential quantifiers) are used. If P is a set of primes then $\mathcal{FAC}(P)$ denotes the family of those formulas in $\mathcal{FAC}$ in which the moduli in the modular quantifiers all belong to P, and $\mathcal{FC}(P)$ denotes the intersection of $\mathcal{FAC}(P)$ and $\mathcal{FC}$.

Throughout the paper, 'formula', means a formula of $\mathcal{FAC}$ with respect to a given alphabet A, unless we explicitly restrict the formula to belong to a subclass of $\mathcal{FAC}$.

1.3 Semantics of logical formulas; statement of the main theorem.

A *word structure* is a pair $(w, (r_1, ..., r_k))$, $k \geq 0$, where $w \in A^*$, and each r_i is a position in w; that is, $r_i \in \{1, ..., |w|\}$. We abbreviate $(r_1, ..., r_k)$ by $\mathbf{r}$. If $k = 0$ we simply write w for the word structure $(w, \mathbf{r})$. If $(w, \mathbf{r}) = (w, (r_1, ..., r_k))$ is a word structure and $\phi(x_1, ..., x_k)$ is a formula whose free variables are in $\{x_1, ..., x_k\}$, we define $(w, \mathbf{r}) \models \phi(x_1, ..., x_k)$ by induction, as follows:

$$(w, \mathbf{r}) \models x_i < x_j$$

if and only if $r_i < r_j$.

$$(w, \mathbf{r}) \models Q_a x_i$$

if and only if the letter of w in position r_i is a.

$$(w, \mathbf{r}) \models \exists x_{k+1} \phi(x_1, ..., x_k, x_{k+1})$$

if and only if there is a position r in w such that $(w, (\mathbf{r}, r)) \models \phi(x_1, ..., x_{k+1})$.

$$(w, \mathbf{r}) \models \exists^{(q,s)} x_{k+1} \phi(x_1, ..., x_k, x_{k+1})$$

if and only if the number of positions r in w such that $(w, (\mathbf{r}, r)) \models \phi(x_1, ..., x_{k+1})$ is congruent to s modulo q.

Boolean connectives have their usual interpretation.

If $(w, \mathbf{r}) \models \phi$ we say that the word structure *satisfies* ϕ. Observe that there are no word structures $(w, \mathbf{r})$ where w is the empty word 1 and $\mathbf{r}$ is a nonempty sequence. Thus we have

$$1 \models \neg \exists x \phi$$

and

$$1 \models \exists^{(q,0)} x \phi$$

for any formula ϕ having at most one free variable x.

If ϕ is a sentence then the set of all words $w \in A^*$ such that $w \models \phi$ is a language, denoted L_ϕ. We say that the sentence ϕ *defines* L_ϕ.

Here is our principal result.

Theorem 2.

(a) $\{L_\phi : \phi \in \mathcal{FAC}\} = \mathcal{AC}$.

(b) For any set P of primes, $\{L_\phi : \phi \in \mathcal{FAC}(P)\} = \mathcal{AC}(P)$.

(c) $\{L_\phi : \phi \in \mathcal{FC}\} = \mathcal{C}$.

(d) For any set P of primes, $\{L_\phi : \phi \in \mathcal{FC}(P)\} = \mathcal{C}(P)$

The proof of Theorem 2 will be given in 1.5 and 1.6. Observe that part *(b)* with $P = \emptyset$ is the characterization of star-free languages as the languages defined by first-order sentences.

1.4 Examples.

In all of these examples, the underlying alphabet is $A = \{a, b\}$.

(a) Let L_1 consist of all words in $\{a, b\}^*$ that contain the letter a. The syntactic monoid M contains two elements—the congruence class of all words without a, which is the identity of M, and the class of all words with a, which is the zero. As this monoid is *aperiodic* (that is, it contains no nontrivial groups), Theorem 2 implies that L_1 is defined by a sentence in $\mathcal{FAC}(\emptyset)$—an ordinary first-order sentence. An example of such a defining sentence is

$$\exists x Q_a x.$$

It also follows from Theorem 2 that L_1 is not definable by any sentence that uses only modular quantifiers.

(b) Let L_2 consist of all words in $\{a, b\}^*$ in which the number of occurrences of a is even. (L_2 is called the PARITY language.) The congruence class of a word is determined by the number, modulo 2, of occurrences of a in the word. Thus $M(L_2)$ is the group of order 2. It thus follows from Proposition 1 that $L_2 \in \mathcal{C}(\{2\})$. By Theorem 2, L_2 is definable by a sentence in $\mathcal{FC}(\{2\})$, namely

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

We can also conclude from Theorem 2 that L_2 is not definable by any sentence of $\mathcal{FAC}(P)$, where P is the set of odd primes.

(c) Let L_3 consist of those words in which the number of runs of consecutive letters a is even. For example,

$$aababbaabaaab$$

contains four such runs, and is thus in L_3. It is easy to verify that the congruence class of a word is completely determined by the number, modulo 2, of such runs, together with the first letter of the word and the last letter of the word. The empty word is in a congruence class by itself. Thus the syntactic monoid has nine elements. Each of the two-element subsets of the syntactic monoid formed by specifying the first and last letter is a group, and these are the maximal groups in the syntactic monoid. Thus our theorem tells us that L_3 is definable by a sentence in $\mathcal{FAC}(\{2\})$, but not by any first-order sentence, nor by any sentence using only modular quantifiers.

Let us see how to construct a sentence that defines L_3. A position is the start of a run of consecutive a's if it contains a, and if it is either the first position in the word, or the preceding letter is b. Thus the defining sentence is

$$\exists^{(2,0)}x(Q_a x \wedge ((x = 1) \vee \exists y((x = y + 1) \wedge Q_b y))).$$

Here '$x = 1$' is an abbreviation for

$$\neg\exists z(z < x),$$

and '$x = y + 1$' is an abbreviation for

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

(d) Let $n > 1$. Let us think of a and b as permutations of $\{1, \ldots, n\}$; a denotes the transposition $(1 \quad 2)$ and b denotes the n-cycle $(1 \quad 2 \quad \cdots \quad n)$. Thus to each word in $\{a, b\}^*$ we can associate the composition of the underlying sequence of permutations in $\mathcal{S}_n$, the symmetric group of degree n. Let U_n denote the set of words such that this associated permutation fixes 1. U_n is the language recognized by the automaton pictured in Figure 1 (for the case $n = 5$).

FIGURE 1

It is well known that a and b generate all of $\mathcal{S}_n$. Given any two distinct permutations π_1 and π_2 we can find permutations ρ_1 and ρ_2 such that $\rho_1\pi_1\rho_2$ fixes 1 and $\rho_1\pi_2\rho_2$ does not. From these two facts it follows that the syntactic monoid of U_n is isomorphic to $\mathcal{S}_n$.

An interesting consequence emerges: U_n can be defined by sentences of $\mathcal{FC}(\{2,3\})$ if $n = 3$ or $n = 4$, but cannot be defined by any sentence of $\mathcal{FAC}$ if $n > 4$. This follows from our theorem along with the solvability of the groups $\mathcal{S}_3$ and $\mathcal{S}_4$, and the nonsolvability of $\mathcal{S}_n$ for $n > 4$. In particular, we cannot define all regular languages within the logical framework introduced here.

It is not evident how to produce defining sentences for U_4, or even for U_3. In fact there is a somewhat cumbersome algorithm for doing this: From a composition series for the group, one can construct a wreath product decomposition of the group, and from this, an expression for the language in terms of the operations $L \mapsto (L, a, A^*, q, r)$ (Straubing [21]). The first part of our proof of Theorem 2, given below, shows how to derive a defining sentence from such an expression.

