Circuit Complexity and the Expressive Power of Generalized First-Order Formulas

Howard Straubing*

Department of Computer Science, Boston College
Chestnut Hill, Massachusetts, USA 02167

Abstract. The circuit complexity classes AC^0, ACC, and CC (also called pure-ACC) can be characterized as the classes of languages definable in certain extensions of first-order logic. All of the known and conjectured inclusions among these classes have been shown to be equivalent to a single conjecture concerning the form of the formulas required to define the regular languages they contain. (The conjecture states, roughly, that when a formula defines a regular language, predicates representing numerical relations on the positions in a string can be replaced by predicates computed by finite state automata.) Here this conjecture is established in a special case: It is shown that the conjecture holds for the subclasses of AC^0, ACC, and CC defined by restricting all the numerical predicates occurring in the logical formulas to be either unary relations, or the order relation $<$.

1 Introduction

Certain formulas of predicate logic can be interpreted in a natural way in strings over a finite alphabet. First-order variables are interpreted as positions in the string, and the first-order predicate symbols are interpreted as either purely numerical relations on the positions, or as asserting that a certain input letter is found at a given position. Higher-order variables, if there are any, are interpreted as ranging over sets of positions or relations on positions. Thus a sentence (a formula without free variables) defines a language—namely the set of all strings that satisfy the sentence—and a class of sentences defines a class of languages. (See the examples in Section 2, where a precise account of the logical formalism is given.)

This viewpoint has been used in at least two distinct ways: The first is the characterization of computational complexity classes in terms of formal logic. For example, Fagin [F], showed that existential second-order sentences define precisely the languages in NP, and Immerman [I] characterized a large number of complexity classes within P in terms of logic. He showed, among other results, that the nonuniform circuit complexity class AC^0, defined by constant-depth, polynomial-size boolean circuits with unbounded fan-in, consists of exactly those languages defined by first-order sentences with arbitrary numerical predicates.

The second use has been in the classification of finite automata: Büchi [Bu] showed that the regular languages are precisely those defined by second-order sentences with the numerical predicate $x < y$ on first-order variables and with monadic second-order variables. McNaughton [MP] showed that the first-order sentences with the numerical

* Supported by NSF Grant CCR-8902369

predicate $x < y$ define exactly the 'star-free' regular languages: These can be characterized as those languages represented by an extended regular expression without the use of the $*$ operator, or alternatively as the class of regular languages whose syntactic monoids contain no nontrivial groups. Straubing, Thérien and Thomas [STT] introduced *modular quantifiers,* and showed that using these alone or in combination with the ordinary first-order quantifiers and the numerical predicate $x < y$, one defines classes of regular languages that can also be characterized in terms of their associated syntactic monoids.

These two applications of the logical characterization of language classes turn out to be closely related. Barrington and Thérien [BT] have shown that many circuit complexity classes within NC^1, including NC^1 itself, can be characterized as the behavior of polynomial-length families of programs over finite monoids, where different classes of monoids yield different circuit complexity classes. (NC^1 consists of those languages accepted by polynomial-size log-depth families of bounded fan-in boolean circuits. The subclass CC consists of those languages accepted by polynomial-size constant-depth unbounded fan-in families of circuits in which each gate determines whether the sum of its input bits is divisible by q, for some fixed $q > 0$. The subclass ACC is defined similarly, except here one allows both the $MODq$ gates and unbounded fan-in boolean gates.) These programs can be viewed as generalizations of ordinary finite automata, so that, in a sense, the classification of regular languages according to the structure of the associated syntactic monoid extends to languages within NC^1.

In [BCST] these results on programs, together with McNaughton's theorem on regular languages defined with first-order sentences, are used to give a new proof of Immerman's logical characterization of the circuit complexity class AC^0. In [BCST] and [CDPC], the results of Straubing, Thérien and Thomas on modular quantifiers are used to give analogous characterizations of the classes ACC and CC.

