Languages Defined With Modular Counting Quantifiers

Howard Straubing

Computer Science Department
Boston College
Chestnut Hill, MA 02167
Tel.: (617)-552-3977
e-mail:straubin@cs.bc.edu

Abstract. We prove that a regular language defined by a boolean combination of generalized Σ_1-sentences built using modular counting quantifiers can be defined by a boolean combination of Σ_1-sentences in which only regular numerical predicates appear. The same statement, with "Σ_1" replaced by "first-order" is equivalent to the conjecture that the non-uniform circuit complexity class ACC is strictly contained in NC^1. The argument introduces some new techniques, based on a combination of semigroup theory and Ramsey theory, which may shed some light on the general case.
A preliminary version of this paper appeared in the Proceedings of the 1998 STACS conference.

1 Background

1.1 Lower bounds questions for small-depth circuit families

This paper was motivated by some open problems about the computational power of families of boolean circuits. As it turns out, we will not mention circuits at all after this introductory section. Nonetheless, our main result represents a positive contribution toward the resolution of these problems.

For the moment, we define a circuit with n inputs to be a directed acyclic graph with $2n$ source nodes (labeled $0, 1, x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}$) and a single sink node, with all the nodes other than the sources labeled AND or OR. The *size* of the circuit is the number of nodes, and the *depth* of the circuit is the length of the longest path from a source to the sink. A circuit $\mathcal{C}$ computes a function $f_{\mathcal{C}} : \{0,1\}^n \to \{0,1\}$ as follows. Given $a_1 \cdots a_n \in \{0,1\}^n$, we assign the value a_i to the node x_i, and $1 - a_i$ to the node $\overline{x_i}$. Each node other than a source node is assigned the conjunction or disjunction of the values of its predecessor nodes, depending on whether the node is labeled AND or OR. Since there are no cycles in the graph, this procedure assigns a well-defined boolean value to every node in the circuit; $f_{\mathcal{C}}(a_1 \cdots a_n)$ is the value assigned to the sink node.

Ordinarily we talk about the behavior of *families* of circuits, consisting of one circuit for each input length n. A family $\{\mathcal{C}_n\}_{n\geq 0}$ of circuits recognizes the language

$$\{w \in \{0,1\}^* : f_{\mathcal{C}_{|w|}}(w) = 1\}.$$

AC^0 is the name given to the class of languages recognized by families of circuits whose depth is bounded above by a constant and whose size is bounded above by a polynomial in n.

What can we do in AC^0? We can compare two numbers in binary–that is, AC^0 contains the set of strings $a_{n-1}\cdots a_0 b_{n-1}\cdots b_0$ such that the integer whose binary representation is $a_{n-1}\cdots a_0$ is less than the integer whose binary representation is $b_{n-1}\cdots b_0$. If we allow the circuits to have more than one output, we can add two numbers in binary with a polynomial-size family of depth 3 circuits. We can recognize aperiodic regular languages (see subsection 1.4); in particular, we can determine whether the number of 1's in the input string is at least k, where k is a constant. We can even determine whether the number of 1's in the input is at least $\log n$. (This last fact is far from obvious—see Fagin, *et. al.* [9].)

Furst, Saxe and Sipser [10] and, independently, Ajtai [1] showed that if $k > 1$ then the regular language

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

is not in AC^0. It follows from this that one cannot perform binary multiplication in AC^0, or determine whether the majority of the input bits are on.

What happens if we build the ability to recognize MOD_k directly into our circuits? That is, we will allow nodes to be labeled MOD_k; the value assigned to such a node is 1 if and only if the sum of the values assigned to the predecessor nodes is divisible by k. There are several outstanding open problems concerning the power of circuits that contain such "modular gates". Let $CC^0(k)$ denote the class of languages recognized by constant-depth polynomial-size families of circuits, all of whose nodes are labeled MOD_k.

Conjecture 1. *If p is a prime that does not divide k, then $MOD_p \notin CC^0(k)$.*

Let $CC^0 = \bigcup_{k>0} CC^0(k)$. (The class CC^0 has been called "pure-ACC" elsewhere in the literature.) Let AND denote the language $1^* \subseteq \{0,1\}^*$. The following conjecture is a kind of dual to the theorem of Furst-Saxe-Sipser and Ajtai cited above.

Conjecture 2. *$AND \notin CC^0$.*

We denote by $ACC(k)$ the class of languages recognized by constant-depth polynomial-size families of circuits in which nodes may be labeled AND, OR, or MOD_k. We further set $ACC = \bigcup_{k>0} ACC(k)$. The following is a strengthened form of Conjecture 1:

Conjecture 3. *If p is a prime that does not divide k, then $MOD_p \notin ACC(k)$.*

Conjecture 3 implies, among other things, that we cannot determine in ACC whther the majority of the bits in an input string are equal to 1. (If we could, then we could do this in $ACC(k)$ for some particular k; but [10] shows that we can recognize any MOD_p by a constant-depth circuit built from AND, OR, and $MAJORITY$ gates, and thus we would have $MOD_p \in ACC(k)$ for all p, contradicting Conjecture 3.)