Let us give such a sentence explicitly in the case $n = 3$. Consider $w \in \{a, b\}^*$. Since $ab = b^2a$ in $\mathcal{S}_3$, we may move all the a's in w to the right to obtain a word $b^r a^s$ that represents the same permutation as w. Here s is simply the number of occurrences of a in w. To determine r, we replace each b in w by b^{2^m} where m is the number of a's that precede the given occurrence of b—r is then the sum of the 2^m over all occurrences of b in w. Since $b^3 = 1$, we can just as well replace the occurrence of b in w by b^2 if m is odd and by b if m is even; we can then take r to be the number of occurrences of b in the result. The permutations that fix 1 are the identity and ba. Thus $w \in L_3$ if and only if either $r \equiv 0 \pmod 3$ and s is even, or $r \equiv 1 \pmod 3$ and s is odd. Let α_0 (respectively, α_1) denote the number of occurrences of b in w such that the number of occurrences of a that precede this occurrence of b is even (respectively odd). Then '$\alpha_i \equiv j \pmod 3$' is expressed by

$$\phi_{i,j} \equiv \exists^{(3,j)} x(Q_b x \wedge \exists^{(2,i)} y((y < x) \wedge Q_a x)).$$

U_3 is defined by the sentence

$$[(\exists^{(2,0)} x Q_a x) \wedge ((\phi_{0,0} \wedge \phi_{1,0}) \vee (\phi_{0,1} \wedge \phi_{1,1}) \vee (\phi_{0,2} \wedge \phi_{1,2}))] \vee$$
$$[(\exists^{(2,1)} x Q_a x) \wedge ((\phi_{0,0} \wedge \phi_{1,2}) \vee (\phi_{0,1} \wedge \phi_{1,0}) \vee (\phi_{0,2} \wedge \phi_{1,1}))].$$

1.5 Formulas from languages.

We first prove $\mathcal{AC} \subseteq \{L_\phi : \phi \in \mathcal{FAC}\}$. The analogous inclusions for parts *(b)-(d)* of Theorem 2 are proved in an identical fashion. This is the easy part of Theorem 2.

First note that for every sentence $\phi \in \mathcal{FAC}$ there is a formula $\phi[< x] \in \mathcal{FAC}$ (called a *relativization* of ϕ) having one free variable x, such that $(w, (r)) \models \phi[< x]$ if and only if $u \models \phi$, where u is the prefix of w of length $r - 1$. $\phi[< x]$ is obtained from ϕ by replacing all occurrences of $\mathcal{Q} y \psi$, where $\mathcal{Q}$ is a quantifier $\exists$ or $\exists^{(q,r)}$ and y is a variable, by $\mathcal{Q} y((y < x) \wedge \psi)$. Similarly, there is a formula $\phi[> x]$ such that $(w, (r)) \models \phi[> x]$ if and only if $u \models \phi$, where u is the suffix of w of length $|w| - r$. The following four facts are all easy to verify:

(i) $\emptyset = L_\psi$, where $\psi \equiv \exists x(x < x)$.

(ii) $A^* \backslash L_\psi = L_{\neg\psi}$; $L_\phi \cap L_\psi = L_{\phi \wedge \psi}$.

(iii) $L_\phi a L_\psi = L_\chi$, where $\chi \equiv \exists x(\phi[< x] \wedge \psi[> x] \wedge Q_a x)$.

(iv) $(L_\phi, a, L_\psi, q, r) = L_\chi$, where $\chi \equiv \exists^{(q,r)} x(\phi[< x] \wedge \psi[> x] \wedge Q_a x)$.

It now follows by induction on the construction of L that if $L \in \mathcal{AC}$, then $L = L_\phi$ for some sentence $\phi \in \mathcal{FAC}$.

1.6 Languages from formulas.

In this section we shall show the inclusions from left to right in Theorem 2. Our proof uses the algebraic characterizations of the classes $\mathcal{C}(P)$ and $\mathcal{AC}(P)$ given in Proposition 1. We say that a group G has *exponent* k if $g^k = 1$ for all $g \in G$. A group G is said to be an *extension* of a group H by a group K if H is a normal subgroup of G, and G/H is isomorphic to k.

Lemma 3. Let $\alpha : A^* \to S$, $\beta : A^* \to T$, and $\eta : S \to T$ be homomorphisms of monoids, where S and T are finite, and $\eta \circ \alpha = \beta$. Suppose there exists $k > 0$ such that for every $u \in A^*$ for which $\beta(u)$ is idempotent, $\alpha(u)^k = \alpha(u)^{k+1}$. Then every group in $\alpha(A^*)$ is isomorphic to a group in $\beta(A^*)$.

Proof. Let $G \subseteq \alpha(A^*)$ be a group. Then $H = \eta(G)$ is a group in T. Let $\alpha(u)$ be in the kernel of the restriction of η to G. Then $\beta(u)$ is idempotent, so by assumption $\alpha(u)^k = \alpha(u)^{k+1}$. This implies that $\alpha(u)$ is the identity of G. Thus the restriction of η to G has a trivial kernel, so G is isomorphic to $\eta(G)$.□

Lemma 4. Let $\alpha, \beta, \eta, S, T$ be as in Lemma 3. Let $q > 1$. Suppose that for all $u, x, y, v \in A^*$ satisfying

$$\beta(ux) = \beta(u), \beta(xv) = \beta(v), \beta(uy) = \beta(u), \beta(yv) = \beta(v),$$

we have

$$\alpha(uxyv) = \alpha(uyxv)$$

and

$$\alpha(ux^qv) = \alpha(uv).$$

Then every group in $\alpha(A^*)$ is an extension of an abelian group of exponent q by a group in T. Furthermore, if $\beta(A^*)$ is itself a group then $\alpha(A^*)$ is a group.

Proof. Let G be a group in $\alpha(A^*)$. Let $u, v \in A^*$ be such that $\alpha(u) = \alpha(v)$ is the identity of G; and let $x, y \in A^*$ be such that $\alpha(x), \alpha(y)$ belong to the kernel of the restriction of η to G. Then

$$\beta(ux) = \beta(u), \beta(xv) = \beta(v), \beta(uy) = \beta(u), \beta(yv) = \beta(v),$$

so by assumption

$$\alpha(x)\alpha(y) = \alpha(y)\alpha(x),$$

and $\alpha(x)^q$ is the identity of G. Thus the kernel of the restriction of η to G is an abelian group of exponent q.

Suppose further that T is a group of exponent r. Let $y \in A^*$, $x = y^r$, and $u = v = 1$. Then $\beta(x) = \beta(u) = \beta(v)$ is the identity of T, and thus $\alpha(y^{qr}) = \alpha(x^q) = \alpha(1)$ is the identity of $\alpha(A^*)$. Thus $\alpha(A^*)$ is a group of exponent qr. □

A word structure $(w, (r_1, \ldots, r_k))$ can be viewed in a natural manner as a word over the extended alphabet $A \times 2^{\{x_1,\ldots,x_k\}}$: The j^{th} letter of this word is $(a, \{x_i : r_i = j\})$, where a is the j^{th} letter of w. With this convention, a formula ϕ whose free variables are in $\{x_1, \ldots, x_k\}$ defines a language L_ϕ over this extended alphabet; L_ϕ is the set of all word structures that satisfy ϕ. Thus we can, in particular, form the syntactic monoids of these languages. It is important to note that not every word over this extended alphabet is a word structure, and that the concatenation of two word structures is never a word structure, unless the set of variables is empty. The word structures are those words in which each of the variables x_i appears exactly once within the second components of the letters of the word.

We will often use a restricted version of the syntactic monoid and the syntactic morphism: Let $\iota : A^* \to (A \times 2^{\{x_1,\ldots,x_k\}})^*$ denote the homomorphism defined by mapping each $a \in A$ to $(a, \emptyset)$. Given $L \subseteq (A \times 2^{\{x_1,\ldots,x_k\}})^*$, set $\nu_L = \mu_L \circ \iota : A^* \to M(L)$ and let $N(L) = \nu_L(A^*)$. We usually regard A as a subset of $A \times 2^{\{x_1,\ldots,x_k\}}$ with the inclusion given by ι.