At this point an interesting fact emerges: Let A be a finite alphabet. If $\mathcal{Q}$ is a set of quantifiers and $\mathcal{S}$ a set of numerical predicates, then $\mathcal{Q}[\mathcal{S}]$ denotes the class of languages defined by sentences using quantifiers in $\mathcal{Q}$, numerical predicates in $\mathcal{S}$, and, for each input letter $a \in A$, a monadic predicate Q_a that says that the letter in the given position is a. Let $\mathcal{N}$ be the set of all numerical predicates. The characterizations of AC^0, ACC and CC can then be written

$$AC^0 = \{\exists\}[\mathcal{N}],$$

$$CC = \{\exists_q^r : 0 \leq r < q\}[\mathcal{N}],$$

and

$$ACC = (\{\exists\} \cup \{\exists_q^r : 0 \leq r < q\})[\mathcal{N}],$$

where $\{\exists_q^r : 0 \leq r < q\}$ is the set of modular quantifiers of modulus q. Let $\mathcal{P}$ denote the set of numerical predicates that can be obtained by first-order quantification over predicates of the form $x \equiv 0 \pmod r$ and $x < y$. Let $\mathcal{R}$ denote the class of regular languages over A. The characterization of the regular languages in AC^0 given in [BCST] shows

$$\{\exists\}[\mathcal{N}] \cap \mathcal{R} = \{\exists\}[\mathcal{P}].$$

Remarkably enough, this result, which is equivalent to the circuit lower bounds of Furst, Saxe and Sipser [FSS] and Ajtai [A] was conjectured twenty years ago by McNaughton and Papert [MP], before the connection with circuit complexity was appreciated.

Let T be a set of integers, and let MOD_T be the set of modular quantifiers with moduli in T. It is conjectured that for any $\mathcal{Q} = MOD_T$, or $\mathcal{Q} = \{\exists\} \cup MOD_T$,

$$\mathcal{Q}[\mathcal{N}] \cap \mathcal{R} = \mathcal{Q}[\mathcal{P}].$$

This conjecture is equivalent to most of the conjectured properties of the complexity classes AC^0, ACC and CC. For instance, it implies, among other things, that the AND fucntion of n input bits cannot be computed in CC, and that ACC is properly contained in NC^1. Thus one approach to these open problems of circuit complexity is to study the question of the kind of sentences needed to define regular languages, in the hope of directly establishing the conjecture. (See [CDPC] and [BCST].)

In the present paper the conjecture is shown to hold for formulas in which each numerical predicate is either a monadic predicate or the order relation $<$. More precisely, let $\mathcal{M}$ be the class of all monadic numerical predicates. Then for any $\mathcal{Q} = MOD_T$, or $\mathcal{Q} = \{\exists\} \cup MOD_T$,

$$\mathcal{Q}[\{<\} \cup \mathcal{M}] \cap \mathcal{R} \subseteq \mathcal{Q}[\mathcal{P}].$$

The proof of this theorem relies on semigroup-theoretic constructions and previous work of Barrington and Straubing [BS] on the power of linear-size programs over finite monoids. In Section 2 I give formal definitions of the underlying logical apparatus and a precise statement of the main theorem. Section 3 contains the necessary background concerning finite semigroups, and Section 4 the proof of the main theorem.

2 The Logical Apparatus

Let A be a finite alphabet. Informally, the logical formulas used here are built from two kinds of atomic formulas: For each letter $a \in A$ there is a monadic predicate symbol Q_a, where $Q_a x$ is interpreted to mean 'the letter in position x is a'. The second kind of atomic formula is a *numerical predicate*. This can be of any arity (including 0)—the important property of a numerical predicate θ is that $\theta(x_1, \ldots, x_k)$ depends only on the positions x_i and the length of the string in which the formula is interpreted, and not on the letters that appear in those positions. Larger formulas are built from atomic formulas by applying boolean connectives and quantifiers. There are two kinds of quantifiers: the ordinary existential quantifier $\exists$ and modular quantifiers $\exists_q^r$, where $0 \leq r < q$. $\exists_q^r \Phi(x)$ is interpreted to mean 'there exist exactly $r \bmod q$ positions x such that $\Phi(x)$'. The universal quantifier $\forall$ can of course be defined in terms of $\exists$.