All these conjectures are known to hold when k is itself prime or a prime power. The proof of Conjecture 3 in the prime-power case is due to Smolensky [21]. Smolensky's methods do not work when k has two distinct prime factors, and almost nothing is known about the status of the three conjectures in this case. There are some results for Conjectures 1 and 2 for circuits with MOD_m gates on the input level, and a MOD_p gate at the outputs, where p is prime. (See Barrington, Thérien and Straubing [4], Barrington and Straubing [5], Krause and Pudlak [15], Grolmusz and Tardos [12].)

1.2 Connections with logic

For a full account of the results cited in this subsection and the next one, see Straubing [22].

We will use formulas of first-order logic to define properties of strings over a finite alphabet A. The variables in these formulas denote positions in the string (that is, integers in the range between 1 and the length of the string, inclusive). There are two kinds of atomic formulas: First, for each $a \in A$, there is a unary 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 called a *numerical predicate*—the truth of $\nu(x_1, \ldots, x_n)$, where ν is a numerical predicate, depends only on the values of the positions $x_1, \ldots, x_n$ and the length of the string, and not on the letters in those positions. For example, $x < y$, $x \equiv 1 \pmod 2$, $x + y = \text{length}$, and $\text{length} \equiv 0 \pmod 3$ are numerical predicates of arity 2,1,2 and 0, respectively.

First-order formulas are built from atomic formulas in the usual way by applying boolean operations and the existential quantifier. (The universal quantifier is obtained from the existential quantifier by negation.) A formula without free variables is called a *sentence*. A sentence ϕ defines a property of strings over A, and the set of all strings over A that satisfy this property is called the *language defined by* ϕ.

For example, let $A = \{a, b\}$. Then the sentence

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

defines the regular language $A^*aA^*bA^*$. The sentence

$$\exists x (x + x = \text{length})$$

defines the set of strings of even length. The sentence

$$\exists x ((x + x = \text{length}) \wedge \forall y (y \leq x \leftrightarrow Q_a y))$$

defines the nonregular language $\{a^n b^n : n > 0\}$.

Immerman [14] and Gurevich and Lewis [13] showed that if $A = \{0, 1\}$, then the class of languages defined by first-order sentences is precisely the circuit complexity class AC^0 introduced in 1.1.

We can give similar logical characterizations of $CC^0(q)$ and $ACC(q)$ by introducing a new kind of quantifier. If $s \geq 0$, $p \geq 1$, we define an equivalence relation on the set $\mathbf{N}$ of nonnegative integers by setting

$$m \equiv n \pmod{(s,p)}$$

if and only if either $m = n$, or $m, n \geq s$ and $m \equiv n \pmod{p}$. Every nontrivial equivalence relation on $\mathbf{N}$ that is compatible with addition has this form. We call s the *stem* of the equivalence relation, and p the *period*. Observe that if $s = 0$, then this is just the usual congruence modulo p.

The new quantifiers are denoted $\exists^{(i,s,p)}$, where $s \geq 0$, $p > 0$, and $0 \leq i < s+p$. We interpret

$$\exists^{(i,s,p)} x\phi(x)$$

to mean that the number of positions x for which $\phi(x)$ holds is equivalent to i modulo (s,p). Observe that $\exists^{(1,1,1)}$ is the ordinary existential quantifier, and $\exists^{(0,1,1)}$ is the negated existential quantifier.

If $q > 2$, then $CC^0(q)$ is the class of languages defined by sentences that use only the quantifiers $\exists^{(i,0,q)}$, and if $q > 1$, then $ACC(q)$ is the class of languages defined by sentences that use only quantifiers of the form $\exists^{(i,s,q)}$.