To illustrate the distinction, let $A = \{a, b\}$, and let ϕ be the formula $Q_a x$. Then ϕ defines a language L over the extended alphabet $A \times 2^{\{x\}}$. L consists of all words over the extended alphabet in which exactly one letter is $(a, \{x\})$, and every remaining letter is $(a, \emptyset)$ or $(b, \emptyset)$. The syntactic monoid of L is $\{1, s, 0\}$, where $s^2 = 0$. The syntactic morphism μ_L maps $(a, \emptyset)$ and $(b, \emptyset)$ to 1, $(a, \{x\})$ to s, and $(b, \{x\})$ to 0. However $N(L)$ is the trivial monoid 1.

When we use this interpretation of word structures we must be careful to avoid the situation where the same variable is used in two different contexts in a formula. (For example $\exists x(x < y) \wedge \exists x(x > y)$.) While there is no problem in interpreting such a formula, the induction in the proofs we give below will not work correctly. So we shall assume that in our formulas no variable appears both bound and free, and each bound variable that appears is in the scope of only one quantifier. Of course, there is no loss of generality entailed in such an assumption, since we can replace any sentence by an equivalent one in which this condition is met.

Lemma 5. Let ϕ be a formula with free variables in $\{x_1, \ldots, x_k\}$. Then $M(L_\phi)$ is finite.

*Proof.*Equivalently, we have to prove that L_ϕ is a regular language. First observe that the set $\mathcal{S}$ of all word structures over a fixed set of free variables is a regular language, since we can check with a finite automaton that each variable occurs exactly once. $L_{Q_a x}$ and $L_{x<y}$ are regular languages, since in the first case we can check with a finite automaton that a word contains a letter of the form (a, P), with $x \in P$, and in the second case that a word contains an occurrence of the variable x before an occurrence of the variable y. We obtain $L_{Q_a x}$ and $L_{x<y}$ by intersecting the languages recognized by these automata with $\mathcal{S}$. If L_ϕ and L_ψ are regular languages, then $L_{\phi\wedge\psi} = L_\phi \cap L_\psi$ and $L_{\neg\phi} = S \backslash L_\phi$ are regular languages, by the usual closure properties of regular languages.

Suppose now that L_ϕ is a regular language recognized by a nondeterministic finite automaton

$$\mathcal{M} = (Q, i, F, \mathcal{E}),$$

where Q is the set of states, i the initial state, F the set of final states, and $\mathcal{E} \subseteq Q \times (A \times 2^{\{x_1,\ldots x_k\}}) \times Q$ the set of edges. We will construct an automaton that recognizes $L_{\exists x\phi}$. We define

$$\mathcal{M}' = (Q \times \{0, 1\}, (i, 0), F \times \{1\}, \mathcal{E}'),$$

where $\mathcal{E}'$ consists of two kinds of edges:

$$((q, u), (a, P), (q', u)),$$

where $u \in \{0, 1\}$, $x \notin P$, and $(q, (a, P), q') \in \mathcal{E}$, and

$$((q, 0), (a, P\backslash\{x\}), (q', 1)),$$

where $x \in P$, and $(q, (a, P), q') \in \mathcal{E}$. A word w is accepted by this automaton if and only if there is some way to adjoin x to a letter of w and obtain an element of L_ϕ. Thus the automaton recognizes $L_{\exists x\phi}$, and consequently $L_{\exists x\phi}$ is regular. We treat the modular quantifiers similarly: Let ψ be the formula $\exists^{(q,r)} x\phi$. Given $\mathcal{M}$ as above, we define

$$\mathcal{M}' = (Q \times \mathbf{Z}_q, (i, 0), F \times \{r\}, \mathcal{E}').$$

Again $\mathcal{E}'$ contains two kinds of edges: those of the first kind defined above, and edges of the form

$$((q,u),(a,P\backslash\{x\}),(q',(u+1) \bmod q)z)),$$

where $u \in \mathbf{Z}_q$, $x \in P$, and $(q,(a,P),q') \in \mathcal{E}$. This automaton recognizes L_ψ. □

Lemma 6. Let ϕ be a formula with free variables in $\{x_1,\ldots,x_{k+1}\}$. Let $\psi \equiv \exists x_{k+1}\phi$. Then every group in $N(L_\psi)$ is a quotient of a group in $N(L_\phi)$.

Proof. Let $S = N(L_\psi)\times N(L_\phi)$, $T = N(L_\phi)$, $\beta = \nu_{L_\phi}$, $\alpha = (\nu_{L_\psi},\beta) : A^* \to S$. Suppose $w \in A^*$ and $\beta(w)$ is idempotent. We claim $\alpha(w)^3 = \alpha(w)^4$. Indeed, let $u, v \in (A \times 2^{\{x_1,\ldots,x_k\}})^*$; if $uw^3v \in L_\psi$, then there is a word obtained by adjoining x_{k+1} to some letter of uw^3v that belongs to L_ϕ. The resulting word still has a factor equal to w, and since $\beta(w)$ is idempotent, we may replace this occurrence of w by w^2 and still obtain a word in L_ϕ. Thus $uw^4v \in L_\psi$. Conversely if $uw^4v \in L_\psi$, then we obtain a word in L_ϕ by adjoining x_{k+1} to some letter. The resulting word still contains a factor equal to w^2, which we can replace by w and leave the resulting word in L_ϕ. Thus $uw^3v \in L_\psi$. We have proved $\nu_{L_\psi}(w^3) = \nu_{L_\psi}(w^4)$. Since $\beta(w) = \beta(w^2)$, we obtain $\alpha(w)^3 = \alpha(w^3) = \alpha(w^4) = \alpha(w)^4$. We may now apply Lemma 3 and conclude that every group in $\alpha(A^*)$ is isomorphic to a group in $\beta(A^*) = N(L_\phi)$. Now let G be a group in $N(L_\psi)$. Then $\{\alpha(w) : \nu_{L_\phi}(w) \in G\}$ is a subsemigroup of $\alpha(A^*)$ that has G as a quotient. By Lemma 5, this subsemigroup is finite. It is easy to show that the smallest subsemigroup mapping onto G must itself be a group in $\alpha(A^*)$, and hence isomorphic to a group in $N(L_\phi)$.□

Lemma 7. Let ϕ be a formula with free variables in $\{x_1,\ldots,x_{k+1}\}$. Let $\psi \equiv \exists^{(q,r)}x_{k+1}\phi$, with $q > 1$. Then every group in $N(L_\psi)$ is a quotient of an extension of an abelian group of exponent r by a group in $N(L_\phi)$. If $N(L_\phi)$ is itself a group, then $N(L_\psi)$ is a group.