For example, let $A = \{a, b, c\}$. The set of all strings in A^* that contain an occurrence of a before some occurrence of b is defined by the sentence

$$\exists x \exists y (Q_a x \wedge Q_b y \wedge (x < y)).$$

The set of all strings with an odd number of occurrences of the factor ac is defined by the sentence

$$\exists_2^1 x (Q_a x \wedge \exists y (Q_c y \wedge (y = x + 1))).$$

In these two formulas the numerical predicates are the binary predicates $x < y$ and $y = x + 1$. There are several ways to define the language L consisting of all strings of even length. One can write

$$\exists_2^0 x (x = x).$$

Alternatively, one can introduce a monadic numerical predicate $\theta(x)$ asserting that x is an even-numbered position, and write

$$\forall x (\forall y (y \leq x) \longrightarrow \theta(x)),$$

which says that the last position is even. However, the numerical predicates are allowed to depend on the length of the string in which they are interpreted, so there is a perfectly legal 0-ary numerical predicate α with the interpretation 'the length is even', and thus L is also defined by the sentence

$$\alpha.$$

This framework must be defined more precisely for the purpose of proving theorems about formulas. Let $\{x_1, \ldots, x_r\}$ be a set of variables. A *numerical predicate* with free variables $x_1, \ldots, x_r$ is a subset S of $(2^{\{x_1, \ldots, x_r\}})^*$ such that for all

$$v = T_1 \cdots T_k \in S$$

one has

$$T_i \cap T_j = \emptyset,$$

if $i \neq j$, and

$$\bigcup_{i=1}^{k} T_i = \{x_1, \ldots, x_r\}.$$

A *word structure* with free variables $x_1, \ldots, x_r$ is a string

$$w = (a_1, T_1) \cdots (a_k, T_k) \in (A \times 2^{\{x_1, \ldots, x_r\}})^*$$

such that the string

$$T_1 \cdots T_k$$

satisfies the two conditions above. Formulas are built from numerical predicates and the predicates $Q_a x, a \in A$, by applying boolean operations and quantifiers. A formula with free variables $x_1, \ldots, x_r$ is interpreted in a word structure whose free variables include $x_1, \ldots, x_r$. It remains to define the satisfaction relation $\models$. Let w be a word structure and let N be any numerical predicate whose set of free variables is contained in the set of free variables of w. Then

$$w \models N$$

if and only if the string $\hat{w}$, obtained from w by erasing the letters of A and eliminating the variables that are not free variables of N, belongs to N.

$$w \models Q_a x$$

if and only if w includes a symbol (a, T) where $x \in T$. Boolean operations are interpreted in the usual fashion.

$$w \models \exists x \Phi(x)$$

if and only if x is not a free variable of w, and w includes a symbol (a, T) such that if v is the string that results from w upon replacing (a, T) by $(a, T \cup \{x\})$, then

$$v \models \Phi(x).$$

If the number of symbols with this property is congruent to r modulo q, then

$$w \models \exists_q^r x \Phi(x).$$

Let Ψ be a formula with free variables $x_1, \ldots, x_r$. Then L_Ψ denotes the set of all word structures w with these free variables such that $w \models \Psi$. If Ψ is a *sentence,* that is, a formula without free variables, then L_Ψ is a language in A^*.