Let us make a couple of remarks concerning this last statement. Observe that

$$\exists x\phi$$

is equivalent to

$$\bigvee_{i=1}^{q} \exists^{(i,1,q)} x\phi.$$

If $i < s$, then

$$\exists^{(i,s,q)} x\phi$$

is equivalent to

$$\exists^{=i} x\phi(x),$$

where $\exists^{=i}$ means "there exist exactly i", while if $i \geq s$, then

$$\exists^{(i,s,q)} x\phi$$

is equivalent to

$$\exists^{\geq s} x\phi(x) \wedge \exists^{(i,0,q)} x\phi(x),$$

where $\exists^{\geq s}$ means "there exist at least s". It is easy to express both $\exists^{=i}$ and $\exists^{\geq s}$ in terms of the ordinary existential quantifier (as long as we have equality in our language). Thus, we could just as well say that $ACC(q)$ consists of all languages defined by sentences using only ordinary existential quantifiers and generalized quantifiers of the form $\exists^{(i,0,q)}$, which is in fact the form in which this theorem is stated in [22]. Furthermore, this argument shows that it suffices to work with the quantifiers $\exists^{(i,0,q)}$ and $\exists^{(i,1,q)}$, as all the others can be expressed in terms of these two. The second remark is that the circuit complexity class $CC^0(2)$ is somewhat anomalous, since circuits that contain MOD_2 gates alone have very restricted computing power. In fact, the languages definable by sentences that use only the quantifiers $\exists^{(i,0,2)}$ are precisely those in $CC^0(4)$, which is identical to $CC^0(2^k)$ for all $k > 1$.

The theorem of Furst, Saxe and Sipser and Ajtai cited in 1.1 thus asserts that the languages MOD_q for $q > 1$ are not definable by first-order sentences. As MOD_q is a regular language, it is natural to pose the question of which regular languages are first-order definable. The answer is given by a result of Barrington, Compton, Straubing and Thérien [2]: Let us say that a numerical predicate is *regular* if and only if it is equivalent to a first-order formula over the atoms

$$x < y$$

and

$$x \equiv i \pmod{q}.$$

For example, the 0-ary numerical predicate

$$\text{length} \equiv 0 \pmod{2}$$

is equivalent to

$$\exists x \forall y (y \leq x \wedge x \equiv 0 \pmod{2}),$$

and is consequently a regular numerical predicate. The regular numerical predicates form the largest class of numerical predicates such that every sentence which uses only the numerical predicates of this class defines a regular language. It is proved in [2] that a regular language is first-order definable (and hence in AC^0) if and only if it is defined by a first-order sentence in which only regular numerical predicates are used. There is a particularly compelling notation in which to express this result. Let **FO** denote the family of languages definable by first-order sentences (so that in the case of a binary alphabet $\mathbf{FO} = AC^0$) and let $\mathbf{FO}[\mathbf{Reg}]$ denote the family of languages defined by first-order sentences that use only regular numerical predicates. Let **Reg** denote the family of all regular languages. Then

$$\mathbf{FO} \cap \mathbf{Reg} = \mathbf{FO}[\mathbf{Reg}].$$

This result, which holds over any finite alphabet, is proved by appeal to the circuit lower bounds of Ajtai and Furst, Saxe and Sipser, although the statement does not itself refer to circuits, only to the definability of regular languages in first-order logic. It would be of more than passing interest to find a direct proof of the above equality, since this would give an alternative proof of the circuit lower bounds.

We conjecture that the analogous equalities hold for classes of languages defined by sentences containing the generalized quantifiers. This is equivalent to the conjectures from Subsection 1.1 concerning CC^0 and ACC. More precisely, let $\mathbf{Mod(s,q)}$ denote the class of languages defined by sentences that use only the quantifiers $\exists^{(i,s,q)}$ and let $\mathbf{Mod(s,q)}[\mathbf{Reg}]$ denote the the subclass defined by such sentences in which only regular numerical predicates are used. Then

Theorem 4. *Let $q > 0$. The following are equivalent:*
(1) Conjectures 1 and 2.
(2) $\mathbf{Mod(0,q)} \cap \mathbf{Reg} = \mathbf{Mod(0,q)}[\mathbf{Reg}]$.

Theorem 5. *Let $q, s > 0$. The following are equivalent:*
(1) Conjecture 3.
(2) $\mathbf{Mod(s,q)} \cap \mathbf{Reg} = \mathbf{Mod(s,q)}[\mathbf{Reg}]$.

1.3 Statement of the main result

One might try to prove an equality of the form

$$\mathbf{FO} \cap \mathbf{Reg} = \mathbf{FO}[\mathbf{Reg}]$$

by induction on the quantifier complexity of sentences, beginning with the Σ_1-sentences. Indeed, if we denote by $\boldsymbol{\Sigma_1}$ the family of languages defined by Σ_1-sentences, and by $\boldsymbol{\Sigma_1}[\mathbf{Reg}]$ the family of languages defined by Σ_1-sentences that use only regular numerical predicates, then

$$\Sigma_1 \cap \mathbf{Reg} = \boldsymbol{\Sigma_1}[\mathbf{Reg}].$$

The rather simple proof (see [22]) directly transforms a Σ_1-sentence for L into an equivalent sentence which, if L is regular, contains only regular numerical predicates. The analogous identity is true, by complementation, for Π_1-sentences, but to extend this even to general boolean combinations of Σ_1-sentences is already a very difficult problem.