Proof. Let S, T, α, β be as in the proof of Lemma 5. Suppose $u, v, y, z \in A^*$ satisfy

$$\beta(uy) = \beta(uz) = \beta(u), \qquad \beta(yv) = \beta(zv) = \beta(v).$$

Consider w_1uyzvw_2, where $w_1, w_2 \in (A \times 2^{\{x_1,\ldots,x_k\}})^*$. Suppose that adjoining x_{k+1} to a letter in this word yields a word in L_ϕ. If the letter occurs in y, let y' denote the word obtained from y by the adjunction of x_{k+1}. We then have $w_1uy'zvw_2 \in L_\phi$, whence $w_1uy'vw_2 \in L_\phi$, and thus $w_1uzy'vw_2 \in L_\phi$. We can argue similarly wherever else the letter occurs. Thus there is a one-to-one correspondence between the letters in w_1uyzvw_2 and the letters in w_1uzyvw_2 for which adjoining x_{k+1} to the letter gives a word in L_ϕ. This shows

$$\nu_{L_\psi}(uyzv) = \nu_{L_\psi}(uzyv),$$

and consequently

$$\alpha(uyzv) = \alpha(uzyv).$$

Now consider $w_1uy^qvw_2$. Let y' be obtained from y by adjoining x_{k+1} to some letter. Then $w_1uy'y^{q-1}vw_2 \in L_\phi$ if and only if $w_1uyy'y^{q-2}vw_2 \in L_\phi$ if and only if $w_1uy^{q-1}y'vw_2 \in L_\phi$. It follows readily the number of letters in $w_1uy^qvw_2$ for which adjoining x_{k+1} to the letter gives a word in L_ϕ is congruent modulo q to the number of such letters in w_1uvw_2. Thus $w_1uy^qvw_2 \in L_\psi$ if and only if $w_1uvw_2 \in L_\psi$. So

$$\nu_{L_\psi}(uy^qv) = \nu_{L_\psi}(uv),$$

and thus

$$\alpha(uy^qv) = \alpha(uv).$$

We may now apply Lemma 4. The desired conclusion follows as in Lemma 6. □

Lemma 8. *(a)* Let ϕ_1 and ϕ_2 be formulas with free variables in $\{x_1, \ldots, x_k\}$. Let $\psi \equiv \phi_1 \wedge \phi_2$. Then every group in $N(L_\psi)$ is a quotient of a group in $N(L_{\phi_1}) \times N(L_{\phi_2})$. If further $N(L_{\phi_1})$ and $N(L_{\phi_2})$ are themselves groups, then $N(L_\psi)$ is a group.

(b) Let ϕ be a formula with free variables in $\{x_1, \ldots, x_k\}$. Then $N(L_\phi)$ and $N(L_{\neg\phi})$ are isomorphic.

Proof.

(a) Let $w_1, w_2 \in A^*$, and suppose $\nu_{L_{\phi_i}}(w_1) = \nu_{L_{\phi_i}}(w_2)$ for $i = 1, 2$. It follows directly from the definition of the syntactic congruence that $\nu_{L_\psi}(w_1) = \nu_{L_\psi}(w_2)$. Thus ν_{L_ψ} factors through the homomorphism

$$(\nu_{L_{\phi_1}}, \nu_{L_{\phi_2}}) : (A \times 2^{\{x_1,\ldots,x_k\}})^* \to N(L_{\phi_1}) \times N(L_{\phi_2}),$$

from which the desired conclusions follow.

(b) Let $w_1, w_2 \in A^*$ and suppose $\nu_{L_\phi}(w_1) = \nu_{L_\phi}(w_2)$. Let $u, v \in (A \times 2^{\{x_1,\ldots,x_k\}})^*$, and suppose $uw_1v \in L_{\neg\phi}$. Then, in particular, uw_1v is a word structure, so uw_2v is a word structure. If $uw_2v \models \phi$ we would have $uw_1v \models \phi$, a contradiction. Thus $uw_2v \in L_{\neg\phi}$. The identical argument shows that $uw_2v \in L_{\neg\phi}$ implies $uw_1v \in L_{\neg\phi}$, so $w_1 \equiv_{L_{\neg\phi}} w_2$, and thus $\nu_{L_{\neg\phi}}(w_1) = \nu_{L_{\neg\phi}}(w_2)$. Symmetrically, $\nu_{L_{\neg\phi}}(w_1) = \nu_{L_{\neg\phi}}(w_2)$ implies $\nu_{L_\phi}(w_1) = \nu_{L_\phi}(w_2)$.

Note. For formulas ϕ with free variables, $M(L_\phi)$ and $M(L_{\neg\phi})$ are not, in general, isomorphic. For example, let $\phi \equiv ((x < y) \wedge (y < x))$. Then L_ϕ is the empty language, whose syntactic monoid is trivial, but $L_{\neg\phi}$ consists of all word structures in $(A \times 2^{\{x,y\}})^*$, and thus has a nontrivial syntactic monoid.

Proof of Theorem 2. Let ϕ be a formula with free variables in $\{x_1, \ldots, x_k\}$. We claim that for every group G in $N(L_\phi)$, G is finite and solvable, and its cardinality divides a product of the moduli of the modular quantifiers in ϕ. We claim further that if all the quantifiers in ϕ are modular quantifiers, then $N(L_\phi)$ is a group. In the case where ϕ is a sentence, these claims along with Proposition 1 give the Theorem.

We prove the claims by induction on the construction of ϕ. In the case where ϕ is an atomic formula, all words in A^* are congruent under the syntactic congruence of L_ϕ, and consequently $N(L_\phi)$ is trivial (and, in particular a group). Lemma 5 tells us that $N(L_\phi)$ will remain finite throughout all steps of the construction. Lemmas 6 and 7 show that the required properties are preserved by the application of quantifiers, and Lemma 8 shows that the required properties are preserved by boolean operations.□

2. ω-languages

The results of Section 1 have natural counterparts in the domain of ω-languages; that is, sets of infinite sequences of letters from a finite alphabet A. In this section we give a theorem, based on Theorem 2 above, that characterizes a class of regular ω-languages, analogous to the class $\mathcal{AC}$. As before, the characterization is in terms of finite monoids, language theory, and logic. We obtain similar characterizations for subclasses of these languages defined by restricting the moduli with respect to which we count.

Regular ω-languages were introduced by Büchi [8]. For a general discussion, see Thomas [27]. Here we will just review the essential points. A^ω denotes the set of all infinite sequences over the alphabet A. If $v \in A^+$, then v^ω denotes the sequence $vvv\cdots$. If $L \subseteq A^+$, then L^ω denotes the subset of A^ω consisting of all infinite sequences

$$w_1 w_2 \cdots,$$

where $w_i \in L$ for all $i \geq 1$. We denote by $\lim L$ the subset of A^ω consisting of all sequences v such that infinitely many initial segments of v belong to L.

There are a number of equivalent definitions of regular subsets of A^ω; we shall use the following one: $L \subseteq A^\omega$ is a regular ω-language if and only if L is a finite union of sets of the form $L_1 L_2^\omega$, where $L_1, L_2 \subseteq A^+$ are regular languages. There is also a characterization in terms of a notion of acceptance by finite automata ("Büchi automata"; see, for example, [27]).

Let $L \subseteq A^\omega$. The syntactic congruence of L, denoted $\sim_L$, is a congruence on A^* defined as follows: $x \sim_L y$ if and only if

$$\{(u,v,w) \in A^* \times A^* \times A^+ : uxvw^\omega \in L\} = \{(u,v,w) \in A^* \times A^* \times A^+ : uyvw^\omega \in L\},$$

and

$$\{(u,v) \in A^* \times A^+ : u(xv)^\omega \in L\} = \{(u,v) \in A^* \times A^+ : u(yv)^\omega \in L\}.$$

The syntactic congruence was introduced in Arnold [2]. The quotient monoid $A^*/{\sim_L}$ is called the syntactic monoid of L, and is denoted, as in the case of sets of finite words, $M(L)$. If L is a regular ω-language, then $M(L)$ is finite, although the converse is false.

Our logical formulas of Section 1 can be interpreted in infinite words. The only thing that needs to be added is that we say a word structure satisfies $\exists^{(q,r)} x\phi(x)$ if and only if the number of positions x satisfying $\phi(x)$ is *finite* and congruent to r modulo q. In order to show that every ω-language defined by a formula is regular, one can use the automaton-theoretic characterization of the regular ω-languages, combined with a straightforward adaptation of the constructions of Lemma 5.

Here is our theorem:

Theorem 9. Let P be a set of primes, and let $L \subseteq A^\omega$. The following are equivalent.