The family of languages definable by sentences that use only the quantifier $\exists$ is denoted AC^0; the family of languages definable by sentences that use only modular quantifiers of modulus q is denoted $CC(q)$, and the family of languages definable by sentences that use both modular quantifiers of modulus q and $\exists$ is denoted $ACC(q)$. Observe that $AC^0 = ACC(1)$. Of course, these families were originally defined in terms of constant-depth circuits. The equivalence of the circuit definitions with these logical formulations is proved in [I] for AC^0, in [BCST] for $ACC(q)$, and in [CDPC] for $CC(q)$. Let mAC^0, $mACC(q)$, $mCC(q)$ denote the families obtained by restricting to sentences in which the only numerical predicates are the order relation $<$, and 0-ary or unary (*i.e.,* monadic) predicates, that is, predicates with no more than one free variable. In the notation of Section 1,

$$mAC^0 = \{\exists\}[\{<\} \cup \mathcal{M}],$$

and the other new classes are defined analogously. As in Section 1, let $\mathcal{P}$ denote the class of numerical predicates obtained as first-order formulas in numerical predicates of the form $x \equiv 0 \pmod r$ and $x < y$, and let $\mathcal{R}$ denote the class of regular languages over A. The main result of this paper is:

Theorem 2.1. For all $q > 0$,

$$mACC(q) \cap \mathcal{R} \subseteq (\{\exists\} \cup MOD_{\{q\}})[\mathcal{P}],$$

and

$$mCC(q) \cap \mathcal{R} \subseteq MOD_{\{q\}}[\mathcal{P}].$$

The corresponding assertion for the classes $ACC(q)$ and $CC(q)$ is known to hold when q is a prime power, but is in general, as noted in Section 1, an important open problem in circuit complexity.

The proof of Theorem 2.1 will be given in Section 4. Some notions concerning finite semigroups, necessary to the proof, will be given in the next section.

3 Background on Finite Semigroups

A *semigroup* is a set together with an associative multiplication. A *monoid* is a semigroup with an identity element; the identity element is denoted 1. A subset of a monoid closed under multiplication forms a subsemigroup of the monoid, and this subsemigroup may well be a group. I denote by $\mathbf{G}_q$ the family of all finite groups that are solvable and whose cardinality divides a power of q; and I denote by $\mathbf{M}_q$ the family of all finite monoids M such that all the groups contained in M belong to $\mathbf{G}_q$. The family $\mathbf{M}_1$ thus consists of all finite monoids containing only trivial groups—such monoids are said to be *aperiodic.*

Programs over finite monoids were defined by Barrington and Thérien [BT] and, implicitly, by Barrington [B]. Here we will only consider *single-scan* programs. A single-scan program π of length n over a finite monoid M consists of a subset X of M, called the set of *accepting values,* and a map

$$\phi : A \times \{1, \ldots, n\} \to M.$$

On input $a_1 \cdots a_n \in A^n$ the program emits the element

$$\phi(a_1, 1) \cdots \phi(a_n, n)$$

of M and accepts the input if and only if this emitted value is in X. (π can be thought of as a sequence of n *instructions*: each instruction emits an element of M depending on the input letter that is read, and the program accepts or rejects depending on the product of the emitted values.) Thus the program accepts a subset $|\pi|$ of A^n. This apparatus will also be used when the input alphabet is one of the extended alphabets $A \times 2^{\{x_1, \ldots, x_r\}}$ described in Section 2. In this case $|\pi|$ is defined to be the set of word structures accepted by π, rather than the set of all strings over the extended alphabet accepted by π.

Ordinarily, one considers *families of programs,* consisting of a program of length n over a fixed monoid M for each $n \geq 0$. The resulting family thus accepts a language $L \subseteq A^*$. An important special case is that in which there is a single set $X \subseteq M$ of accepting values for all programs in the family, and a single map $\psi : A \to M$ such that $\phi(a, i) = \psi(a)$ for all i and for all programs in the family. In this case ϕ extends to a homomorphism $\hat{\phi} : A^* \to M$, and the language accepted by the family is $L = \hat{\phi}^{-1}(X)$. This occurs if and only if L is a regular language. To any regular language L one can associate the smallest finite monoid for which such a homomorphism exists: To do this one defines an equivalence relation $\cong_L$ on A^*, by setting $u \cong_L v$ if and only if for all $x, y \in A^*$,