We generalize the notion of Σ_1-sentences to our new quantifiers as follows: We can write formulas in which we quantify over k-tuples of positions rather than over individual positions. Thus

$$\exists^{(i,s,p)}(x_1,\ldots,x_k)\phi(x_1,\ldots,x_k)$$

means 'the number of k-tuples of positions satisfying ϕ is congruent to i modulo (s,p)'. If ϕ is quantifier-free, then we call such a formula a *generalized Σ_1-formula.* Observe that while for ordinary Σ_1-formulas,

$$\exists(x_1,\ldots,x_k)\phi$$

is equivalent to

$$\exists x_1 \exists x_2 \cdots \exists x_k \phi,$$

the precise analogue is not true for the generalized quantifiers. Nonetheless, generalized quantification over k-tuples can be expressed in terms of generalized quantification over individuals. For example, in the case $k = 2$,

$$\exists^{(i,s,p)}(x_1,x_2)\phi$$

is equivalent to

$$\bigvee \bigwedge_{j=1}^{s+p-1} \exists^{(j,s,p)}x_1 \exists^{(i(j),s,p)}x_2\phi,$$

where the disjunction is over all sequences $(i(1),\ldots,i(s+p-1))$ such that

$$i \equiv \sum_{j=1}^{s+p-1} i(j)\cdot j \pmod{(s,p)}.$$

Thus, quantification over k-tuples does not introduce any new operations.

Let us denote by $\Sigma_1^{(s,p)}$ the class of languages defined by generalized Σ_1-sentences of modulus (s,p), and by $\mathbf{B\Sigma_1^{(s,p)}}$ the class of languages defined by boolean combinations of such sentences. As usual, we use the suffix $[\mathbf{Reg}]$ to indicate the restriction to regular numerical predicates. Here is our main theorem.

Theorem 6. *For all $s \geq 0, p > 0$,*

$$\mathbf{B\Sigma_1^{(s,p)}} \cap \mathbf{Reg} = \mathbf{B\Sigma_1^{(s,p)}}[\mathbf{Reg}].$$

For the case of $(1,1)$-quantification, this answers the question raised above about the boolean combinations of ordinary Σ_1-sentences; this case was proved independently by Maciel, Péladeau and Thérien [17] using quite different techniques. A theorem very close to the $s=0$ case of Theorem 6 appears in [4], using an argument that has some points in common with the proof we give here. Our proof relies on a combination of semigroup theory and Ramsey-style combinatorics. The important connection to semigroup theory is discussed in the next subsection.

1.4 Connections to algebra

We remarked above that if we had a direct proof of the equality

$$\mathbf{FO} \cap \mathbf{Reg} = \mathbf{FO}[\mathbf{Reg}]$$

then we could prove the circuit lower bounds of [1] and [10] directly. This is because we can give a precise characterization of the languages in $\mathbf{FO}[\mathbf{Reg}]$ in terms of semigroup-theoretic invariants of regular languages. Similarly, we can precisely characterize the classes $\mathbf{Mod(s,q)}[\mathbf{Reg}]$ in semigroup-theoretic terms, and thus we possess an effective means for determining whether a given regular language belongs to any of these classes. (See Barrington, Compton, Straubing and Thérien [2] and Straubing [22]). In particular, we can show that $MOD_p \notin \mathbf{MOD(s,q)}[\mathbf{Reg}]$ if p is a prime that does not divide q, and that $AND \notin \mathbf{MOD(0,q)}[\mathbf{Reg}]$.

The use of finite semigroups in circuit complexity originates in the work of Barrington [3] and Barrington and Thérien [6], who introduced a model of computation called a *program over a finite monoid.* For purposes of the present paper we will give a somewhat different definition of these programs, due to Péladeau, Straubing and Thérien [18] and indicate their connection with the Barrington-Thérien model.

Let A be a finite alphabet, and let $k,n>0$. If $w = a_1\cdots a_n \in A^n$ and $I=(i_1,\ldots,i_k)\in\{1,\ldots,n\}^k$, then $w(I)$ denotes the k-tuple $(a_{i_1},\ldots,a_{i_k})\in A^k$.

A *k-program* over a finite monoid M associates to each $I \in \{1, \ldots, n\}^k$ a map $f_I : A^k \to M$. The *value* of the program on $w \in A^n$ is

$$\prod_{I \in \{1,\ldots,n\}^k} f_I(w(I)) \in M,$$

where the k-tuples are ordered lexicographically. (In our application in this paper, M will be commutative, so the ordering is irrelevant.) Ordinarily, we consider families of k-programs over M, consisting of one program over M for each input length n. The family of programs thus computes a map $F : A^* \to M$. (For $n = 0$ we always take the program's value to be the identity of M.)