(a) L is a regular ω-language, and every group in $M(L)$ is solvable, and every prime dividing the order of these groups is in P.

(b) L can be obtained from A^ω by boolean operations and left concatenation with languages in $\mathcal{AC}(P)$.

(c) L is the ω-language defined by a sentence in $\mathcal{FAC}(P)$.

The case $P = \emptyset$ was treated by Thomas [25]. Before proceeding to the proof, let us remark why we do *not* get an analogous result for the ω-languages defined by sentences of $\mathcal{FC}(P)$: Let $A = \{a, b\}$, and let $L \subseteq A^\omega$ be the set of sequences that contain finitely many occurrences of the letter a. Then L is defined by the sentence

$$\exists^{(2,0)} xQ_a x \vee \exists^{(2,1)} xQ_a x.$$

The analogue to Theorem 2 would imply that $M(L)$ is a group; however the $\sim_L$-classes are b^* and $A^*\backslash b^*$, and thus $M(L)$ has two elements, an identity and a zero, and is not a group.

Proof of Theorem 9.

$(a) \Rightarrow (b)$. It is proved by Arnold [2] that if L is a regular ω-language whose syntactic monoid has the form described, then L is a finite union of sets of the form $L_1 L_2^\omega$, where $L_1, L_2 \subseteq A^*$ are regular languages such that every group in $M(L_1)$ and $M(L_2)$ is solvable, with cardinality dividing a product of primes in P. Thus, by Proposition 1, $L_1, L_2 \in \mathcal{AC}$. It now follows from a result of Perrin [16] concerning regular ω-languages whose syntactic monoids belong to concatenation-closed varieties that L is a boolean combination of sets of the form $\lim K$, where $K \in \mathcal{AC}$. It remains to show that $\lim K$ can be constructed in the manner described in part *(b)* of the theorem. Consider the set $A^\omega \backslash \lim K$. This consists of all sequences that have either no prefix in K, or a longest prefix in K. The set of sequences with no prefix in K is $A^\omega \backslash K A^\omega$, which is constructed in the required form. To treat the set of sequences with a longest prefix in K, consider the left quotients $v^{-1}K = \{w \in A^* : vw \in K\}$, where $v \in A^*$. Define $v_1 \cong v_2$ if and only if $v_1^{-1}K = v_2^{-1}K$. It is well known that recognizability of of K implies that $\cong$ is an equivalence relation on A^* of finite index, and that K is a union of $\cong$-classes. Let $\{v_1, \ldots, v_k\}$ be representatives of the $\cong$-classes contained in K; the classes themselves are denoted $[v_1], \ldots, [v_k]$. The set of sequences with a last prefix in K is

$$\bigcup_{i=1}^{k} [v_i](A^\omega \backslash ((v_i^{-1}K)A^\omega)).$$

The syntactic monoid of $v_i^{-1}K$ is a homomorphic image of the syntactic monoid of K, and thus $v_i^{-1}K \in \mathcal{AC}$. (That is $\mathcal{AC}$ is closed under left quotients.) We can express the equivalence class $[v]$ in terms of the right quotients:

$$[v] = \Big(\bigcap_{u \in v^{-1}K} K u^{-1} \Big) \Big\backslash \Big(\bigcup_{u \notin v^{-1}K} (A^* \backslash K) u^{-1} \Big).$$

Since K and $A^*\backslash K$ are regular languages, they have only finitely many distinct right quotients, and thus the infinite index sets for the above union and intersection can be replaced by finite index sets. Since $\mathcal{AC}$ is closed under boolean combinations and right quotients, each $[v_i]$ in in $\mathcal{AC}$. It now follows that $\lim K$ can be expressed in the required form.

$(b) \Rightarrow (c)$. Assume $L \subseteq A^\omega$ has the form described in *(b)*. If $L = A^\omega$ then L is defined by

$$\forall x(x = x).$$

If $K \in \mathcal{AC}(P)$ then by Theorem 2, K is the set of finite words defined by some $\phi \in \mathcal{FAC}(P)$. Suppose $L \subseteq A^\omega$ is the ω-language defined by some sentence $\psi \in \mathcal{FAC}(P)$. We now relativize ϕ and ψ as in 1.5, forming formulas $\phi[< x]$ and $\psi[\geq x]$ with one free variable. KL is thus the ω-language defined by

$$\exists x(\phi[< x] \wedge \psi[\geq x]).$$

Thus the family of ω-languages defined by formulas in $\mathcal{FAC}(P)$ is contains A^ω and is closed under concatenation on the left with members of $\mathcal{AC}(P)$. Since it is obviously closed under boolean operations, the desired result follows.

$(c) \Rightarrow (a)$. As in Section 1, we can view sets of infinite word structures as ω-languages over an extended alphabet, and define the restricted syntactic morphism and syntactic monoid for such sets. The proof now essentially consists in verifying that Lemmas 5, 6, 7 and 8 remain true in this new context. We have already noted that the analogue of Lemma 5 holds: every ω-language defined by a formula (possibly with free variables) is regular, and thus has a finite syntactic monoid. The proof of Lemma 8 is unchanged. Let us note what modifications are necessary in the proofs of Lemma 6 and 7.

As in Lemma 6, let $\psi \equiv \exists x_{k+1}\phi$. Let $\nu_{L_\phi}(w) = \nu_{L_\phi}(w^2)$. Now suppose $uw^3vy^\omega \in L_\psi$. Then we can adjoin x_{k+1} to a letter of this sequence and obtain a sequence in L_ϕ. If this letter occurs in one of the factors, u, v or w^3, then we proceed just as in the proof of Lemma 6. If the letter occurs in the factor y^ω, then we can write

$$uw^3vy^my'y^\omega \in L_\phi,$$

thus

$$uw^4vy^my'y^\omega \in L_\phi,$$

so

$$uw^4vy^\omega = uw^4vy^{m+1}y^\omega \in L_\psi.$$

Similarly, suppose $u(w^3v)^\omega \in L_\psi$. Then there is some letter in the the sequence to which we can adjoin x_{k+1} and obtain a sequence in L_ϕ. If the letter occurs in the factor u then we proceed as in Lemma 6. If the letter occurs elsewhere (say, for example, in one of the occurrences of v,) then we can write

$$u(w^3v)^m w^3 v'(w^3v)^\omega \in L_\phi.$$

We can apply $m+1$ times the fact that w and w^2 are equivalent under ν_{L_ϕ} and obtain

$$u(w^4v)^m w^4 v'(w^3v)^\omega \in L_\phi,$$

and once more to obtain

$$u(w^4v)^m w^4 v'(w^4v)^\omega \in L_\phi.$$

Thus $u(w^4v)^\omega \in L_\psi$. The same sort of argument shows that $uw^4vy^\omega, u(w^4v)^\omega \in L_\psi$ implies $uw^3vy^\omega \in L_\psi$ and $u(w^3v)^\omega \in L_\psi$, respectively.

We turn to the extension of Lemma 7. Keeping the same notations as in the proof of Lemma 7, let

$$\beta(uy) = \beta(uz) = \beta(u), \beta(yv) = \beta(zv) = \beta(v),$$

and let $w = w_1uyzvw_2w_3^\omega \in L_\psi$, where $\psi \equiv \exists^{(q,r)}x_{k+1}\phi$. By definition, there are only finitely many letters in w for which adjoining x_{k+1} gives a word in L_ψ. We can thus rewrite $w = w_1uyzvw_2w_3^m w_3^\omega$, so that all the letters in question occur in the factor $w_1uyzvw_2w_3^m$. We can then prove as before that all the letters of $w' = w_1uzyvw_2w_3^\omega$ for which adjoining x_{k+1} gives a sequence in L_ψ occur in the factor $w_1uzyvw_2w_3^m$, and that there is a one-to-one correspondence between the sets of such letters in w and w'. A similar refactorization is used to prove the other parts of the extension of Lemma 7.