$$xuy \in L \Leftrightarrow xvy \in L.$$

Regularity of L implies that the index of $\cong_L$ is finite, and it is easy to see that equivalence is compatible with concatenation of strings. Thus the quotient set

$$M(L) = A^* / \cong_L$$

forms a monoid, called the *syntactic monoid* of L, and the map η_L that sends each string w to its $\cong_L$-class is a homomorphism, called the *syntactic morphism* of L. Observe that $\eta_L^{-1}(\eta_L(L)) = L$. If $L \subseteq A^+$ (the set of nonempty words over A) it is sometimes more convenient to deal with the *syntactic semigroup* $S(L)$ of L; this is simply the image of A^+ under η_L. (See Eilenberg [E] or Pin [P] for a detailed account of the matters discussed in this paragraph.)

Now let M be a finite monoid and K a finite set. A new monoid $\diamond(M, K)$ is defined as follows: The underlying set of $\diamond(M, K)$ is $M \times 2^{M \times K \times M}$, and the multiplication is given by

$$(m, X)(m', Y) = (mm', mY \cup Xm'),$$

where

$$mY = \{(mm_1, k, m_2) : (m_1, k, m_2) \in Y\},$$

and

$$Xm' = \{(m_1, k, m_2m') : (m_1, k, m_2) \in X\}.$$

It is easy to check that this multiplication is associative, and that $(1, \emptyset)$ is the identity. Thus $\diamond(M, K)$ is a monoid. $\diamond(M, K)$ and $\diamond_q(M, K)$, defined below, are closely related to the *Schützenberger product* and related constructions, and share many of the properties of these products. (see [E], [W]).

Let $q > 0$. Let $\mathbf{Z}_q[S]$ denote the free $\mathbf{Z}_q$-module generated by S. The underlying set of $\diamond_q(M, K)$ is $M \times \mathbf{Z}_q[M \times K \times M]$, and the multiplication is given by

$$(m, X)(m', Y) = (mm', mY + Xm'),$$

where left-multiplication by m and right-multiplication by m' are the unique linear transformations defined by

$$m(m_1, k, m_2) = (mm_1, k, m_2), (m_1, k, m_2)m' = (m_1, k, m_2m').$$

In both $\diamond(M, K)$ and $\diamond_q(M, K)$, projection onto the left coordinate is a homomorphism onto M. Consider the restriction of this homomorphism to a group in $\diamond(M, K)$ or $\diamond_q(M, K)$. It is not difficult to show that in the first case the kernel of this homomorphism is trivial, and that in the second case the kernel is an abelian group of exponent q. It is also easy to show that if M is a group then $\diamond_q(M, K)$ is a group. This gives:

Proposition 3.1. Let $q > 0$. If $M \in \mathbf{M}_q$ then $\diamond(M,K) \in \mathbf{M}_q$ and $\diamond_q(M,K) \in \mathbf{M}_q$. If $M \in \mathbf{G}_q$ then $\diamond_q(M,K) \in \mathbf{G}_q$. ∎

Let

$$r_i = (m_i, (1, k_i, 1)), 1 \leq i \leq n,$$

where this is interpreted either as an element of $\diamond(M,K)$, with the second component a singleton set, or as an element of $\diamond_q(M,K)$, with the second component viewed as a sum with a single term. In the proposition below, the expression $m_i \cdots m_j$ is taken to be 1 if $i > j$.

Proposition 3.2. In $\diamond(M,K)$,

$$r_1 \cdots r_n = (m_1 \cdots m_n, \{(m_1 \cdots m_{i-1}, k_i, m_{i+1} \cdots m_n) : 1 \leq i \leq n\}).$$

In $\diamond_k(M,K)$,

$$r_1 \cdots r_n = (m_1 \cdots m_n, \sum_{1 \leq i \leq n} (m_1 \cdots m_{i-1}, k_i, m_{i+1} \cdots m_n)).$$

Proof. This is straightforward by induction on n. ∎

4 Proof of Theorem 2.1

It will first be shown that every language in $mACC(q)$ or $mCC(q)$ is recognized by a family of single-scan programs over a monoid in $\mathbf{M}_q$ or $\mathbf{G}_q$, respectively. I will then apply the results of Barrington and Straubing [BS] concerning the behavior of linear-size programs over finite monoids.