We can also view k-programs as language recognizers. Let $X \subseteq M$. A family of programs over M, with X as the set of accepting values, accepts a string $w \in A^*$ if and only if $F(w) \in X$. Barrington and Thérien give the following definition: Let M be a finite monoid, A a finite alphabet, and $n > 0$. A program over M consists of a sequence of *instructions*

$$(i_1, f_1), \ldots, (i_r, f_r),$$

where each i_j is in $\{1, \ldots, n\}$ and each f_j is a map from A to M. The value of the program on $w = a_1 \cdots a_n \in A^n$ is

$$\prod_{j=1}^{r} f_j(a_{i_j}).$$

In the present paper, we shall call such a program a *BT-program*. Barrington and Thérien characterized various circuit complexity classes in terms of polynomial-length families of BT-programs. While k-programs are not precisely the same thing, any polynomial-length family of BT-programs over M can be simulated by a family of k-programs over M, for some k. Conversely, any family of k-programs over M can be simulated by a polynomial-length family of BT-programs over M', where M' is strucutrally very close to M. (See [18].) This enables us to reformulate the algebraic characterizations of circuit complexity classes in terms of the k-program model. We state these in the next theorem, without proof, as they are not required for the proof of Theorem 6.

In the statement of the theorem below, we use the following terminology: A finite monoid M is *aperiodic* if it contains no nontrivial groups, and *solvable* if all the groups it contains are solvable groups. NC^1 denotes the family of languages recognized by families of circuits built of two-input AND and OR gates, whose depth is bounded by $O(\log n)$.

Theorem 7. *Let $L \subseteq \{0, 1\}^*$.*
(a) $L \in AC^0$ if and only if L is recognized by a family of k-programs over a finite aperiodic monoid.
(b) $L \in CC^0(q)$ if and only if L is recognized by a family of k-programs over a finite solvable group whose cardinality divides a power of q.

(c) $L \in ACC(q)$ if and only if L is recognized by a family of k-programs over a finite solvable monoid in which the cardinality of every group divides a power of q.
(d) $L \in NC^1$ if and only if L is recognized by a family of k-programs over a finite monoid.

We prove our main theorem by first establishing a rather technical lemma about the computational power of k-programs over finite *commutative* monoids. This will be carried out in the next section. In Section 3 we will translate this lemma into our model-theoretic Theorem 6. Section 4 presents some other consequences of the lemma and discusses the prospects for an eventual resolution of the Conjectures 1, 2 and 3.

2 Programs over Finite Commutative Monoids

2.1 A congruence on A^+.

A *congruence* on an algebraic structure is an equivalence relation that is compatible with the operations on the structure. The set $\mathbf{N}$ of nonnegative integers with the operations of addition and multiplication forms a commutative semiring (*i.e.,* a system with operations of addition and multiplication that satisfy the commutative, associative and distributive laws), and the relation of congruence modulo (s, p) defined above is a congruence on this semiring. In fact, all congruences of finite index on the semiring $\mathbf{N}$ have this form. We write $\mathbf{N}_{s,p}$ to denote the quotient of $\mathbf{N}$ by this congruence. $\mathbf{N}_{s,p}$ is itself a commutative semiring, and any commutative monoid M (a monoid is an algebraic structure with an associative operation and an identity element for the operation), with its operation written additively, that satisfies the identity

$$(s + p) \cdot x = s \cdot x$$

for all $x \in M$ is a semimodule over this semiring. ($s \cdot x$ means the sum of s copies of x.) As usual, if A is a finite alphabet, then we denote by A^* the set of all strings over A, and by A^+ the set of all nonempty strings over A. A^* is a monoid, and A^+ a semigroup, with concatenation of strings as the operation.

Let $w \in A^*$, and $v = a_1 \cdots a_k$, with each $a_i \in A$. We say v is a *subword* of w if

$$w \in A^* a_1 A^* \cdots a_k A^*.$$

(This somewhat unusual terminology is from Eilenberg [8]. When the letters of v occur consecutively in w, so that $w \in A^* v A^*$, we say v is a *factor* of w.) An *occurrence* of v as a subword of w is a factorization

$$w = w_0 a_1 w_1 \cdots a_k w_k,$$

with each $w_i \in A^*$. The *signature* of this occurrence is the bit string

$$\sigma = b_0 1 b_1 \cdots 1 b_k,$$

where b_i is the empty string whenever w_i is the empty string, and $b_i = 0$ whenever $|w_i| > 0$.

Example. Consider the boldfaced occurrence of the subword aba in the word *aa***ab***bb***a**. The signature is 01101. In *aa***a***b***b***aa***a***aa*, the signature is 0101010.

Let $w \in A^+$, $v \in A^+$, $|v| \leq k$, and let σ be a bit string with $|v|$ 1's and without two consecutive occurrences of 0. We denote by $c(w, v, \sigma)$ the number of occurrences of v as a subword of w with signature σ.