3. Connections With Circuit Complexity.

Recently Barrington and Thérien [6] demonstrated a close connection between the computational capabilities of constant-depth families of circuits with unbounded fan-in, and the classification of regular languages according to their syntactic monoids. The obvious similarity between circuits and quantified logical formulas, coupled with our results connecting such formulas to the syntactic monoid, enables us to give a straightforward new proof of their result.

For our purposes a *circuit* with n inputs is a directed acyclic graph with a single sink. Each source node is labelled either x_i or $\neg x_i$, for some $i = 1, \ldots, n$. Other nodes are labelled either *AND, OR* or $MOD_{q,r}$, where $0 \leq r < q$. The *depth* of the circuit is the length of the longest path from a source to the sink, and the *size* is the number of nodes. Given a sequence of n bits $a_1 \cdots a_n \in \{0,1\}^*$, we assign a bit to each node, beginning with the source nodes: A node labelled x_i is assigned the bit a_i, and a node labelled $\neg x_i$ is assigned the value $1 - a_i$. A node labelled AND is assigned the conjunction of the values assigned to its predecessors, and a node labelled OR is assigned the disjunction of the values assigned to its predecessors. A node labelled $MOD_{q,r}$ is assigned 1 if and only if the number of its predecessors having the value 1 is congruent to r modulo q. The input string is accepted if and only if the sink node is assigned the value 1. Ordinarily we consider families of circuits, one for each input length $n \geq 1$. The resulting family of circuits thus recognizes a language over $\{0,1\}$. An important problem in computational complexity theory is to find the minimum size and depth of circuits required to recognize certain languages. Here we restrict our attention to circuit families in which the depth is constant and the size is bounded by a polynomial in the number of inputs.

An *expression* is a circuit whose underlying graph is a tree. Equivalently, expressions can be defined inductively as follows: If $1 \leq i \leq n$, x_i and $\neg x_i$ are expressions. If $E_1, \ldots, E_m$ are expressions, then

$$OR(E_1, \ldots, E_m)$$

$$AND(E_1, \ldots, E_m)$$

$$MOD_{q,r}(E_1, \ldots, E_r)$$

are expressions. It is easy to show that a circuit of size s and depth d can be replaced by an equivalent expression of size no more than s^{2d} and depth d. In particular, a family of circuits of constant depth and polynomial size yields a family of expressions of the same depth and polynomial size.

A *program* with n inputs over a language $L \subseteq A^*$ is a sequence of *instructions*

$$(i_1, a_1, b_1), \ldots, (i_r, a_r, b_r),$$

where for all j, $1 \leq i_j \leq j$ and $a_j, b_j \in A$. Given a string $c_1 \ldots c_n \in \{0,1\}^n$, the program emits the string $d_1 \cdots d_r \in A^r$, where $d_j = a_j$ if $c_{i_j} = 1$, and $d_j = b_j$ if $c_{i_j} = 0$. We say that the j^{th} instruction of the program *queries* the i_j^{th} bit of the input. The input $c_1 \cdots c_n$ is accepted if and only if $d_1 \cdots d_r \in L$. As before we consider families of programs, one for each input length n. The resulting family recognizes a language $S \subseteq \{0,1\}^*$. We are particularly interested in the case where the programs in the family are over a fixed regular language L, and the lengths of the programs are bounded by a polynomial in the input length.

Here is the theorem of Barrington and Thérien:

Theorem 10. Let P be a set of primes. Let $L \subseteq \{0,1\}^*$. The following are equivalent:

(a) L is recognized by a polynomial-size constant-depth family of circuits in which every $MOD_{q,r}$ gate has the same modulus q, where q is a product of primes in P.

(b) L is recognized by a polynomial-length family of programs over a language $L' \in \mathcal{AC}(P)$.

Proof.

$(a) \Rightarrow (b)$. As we saw above, we may assume that the polynomial-size constant-depth family of circuits is a family of expressions. Let us say that the depth of a node in an expression is the length of the longest path from the node to a source node. We may assume that the expression is *levelled* in the sense that all predecessors of a node of depth $s+1$ are nodes of depth s; indeed, if there is a predecessor node of depth $s-k$, with $k>0$, then we may place a path of k nodes labelled OR between the two nodes. Such leveling preserves the overall depth of the expression and requires at worst a polynomial increase in size.

We now write the expression in prefix form; that is, with each node preceding its predecessor nodes, without commas or parentheses. In order to parse the resulting string unambiguously, we affix to each node of depth >0 a superscript indicating the depth. Thus the expression

$AND(OR(x_1, \neg x_2, x_3), MOD_{2,1}(\neg x_1, x2))$

is rewritten

$AND^2 \quad OR^1 \quad x_1 \quad \neg x_2 \quad x_3 \quad MOD^1_{2,1} \quad \neg x_1 \quad x_2.$

Finally, given an assignment of binary values to the inputs $x_1, \ldots x_n$, we replace the occurrences of x_i and $\neg x_i$ in the prefix expression by their assigned values. Thus to an expression and an input word we have associated a word over the alphabet

$A = \{0,1\} \cup \{AND^i : 1 \leq i \leq d\} \cup \{OR^i : 1 \leq i \leq d\} \cup \{MOD^i_{q,r} : 1 \leq i \leq d, 0 \leq r < q\}.$

(Here d is the depth of the expression.) Observe that the length of this word is bounded by a polynomial in the length of the input. Let C_d denote the set of words in A^* that result from such expression-word pairs, where the expression has depth d. A word in A^* is in C_d if and only if the first letter has depth d, no other letter has depth d, and the successor of each letter of depth $t>0$ is a letter of depth $t-1$. (Letters of depth 0—that is, the bits 0 and 1—may be followed by letters of any depth less than d.) It is easy to write a first-order sentence ϕ_{C_d} that defines C_d.

Now let T_d denote the set of strings in C_d obtained from expression-word pairs in which the expression accepts the word. (That is, where the expression evaluates to 1 when the n bits of the word are substituted for the variables $x_1, \ldots, x_n$ in the expression.) T_d can be defined inductively: T_0 is just the singleton $\{1\}$. A word w is in T_{t+1} if and only if it is in C_{t+1}, and one of the following three conditions holds: *(i)* The first letter of w is AND^{t+1} and every maximal segment of w in C_t is in T_t; *(ii)* the first letter of w is OR^{t+1} and some maximal segment of w in C_t is in T_t; *(iii)* the first letter of w is $MOD_{q,r}^{t+1}$ for some r, and the number of maximal segments of w in C_t that are also in T_t is congruent to r modulo q. Using the relativization technique of 1.5 it is easy to express in first order logic 'x is the start of a maximal segment in C_t': Either no position after x has a letter of depth t and $\phi_{C_t}[\geq x]$ holds, or there exists a y that is the first position after x with a letter of depth t, and $\phi_{C_t}[\geq x, < y]$ holds. Assuming, by the inductive hypothesis, that we have a formula defining T_t, we may use the same idea to write a formula that says that x is the start of a maximal segment in C_y, and this segment is also in T_t. When we put this together with the three conditions above we obtain a sentence defining T_{d+1}. Observe that the modular quantifiers in this sentence all have modulus q. These can be replaced by modular quantifiers with moduli in P. Thus $T_d \in \mathcal{AC}(P)$.

It remains to construct a program over T_d that, given an input word in $\{0,1\}^n$, emits the corresponding word in C_d. This is trivial to do, since the j^{th} letter of the output word depends upon at most one letter of the input word. Thus if the j^{th} letter of the output word is of the type AND, OR or MOD, the j^{th} instruction of the program can query any bit of the input and emit the appropriate letter, independent of the value of the input bit. If the j^{th} letter of the output word is 0 or 1, then it replaced x_i or $\neg x_i$ in the original expression. In this case the j^{th} instruction of the program is $(i, 1, 0)$ or $(i, 0, 1)$.