It should be noted that the class $mCC(1)$ is somewhat anomalous, since application of a modular quantifier of modulus 1 gives a formula that is satisfied by all word structures with the right set of free variables. It follows from this that membership in a language in $mCC(1)$ (or, for that matter, in $CC(1)$) depends only on the length of the string, and thus Proposition 4.1 holds trivially. However the argument given below does not apply in this case.

Proposition 4.1. Let Φ be a formula with free variables $x_1, \ldots, x_r$, in which all numerical predicates are monadic, or $x < y$. If Φ uses only modular quantifiers of modulus q, then L_Φ is accepted by a family of single-scan programs over a monoid in $\mathbf{G}_q$. If Φ uses $\exists$ as well, then L_Φ is accepted by a family of single-scan programs over a monoid in $\mathbf{M}_q$.

Proof. The proof is by induction on the construction of Φ. First, if the assertion is true for Φ_1 and Φ_2, then it is true for $\Phi_1 \wedge \Phi_2$ and $\neg\Phi_1$. In the first case, the program is constructed by taking the cartesian product of the two programs over the direct product of the underlying monoids. In the second case, one keeps the same underlying monoid but changes the set of accepting values from X to $M \backslash X$. The conclusion follows from the fact that $\mathbf{M}_q$ and $\mathbf{G}_q$ are closed under direct products. Note that all the other boolean connectives can be expressed in terms of these two.

I now prove the assertion for quantifier-free formulas. By the preceding remarks concerning boolean operations, it suffices to treat each of the three kinds of formulas

$$Q_a x, \theta(x), x < y,$$

where θ is a numerical predicate. Consider the single-scan program π over a nontrivial monoid M that emits $m \neq 1$ on reading (a, S) if $x \in S$, and emits 1 otherwise. Let the accepting set be $\{m\}$; then $|\pi| = L_{Q_a x_i}$. (This is a slight abuse of notation: More precisely, this construction gives a program π for each length input, and the language accepted by the resulting family is $L_{Q_a x_i}$.) Note that the only requirement is that M be nontrivial, so M can be chosen to be in $\mathbf{M}_q$ for any q or in $\mathbf{G}_q$ if $q > 1$. Let ρ be the single-scan program over a nontrivial monoid M such that the j^{th} instruction emits $m \neq 1$ if and only if $\theta(j)$ and the j^{th} input letter (a, S) satisfies $x \in S$, and emits 1 otherwise. Again let the accepting set be $\{m\}$. Then $|\rho| = L_{\theta(x)}$, and, as before, M can be chosen to be in $\mathbf{M}_q$ for any q or in $\mathbf{G}_q$ if $q > 1$. Finally, if M is a nontrivial monoid then the *wreath product* $M \circ M$ is the monoid whose underlying set is $M^M \times M$, with multiplication given by

$$(f_1, m_1)(f_2, m_2) = (F, m_1 m_2),$$

where, for $m \in M$,

$$F(m) = f_1(m) f_2(m m_1).$$

As is well known (see, for example, [E]) $M \circ M$ is in $\mathbf{M}_q$ (respectively $\mathbf{G}_q$) if M is. Pick $m \in M \backslash \{1\}$. Let $I : M \to M$ be the map that sends every element to 1, and let $J : M \to M$ be the map that sends m to m and all other elements to 1. Consider the single-scan program σ over $M \circ M$ that, upon reading (a, S), emits (I, m) if $x \in S$ and $y \notin S$, $(J, 1)$ if $y \in S$ and $x \notin S$, and $(I, 1)$ otherwise. Let v be a word structure whose free variables include x and y. Upon reading v, σ emits the value (K, m), where $K(1) = m$ if $v \models x < y$, and where $K(1) = 1$ otherwise. Thus if the set of accepting values is $\{(K, m) : K(1) = m\}$, the program accepts $L_{x<y}$. Once again, $M \circ M$ can be chosen to be in $\mathbf{M}_q$ for any $q > 0$, and in $\mathbf{G}_q$ for any $q > 1$.