We define $w_1 \theta^k_{s,p} w_2$ if for all v with $|v| \leq k$, and all signatures σ,

$$c(w_1, v, \sigma) \equiv c(w_2, v, \sigma) \pmod{(s, p)}.$$

This is obviously an equivalence relation on A^+ of finite index. It is also a congruence; to see this, let $a \in A$, $w \in A^*$. If v occurs as a subword of aw with signature σ, and $\sigma = 1\tau$, then $v = av'$, for some v', and $c(aw, v, \sigma) = c(w, v', \tau)$. If, on the other hand, $\sigma = 01\tau$, then $c(aw, v, \sigma) = c(w, v, 1\tau) + c(w, v, \sigma)$. In either case, we have $w_1 \theta^k_{s,p} w_2$ implies $aw_1 \theta^k_{s,p} aw_2$. Similarly, $w_1 \theta^k_{s,p} w_2$ implies $w_1 a \theta^k_{s,p} w_2 a$. Thus if $w_1 \theta^k_{s,p} w_2$ and $v_1 \theta^k_{s,p} v_2$, then $w_1 v_1 \theta^k_{s,p} w_2 v_2$, so $\theta^k_{s,p}$ is a congruence.

We will use the symbol $\theta^k_{s,p}$ to denote not only the congruence, but the homomorphism from A^+ onto the quotient semigroup by this congruence.

2.2 The computing power of k-programs over commutative monoids

In this subsection we state a technical lemma about the computing power of k-programs over finite commutative monoids. This is, in effect, an algebraic formulation of our main theorem. Let S be a finite semigroup, and let $\phi : A^+ \to S$ be a surjective homomorphism. We say that a family of k-programs over a finite monoid M *simulates* ϕ if whenever $w_1, w_2 \in A^*$ have the same length, and the program for inputs of this length has the same value on w_1 and w_2, then $\phi(w_1) = \phi(w_2)$. In effect, we can use the program to determine the product of a sequence of elements of S.

Let M be a finite commutative monoid. We will write the operation in M additively. Since M is finite, there exist $s \geq 0, p > 0$ such that for all $x \in M$, $(s + p) \cdot x = s \cdot x$. As noted in Subsection 2.1, this makes M a semimodule over $\mathbf{N}_{s,p}$.

It is easy to see that we can simulate the natural homomorphism from A^+ onto $A^+/\theta^k_{s,p}$ by a family of k-programs over such an M : We take one copy of $\mathbf{N}_{s,p}$ for each pair (v, σ) where v is a string of length no more than k, and σ is a signature with $|v|$ 1's, and we set M to be the direct sum of these copies. For each subsequence of $\{1, \ldots, n\}$ of length no more than k we arbitrarily choose a single k-tuple from $\{1, \ldots, n\}$ whose elements are exactly the elements of the subsequence. Given a k-tuple I of positions in a word of length n, the program map f_I gives the identity of M if I is not one of the chosen representatives. If I is one of the chosen representatives, then it represents a subsequence with some signature σ. If v is the subword occupying this sequence of positions, then the program map f_I emits 1 in the component corresponding to (v, σ), and 0 in the other components.

Let $t \geq 0, q > 0$. Define a homomorphism $\lambda_{t,q} : A^* \to \mathbf{N}_{t,q}$ by mapping each word w to $|w| \bmod (t, q)$. Observe that for any finite monoid M with at least $t + q$ elments, a family of k-programs over M can trivially simulate the homomorphism $\lambda_{t,q}$.

We claim that, subject to certain conditions, the only homomorphisms that M can simulate with k-programs are, in effect, combinations of the two homomorphisms above.

Here is a precise statement of this claim: Suppose that the homomorphism $\phi : A^+ \to S$ maps the alphabet A itself onto S (*i.e.*, $\phi(A) = S$) and let us also suppose that $S = S^2 = \{st : s, t \in S\}$. Suppose that M, as above, is a commutative monoid satisfying the identity $(s + p) \cdot x = s \cdot x$. Let $T_r = (\phi \times \theta^k_{s,p})(A^r)$ (so that T_r is a subset of $S \times A^+/\theta^k_{s,p}$). There exist $t \geq 0, q > 0$ such that $T_t = T_{t+q}$, because of the finiteness of $S \times A^+/\theta^k_{s,p}$. It follows that whenever $r_1 \equiv r_2 \pmod{(t, q)}$, then $T_{r_1} = T_{r_2}$.

Theorem 8. *Let S be a finite semigroup and let $\phi : A^+ \to S$ be a homomorphism simulated by a family of k-programs over a commutative monoid M that satisfies the identity $(s + p) \cdot x = s \cdot x$. Suppose $\phi(A) = S$ and $S^2 = S$, and let r, q be as above. Then ϕ is refined by $\theta^k_{s,p} \times \lambda_{t,q}$.*

We will give the proof of this theorem in the next three subsections.