$(b) \Rightarrow (a)$. Now suppose L is accepted by a family of programs over $L' \in \mathcal{AC}(P)$. By Theorem 1, L' is defined by a sentence ϕ of $\mathcal{FAC}(P)$. Beginning with the outermost quantifiers, we replace every occurrence of

$$\neg \exists x \psi$$

in ϕ by

$$\forall x \neg \psi,$$

and every occurrence of

$$\neg \exists^{(q,r)} x \psi$$

by

$$\bigvee_{r' \neq r} \exists^{(q,r')} x\psi.$$

The resulting sentence is equivalent to ϕ, and the only occurrences of negation are at the level of the atomic formulas. We may thus assume that ϕ has this form.

We again rewrite ϕ, this time as an expression. Beginning with the outermost quantifiers, we replace

$$\exists x\psi(x)$$

by

$$OR(\psi(1), \ldots, \psi(n)),$$

$$\forall x\psi(x)$$

by

$$AND(\psi(1), \ldots, \psi(n)),$$

and

$$\exists^{(q,r)} x\psi(x)$$

by

$$MOD_{q,r}(\psi(1), \ldots, \psi(n)).$$

We also replace $\psi_1 \vee \psi_2$ by $OR(\psi_1, \psi_2)$ and $\psi_1 \wedge \psi_2$ by $AND(\psi_1, \psi_2)$. Finally we replace the atomic formulas: $i < j$ is replaced by $OR(x_1, \neg x_1)$ if $i < j$ is true and by $AND(x_1, \neg x_1)$ otherwise. $\neg(i < j)$ is treated similarly. To rewrite $Q_a i$, we look at the i^{th} instruction of the program: (j, b, c). If $b = a$ and $c \neq a$ we replace $Q_a i$ by x_j. If $b = c = a$ we replace $Q_a i$ by $OR(x_1, \neg x_1)$. The other two cases are treated similarly.

It is clear from the construction of this expression that it accepts precisely the words of length n in L, has constant depth, and size bounded by a polynomial in n.□

We mention that the class of languages recognized by circuit families of polynomial size and constant depth, with only AND and OR gates is called AC^0 in the literature in computational complexity. If modular gates for a fixed modulus q are added, the resulting class of languages is denoted $ACC(q)$. The union of the $ACC(q)$ over all q is denoted ACC.

4. Related Research and Open Questions.

As we saw in Example *(d)* of Section 1, the framework of first-order logic with modular quantifiers is not sufficient to define all regular languages. In the monograph of McNaughton and Papert [14] a number of extensions to first-order logic are considered. One is the introduction of fixed unary predicates $x \equiv r \pmod q$. Let us denote by $\mathcal{FOL}$ the class of languages definable by first-order sentences (by Theorem 2, this is the class $\mathcal{AC}(\emptyset)$ of Section 1), and by $\mathcal{FOL}_C$ the class of languages definable using these new predicates. It is easy to define the new predicates in terms of modular quantifiers, so $\mathcal{FOL}_C$ is contained in the class $\mathcal{REG}$ of regular languages. In fact $\mathcal{FOL}_C$ lies strictly between $\mathcal{FOL}$ and $\mathcal{AC}$. The set of strings of even length is defined by the sentence

$$\forall y(\forall x(x \leq y) \Rightarrow y \equiv 0 \pmod 2),$$

and is thus in $\mathcal{FOL}_C$. However, its syntactic monoid is a nontrivial group, and thus by Theorem 2, it is not in $\mathcal{FOL}$. It is shown in Chapter 11 of [14], and also in Barrington, *et. al.,* [4] that the language $PARITY$ of Example 1.4.b, consisting of all binary strings with an even number of 1s, is not in $\mathcal{FOL}_C$, and thus $\mathcal{FOL}_C$ is strictly contained in $\mathcal{AC}$.

A second extension of first-order logic investigated by McNaughton and Papert is the use of an auxiliary predicate U whose interpretation depends upon the given language. The predicate U indicates which proper prefixes of a given word belong to the language under consideration. Here we will use a formal definition of this predicate that is equivalent to the one given by McNaughton and Papert, but somewhat simpler for the present purposes. If $L \subseteq A^*$, then the U*-expansion* of L consists of all strings

$$(a_1, i_1) \cdots (a_r, i_r) \in (A \times \{0, 1\})^*$$

such that $a_1 \cdots a_r \in L$, and such that $i_j = 1$ if and only if $a_1 \cdots a_{j-1} \in L$. Given a class of languages $\mathcal{L}$, we define $\mathcal{L}_U$ to be the class of languages whose U-expansions lie in $\mathcal{L}$. We note that the language $PARITY$ belongs to $\mathcal{FOL}_U$: To see this, observe that a word belongs to the U-expansion of $PARITY$ if and only if *(a)* whenever 1 appears in the first component of the i^{th} letter, the second components of the i^{th} and $(i+1)^{th}$ letters are different, and conversely; and *(b)* the first and second components of the last letter are different. These conditions can easily be expressed by a first-order sentence.

The use of the U-predicate also takes us out of the class $\mathcal{AC}$. Consider again the language L of Example *(d),* of Section 1, with $n = 5$. The set of all strings that lead from the initial state back to the initial state without visiting this state at an intermediate step can be defined by a first-order sentence. This can be used to obtain a first-order sentence that defines the U-expansion of L. On the other hand, the U-predicate keeps us within the family of regular languages. Indeed, suppose that the U-expansion of a language L is a regular language K. We can test whether $w \in L$ by nondeterministically guessing the second component of its U-expansion, and then testing each prefix of the guess with an automaton that recognizes K.

A summary of the language classes thus defined is given in the next theorem. $\mathcal{FOL}_{CU}$ denotes the class of languages defined using formulas in $\mathcal{FOL}_C$ together with the U-expansion.

Theorem 11. The inclusions among the classes $\mathcal{FOL}$, $\mathcal{FOL}_C$, $\mathcal{FOL}_U$, $\mathcal{FOL}_{CU}$, $\mathcal{AC}$, $\mathcal{AC}_U$ and the class $\mathcal{REG}$ of all regular languages are given in the following diagram. Each arrow indicates proper inclusion, and the absence of a path indicates incomparability.

FIGURE 2

Proof. All claims of inclusion (but not proper inclusion) are straightforward, in light of the remarks above. Languages separating $\mathcal{FOL}$, $\mathcal{FOL}_C$, $\mathcal{FOL}_U$, and $\mathcal{FOL}_{CU}$ are given in Section 11.4 of [14]. We have noted above that $PARITY$ separates $\mathcal{AC}$ from $\mathcal{FOL}_C$, so it remains to prove that the three rightmost inclusions in the diagram are strict.

As explained above, the language L of example *(d)* in Section 1 separates $\mathcal{AC}_U$ from $\mathcal{AC}$. For the remaining two claims, it will be useful to introduce an endmarker letter $\#$. When this is appended to a word, the U-predicate becomes useless. It is easy to verify that $PARITY\#$ separates $\mathcal{AC}_U$ from $\mathcal{FOL}_{CU}$, since $PARITY$ separates $\mathcal{AC}$ from $\mathcal{FOL}_C$. Similarly, if L is the language separating $\mathcal{AC}_U$ from $\mathcal{AC}$ cited above, then $L\#$ separates $\mathcal{AC}_U$ from REG.