It remains to study the effect of quantification. Let Φ be a formula with free variables $x, x_1, \ldots, x_r$ for which the assertion holds: Then there is a family $\{\pi_n\}$ of single-scan programs over a monoid M that accepts L_Φ. Furthermore, M is in $\mathbf{M}_q$ or $\mathbf{G}_q$, depending on the quantifiers in Φ. Consider now a single-scan program π'_n over $\diamond(M, K)$, where $K = M$. The j^{th} instruction of π'_n, upon reading (a, S), emits $(m, \{1, m', 1)\})$, where m is the value emitted by the j^{th} instruction of π_n upon reading (a, S), and m' is the value emitted by the j^{th} instruction of π_n upon reading $(a, S \cup \{x\})$. Define the set of accepting values to be all pairs (m, Y) such that Y contains a triple (m_1, m_2, m_3) such that $m_1 m_2 m_3$ is an accepting value for π_n. It follows from Proposition 3.2 that the family π'_n accepts $L_{\exists x \Phi}$. A similar construction gives a family of programs over $\diamond_q(M, K)$ that accepts $L_{\exists_q^r x \Phi}$. (In this case the condition on Y is that the sum of the coefficients of the triples (m_1, m_2, m_3), with $m_1 m_2 m_3$ an accepting value for π_n, be equal to r modulo q.) It follows from Proposition 3.1 that these monoids have the required structure. This proves the Proposition.∎

U_1 denotes the monoid $\{0, 1\}$ with the usual multiplication: $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0, 1 \cdot 1 = 1$.

Propostion 4.2. Let $L \subseteq A^*$ be a regular language. If $L \in mACC(q)$ then every group in $M(L)$ is solvable, and for $t \geq 0$, every group in $\eta_L(A^t)$ has order dividing a power of q. Furthermore, if $L \in mCC(q)$, then $\eta_L(A^+)$ contains no monoid isomorphic to U_1.

Proof. If $M(L)$ contains a nonsolvable group, then by a result in [BCST], there is some $t > 0$ such that $\eta_L(A^t)$ contains a nonsolvable group. Thus it suffices to consider a nontrivial group $G \subseteq \eta_L(A^t)$. Suppose $L \in mACC(q)$. It follows from Proposition 4.1 that L is accepted by a family $\{\pi_n\}$ of single-scan programs over some $M \in \mathbf{M}_q$. Let $C = \eta_L^{-1}(G) \cap A^t$. C can be viewed as a finite alphabet, and the restriction of η_L to C^* gives a homomorphism ψ from C^* onto G. Let e be the identity of G, and $g \in G \backslash \{e\}$. It follows from the definition of the syntactic monoid that there exist $x, y \in A^*$ such that if $\eta_L(v) = e$ and $\eta_L(w) = g$, then $xuy \in L$ and $xvy \notin L$, or vice-versa. By suitably modifying the first and last instructions of the program π_n one obtains a family of programs θ_n that accepts the language $\{u : xuy \in L\}$. If n is a multiple of t, then the composition of θ_n with the embedding of C into A^t gives a family of single-scan programs $\{\rho_m\}$ over M with input alphabet C. It follows from Theorem 2.2 of [BS] that if $G \notin \mathbf{G}_q$, then there exist strings $v, w \in C^m$ such that $\psi(v) = e, \psi(w) = g$, but ρ_m emits the same value on v and w. It thus follows that there exist $v', w' \in A^{mt}$ such that $\eta_L(v') = e, \eta_L(w') = g$, and $xv'y, xw'y$ are either both in L, or both in $A^* \backslash L$, a contradiction. So $G \in \mathbf{G}_q$. It remains to consider what happens when $L \in mCC(q)$. In this case, $M \in \mathbf{G}_q$. Observe that if $\eta_L(A^+)$ contains a copy
of U_1, then for some $t > 0$, $\eta_L(A^t)$ contains a copy of U_1. (Take $v, w \in A^+$ mapping onto U_1 and consider $v^{|w|}$ and $w^{|v|}$.) The proof now proceeds exactly as above, making the same appeal to the results of [BS]. ∎