2.3 Step 1: Setting the stage

To prove Theorem 8, we need to show that if

$$(\theta^k_{s,p} \times \lambda_{t,q})(v) = (\theta^k_{s,p} \times \lambda_{t,q})(w),$$

then

$$\phi(v) = \phi(w).$$

Since $|w| \equiv |v| \pmod{(t, q)}$, we have $T_{|v|} = T_{|w|}$, and thus there exists v' such that $|v| = |v'|$, and

$$(\phi \times \theta^k_{s,p})(v') = (\phi \times \theta^k_{s,p})(w).$$

In particular, $v \theta^k_{s,p} v'$. We need to show $\phi(v) = \phi(v')$, from which it will follow that $\phi(v) = \phi(w)$.

Let us write

$$v = a_1 \cdots a_m,$$

and

$$v' = a'_1 \cdots a'_m.$$

Our hypothesis about ϕ and S implies that for each $a \in A$ and each $c > 0$ there exists a string $\alpha_{a,c}$ of length c such that $\phi(a) = \phi(\alpha_{a,c})$. We will show how to construct strings

$$V = \alpha_{a_1,k_1} \cdots \alpha_{a_m,k_m},$$

and

$$V' = \alpha_{a'_1,k_1} \cdots \alpha_{a'_m,k_m},$$

such that the program for words of length $|V| = |V'|$ has the same value on V and V'. This implies $\phi(V) = \phi(V')$, by hypothesis, and thus $\phi(v) = \phi(v')$.

2.4 Step 2: Families of intervals, signatures, and colorings

Let $m = |v| = |v'|$ as in the preceding subsection. We will choose n much larger than m (exactly how large will be specified later). An *interval* in $\{1, \ldots, n\}$ is a subset K of $\{1, \ldots n\}$ such that whenever $x < y < z$ are integers with $x, z \in K$, then $y \in K$. We will use the usual notation of open and half-open intervals to denote these—for example $[a, b) = \{x : a \leq x < b\}$—but with the understanding that we mean the set of *integers* x such that $a \leq x < b$.

A subset $U = \{p_1 < \cdots < p_r\}$ of $\{2, \ldots, n\}$ partitions $\{1, \ldots, n\}$ into a set of $r + 1$ disjoint non-empty intervals

$$[p_0 = 1, p_1), [p_1, p_2), \ldots, [p_r, p_{r+1} = n + 1).$$

We denote a subset $\mathcal{F}$ of of this set of intervals by its *signature* $\sigma(\mathcal{F})$; this is a string of 0's and 1's of length $r + 1$ that has 1 in the i^{th} position if and only if $[p_i, p_{i+1})$ is in $\mathcal{F}$. We are only interested in subsets $\mathcal{F}$ in which every one of the elements of U is included as an endpoint of at least one of the intervals in $\mathcal{F}$—this is equivalent to requiring that $\sigma(\mathcal{F})$ not contain two consecutive 0's, exactly like the signatures of occurrences of subwords introduced in 2.1. We call such a family $\mathcal{F}$ an *admissible* set of intervals.

Thus each set $U \subseteq \{2, \ldots, n\}$ with $|U| = r$ gives rise to a bijection $\mathcal{I}_U$ between bit strings of length $r + 1$ without two consecutive 0's and admissible (with respect to U) families of disjoint intervals of $\{1, \ldots, n\}$. Note that if we are given a family $\mathcal{F}$ of intervals in $\{1, \ldots, n\}$, and the value of n, we can deduce the unique set U that makes $\mathcal{F}$ admissible, and we can then compute $\sigma(\mathcal{F})$. Note also that for a given signature τ, $\sigma(I_U(\tau)) = \tau$.

Example. Let $U = \{2, 4, 7\}$, $n = 8$. Then $\mathcal{I}_U(0101) = \{[2, 4), [7, 9)\}$ and $\mathcal{I}_U(1101) = \{[1, 2), [2, 4), [7, 9)\}$.

Example. Again with $n = 8$, let $\mathcal{F} = \{[2, 3), [3, 4), [5, 7)\}$. For this to be admissible, we must have $U = \{2, 3, 4, 5, 7\}$, and thus $\sigma(\mathcal{F}) = 011010$.

A k-tuple $I \in \{1, \ldots, n\}^k$ is said to be *compatible* with a family $\mathcal{F}$ of intervals if every component of I belongs to an interval in $\mathcal{F}$, and every interval in $\mathcal{F}$ contains a component of I.

Example. If $\mathcal{F} = \{[1, 4), [4, 7)\}$ and $k = 5$, then $(2, 4, 2, 6)$ is compatible with $\mathcal{F}$, but $(1, 2, 3, 2)$ and $(2, 4, 2, 7)$ are not.

Let z be the number of intervals of $\mathcal{F}$. We will define a map $\chi_{\mathcal{F}}$ from A^z into M, or from A^{z-1} into M. Which of the two domains we choose depends upon $\sigma(\mathcal{F})$. For families with certain signatures, we will also define a second map $\chi'_{\mathcal{F}} : A^z \to M$.