It is shown in Chapter 11 of [14] that the classes $\mathcal{FOL}_C$ and $\mathcal{FOL}_{CU}$ are incomparable. It remains to show that $\mathcal{AC}$ and $\mathcal{FOL}_U$ are incomparable, and that $\mathcal{AC}$ and $\mathcal{FOL}_{CU}$ are incomparable. $PARITY\#$ belongs to $\mathcal{AC}$ but not to either of the other two classes; and $L\#$ (the same L as in the preceding paragraph) belongs to $\mathcal{FOL}_U$, and hence to $\mathcal{FOL}_{CU}$, but not to $\mathcal{AC}$. □

Since the appearance of an earlier version of this paper, the modular quantifiers introduced here have been applied in a number of studies centering around circuit complexity. We will mention a few of these applications here. Barrington, *et. al.,* [4] and Straubing [22] studied the question of what regular languages can be recognized by polynomial-size constant-depth circuit families containing, respectively, boolean gates, modular gates, and both boolean and modular gates. In the first instance they showed that the answer is precisely the class $\mathcal{FOL}_C$ defined above (answering in the process a question posed by McNaughton and Papert). In the other two cases they showed that the answers are $\mathcal{C}$ and $\mathcal{AC}$ if and only if certain conjectures in circuit complexity hold. It is an outstanding open problem whether the class ACC contains all languages recognizable by families of circuits of depth $O(\log n)$ with AND and OR gates that have only two inputs. (In complexity-theoretic terminology, the question is whether the classes ACC and NC^1 are equal.) If this is true, then ACC contains all regular languages; if it is false (as most people believe), then all the regular languages in ACC belong to $\mathcal{AC}$.

Let us indicate how to pose these questions in a purely logical form. Suppose we allow generalized first-order sentences in which the atomic formulas are $Q_a x$ and any arbitrary *numerical* predicates. (By a numerical predicate we mean any k-ary relation over the positive integers; that is, its truth depends only on the positions substituted for the variables, and not on the letters that appear in those positions.) Thus $x < y$, $x \equiv 0 \pmod 2$, 'x is prime', are numerical predicates, while $Q_a x$ is not.) If we consider only ordinary quantifiers, the class of languages defined is AC^0 (this is proved in Immerman [12]). If we allow modular quantifiers as well, the class of languages defined is ACC. Thus the results of [4] say that if a first-order sentence with arbitrary numerical predicates defines a regular language, the formula can be rewritten so as to use only the predicates $x < y$ and $x \equiv 0 \pmod q$. The corresponding statement for modular quantifiers is an equivalent form of the conjecture that ACC is strictly contained in NC^1.

Barrington, Immerman and Straubing [5] consider sentences in which the only numerical predicate is a binary predicate $BIT(x, y)$, meaning that the x^{th} bit in the binary expansion of y is 1. The predicate $x < y$ can be defined in terms of this predicate, so we obtain all the regular languages discussed in this paper, as well as many nonregular ones. It is shown in [5] that the families of languages defined by first-order sentences and generalized first-order sentences using the BIT predicate are natural uniform versions of the complexity classes AC^0 and ACC.

Finally, suppose we introduce a new 'majority quantifier' M acting on k-tuples of variables, so that

$$Mx_1 \cdots x_k \phi$$

means that $\phi(x_1, \ldots, x_k)$ holds for at least half the k-tuples of positions. If we allow arbitrary numerical predicates in our sentences, then by using such quantifiers we obtain exactly the non-uniform circuit complexity class TC^0, consisting of all languages recognized by constant-depth, polynomial-size families of circuits containing majority gates (originally described in [11]). If we use only the numerical predicate '$<$' we obtain a natural uniform version of this class ([5]). Even in the latter case we get some non-regular languages; for example, it is easy to define the set of all bit strings in which the number of occurrences of a exceeds the number of occurrences of b. It is not known whether ACC is properly contained in TC^0, and whether TC^0 is properly contained in NC^1. The latter inclusion is proper if and only if every regular language in TC^0 has a solvable syntactic monoid ([4]); the former inclusion is equivalent to the question of whether we can simulate the majority quantifier with modular quantifiers.

Acknowledgements. We would like to thank Pierre Péladeau and Marek Zawadowski, who pointed out errors in the earlier version of this paper.

References.

1. M. Ajtai, Σ_1^1 formulae on finite structures, *Annals of Pure and Applied Logic* **24** (1983), 1-48.

2.A. Arnold, A syntactic congruence for rational ω-languages, *Theoretical Computer Science* **39**, (1985), 333-336.

3. D. Mix Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Comp. Sys. Sci.* **38**, (1989) 150-164.

4.D. Mix Barrington, K. Compton, H. Straubing and D. Thérien, Regular Languages in NC^1, *J. Comp. Syst. Sci.* **44** (1992) 478-499.

5. D. Mix Barrington, N. Immerman and H. Straubing, On uniformity in NC^1, *J. Comp. Syst. Sci.* **41** (1990), 274-306.

6. D. Mix Barrington and D. Thérien, Finite monoids and the fine structure of NC^1, *J. Assoc. Comp. Mach.* **35**, 941-952 (1988).

7. J. Büchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundlagen Math.* **6**, (1960), 66-92.

8. J. Büchi, On a decision method in restricted second-order arithmetic,in E. Nagel, ed., Logic, Methodology and Philosophy of Science, Stanford University Press, 1962.

9. S. Eilenberg, Automata, Languages and Machines, vols. A &B, Academic Press, New York, 1974-1976.

10. Y. Gurevich, Logic tailored for complexity, *Lecture Notes in Mathematics* **1104**, Springer, Berlin, 175-216.

11. A. Hajnal, W. Maass, P. Pudlák, M. Szegedy and G. Turán, Threshold circuits of bounded depth, *Proc. 28th IEEE FOCS* (1987), 99-110.

12.N. Immerman, Languages that capture complexity classes, *SIAM J. Computing* **16**, (1987), 760-778.

13. R. Ladner, Application of model-theoretic games to discrete linear orders and finite automata, *Information and Control* 33, (1977), 281-303.

14. R. McNaughton and S. Papert, Counter-free Automata, MIT Press, Cambridge, Mass., 1971.

15. J. Paris and C. Willkie, Counting problems in bounded arithmetic, *Lecture Notes in Mathematics* **1130**, Springer, Berlin, 317-340.

16. D. Perrin, Variétés de semigroupes et mots infinis, *C. R. Acad. Sci. Paris* **295**, (1985), 595-598.

17. D. Perrin and J.E. Pin, First-order logic and star-free sets, *J. Comp. Sys. Sci.* **32**, (1986), 393-406.

18. J. E. Pin, Varieties of Formal Languages, Plenum, London, 1986.

19. J. Rhodes and B. Tilson, The kernel of monoid morphisms, *J. Pure and Applied Algebra* **62**, (1989), 227-268.

20. M. P. Schützenberger, On finite monoids having only trivial subgroups, *Information and Control* **8**, (1965), 190-194.

21. H. Straubing, Families of recognizable sets corresponding to certain varieties of finite monoids, *J. of Pure and Applied Algebra* **15**, (1979), 305-318.

22. H. Straubing, Constant-depth periodic circuits, International J. Algebra and Computation, **1** (1991), 49-87.

23. H. Straubing, D. Thérien and W. Thomas, Regular languages defined with generalized quantifiers, *Proc. 15th ICALP,* Lecture Notes in Computer Science **317**, 561-575 (1988).

24. D. Thérien, Classification of finite monoids: the language approach, *Theoretical Computer Science* **14**, (1981), 195-208.

25. W. Thomas, Star-free regular sets of ω-sequences, *Information and Control* **42**, (1979), 148-156.

26. W. Thomas, Classifying regular events in symbolic logic, *J. Computer and System Sciences* **25**, (1982), 360-376.

27. W. Thomas, Automata on infinite objects, in J. van Leeuwen, ed., Handbook of Theoretical Computer Science, North-Holland, 1990.

28. P. Wolper, Temporal logic can be more expressive, *Information and Control* **56**, (1983), 72-79.