The theorem now follows from arguments given in [BCST] and [CDPC]. In Theorem 7 of [BCST] it is shown that if L is a regular language containing no nonsolvable groups, and such that every group in $\eta_L(A^t)$ has order dividing a power of q, then

$$L \in (\{\exists\} \cup MOD_{\{q\}})[\mathcal{P}].$$

In the proof of Theorem 6.3.1 of [CDPC] it is shown that if, additionally, $\eta_L(A^+)$ contains no copy of U_1, then

$$L \in MOD_{\{q\}}[\mathcal{P}].$$

5 Additional Remarks

The results here imply that $mACC$ (the union of the $mACC(q)$ over all $q > 0$) is properly contained in NC^1, and that mCC (the union of the $mCC(q)$) does not contain the AND function. In fact, these corollaries follow directly from Proposition 4.1 and the results of [BS]; the additional work done in Proposition 4.2 to characterize all the regular languages in these classes is not needed.

It should be noted that *equality,* rather than just inclusion, holds for the first part of Theorem 2.1, for the simple reason that the predicates $x \equiv 0 \pmod r$ are themselves monadic. Since ordinary first-order quantification over $x \equiv 0$ and $x < y$ is required to form the numerical predicates in $\mathcal{P}$, it is not clear whether equality holds for the second part as well. A more interesting question is whether the quantifier complexity is preserved in passing from the arbitrary monadic sentence to the sentence with numerical predicates in $\mathcal{P}$. There are strong indications that this is true, but I have not verified this.

Of course, the principal open question is whether one can establish these results for numerical predicates of arbitrary arity. Even if this can be done for dyadic numerical predicates the result will be quite significant: The natural uniform versions of the classes $ACC(q)$ and $CC(q)$ are defined using the single numerical predicate $BIT(x,y)$, which is interpreted to mean the x^{th} bit of the binary representation of y is on [BIS]. Thus an extension of Theorem 2.1 to the dyadic case would resolve most of the open questions about the relations among AC^0, CC and ACC in the uniform setting. My suspicion is that the monadic case is very special, and that any extension beyond this will encounter all of the difficulties present in the most general nonuniform setting.

References

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

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

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

[BCST] D. Barrington, K. Compton, H. Straubing, and D. Thérien, Regular languages in NC^1, to appear in *J. Comp. Syst. Sci.*

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

[BS] D. Barrington and H. Straubing, Superlinear lower bounds for bounded-width branching programs, in *Proc. 6th IEEE Structure in Complexity Theory Conference* (1991) 305-314; to appear in *J. Comp. Syst. Sci.*

[BT] D. Barrington and D. Thérien, Finite monoids and the fine structure of NC^1, *JACM* **35**, (1988), 941-952.

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

[E] S. Eilenberg, *Automata, Languages and Machines, vol. B,* Academic Press, New York (1976).

[F] R. Fagin, Generalized first-order spectral and polynomial-time recognizable sets, *SIAM-AMS Proceedings,* vol. 7, American Mathematical Society, Providence (1974).

[FSS] M. Furst, J. Saxe and M. Sipser, Parity, circuits, and the polynomial time hierarchy, *J. Math Systems Theory* **17**, (1984), 13-27.

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

[MP] R. McNaughton and S. Papert, *Counter-free Automata,* MIT Press, Cambridge, Massachusetts (1971).

[P] J. E. Pin, *Varieties of Formal Languages,* Plenum, New York (1986).

[STT] 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).

[W] P. Weil, Products of languages with counter, *Theoretical Computer Science,* **76** (1990).