Case 1. Suppose $\sigma(\mathcal{F})$ begins with 11. This means that for some $p > t > 1$, $\mathcal{F}$ contains both the intervals $[1, t)$ and $[t, p)$. In this case the domain of $\chi_{\mathcal{F}}$ is A^{z-1}. Given $b_1 \cdots b_{z-1} \in A^{z-1}$, we define a string w of length n as follows: We place $\alpha_{b_1,p-1}$ in the first $p-1$ positions of w. For $i > 1$, we place $\alpha_{b_i,d}$ in the positions corresponding to the $(i+1)^{th}$ interval of $\mathcal{F}$, where d is the length of the interval. The positions of w that do not belong to the intervals of $\mathcal{F}$ can be filled arbitrarily. We then set

$$\chi_{\mathcal{F}}(b_1, \ldots, b_{z-1}) = \sum f_I(w(I)),$$

where the f_I are the program maps for inputs of length n, and where the summation is over all intervals I that are compatible with $\mathcal{F}$. Because of the compatibility, the arbitrarily filled positions of w do not enter into this summation, and so $\chi_{\mathcal{F}}$ is well-defined in this case.

Case 2. Suppose $\sigma(\mathcal{F})$ begins with 10. This means that $\mathcal{F}$ contains an interval $[1, t)$, but no interval whose left-hand endpoint is t. Let p be the left-hand endpoint of the second interval of $\mathcal{F}$. Now the domain of $\chi_{\mathcal{F}}$ is A^z. Given $b_1 \cdots b_z \in A^z$, we define a word w of length n by filling the first $t-1$ positions with the prefix of length $t-1$ of $\alpha_{b_1,p-1}$. For $i > 1$ we place $\alpha_{b_i,d}$ in the positions corresponding to the i^{th} interval of $\mathcal{F}$. The remaining positions of w are filled arbitrarily. We then set

$$\chi_{\mathcal{F}}(b_1, \ldots, b_z) = \sum f_I(w(I)),$$

where again the summation ranges over all k-tuples compatible with $\chi_{\mathcal{F}}$.

Case 3. The remaining case is where $\sigma(\mathcal{F})$ begins with 01, since a signature cannot begin with 00. This means that $\mathcal{F}$ contains no interval whose left endpoint is 1. Let t be the smallest endpoint in $\mathcal{F}$, so that $t > 1$ and $\mathcal{F}$ contains the interval $[t, p)$. Let $b_1 \cdots b_z \in A^z$. We now define two words w and w' of length n. To construct w we fill the positions corresponding to the i^{th} interval of $\mathcal{F}$ with $\alpha_{b_i,d}$, where d is the length of this interval. To construct w', we do the same thing, except we treat the first interval $[t, p)$ differently—we fill these positions with the suffix of length $p-t$ of $\alpha_{b_1,p-1}$. In both words we fill the other positions arbitrarily, and we set

$$\chi_{\mathcal{F}}(b_1, \ldots, b_z) = \sum f_I(w(I)),$$

$$\chi'_{\mathcal{F}}(b_1, \ldots, b_z) = \sum f_I(w'(I)),$$

where the summations range over all k-tuples compatible with $\mathcal{F}$.

Thus our partition set U has given rise to a map Δ_U on signatures of length $|U|+1$. If σ begins with 1, then Δ_U maps σ to $\chi_{\mathcal{F}}$, where $\mathcal{F} = \mathcal{I}_U(\sigma)$; if σ begins with 0, then Δ_U maps σ to the pair $(\chi_{\mathcal{F}}, \chi'_{\mathcal{F}})$, where again $\mathcal{F} = I_U(\sigma)$. Δ_U is called the *color* of the set U. Let $|U| = r$. Then the number of maps from A^z into M is no more than $\gamma(r) = |M|^{|A|^{r+1}}$, and so the number of posible colors is bounded above by $\delta(r) = \gamma(r)^{2|A|^r}$.

2.5 Step 3: Application of Ramsey's Theorem

Let us recall Ramsey's Theorem (see [11]): Let $m, k, c > 0$, with $k \leq m$. Then there exists $n = n(m, k, c)$ such that if all the k-element subsets of $\{1, \ldots, n\}$ are colored from a set of c colors, then there exists an m-element subset T of $\{1, \ldots, n\}$ such that all the k-element subsets of T have the same color.

Here is an extension of Ramsey's Theorem: Let $m, k, c_1, \ldots, c_k > 0$, with $k \leq m$. Then there exists $n = n(m, k, c_1, \ldots, c_k)$ with the following property: If all the i-element subsets of $\{1, \ldots, n\}$, with $1 \leq i \leq k$, are colored from a set of c_i colors, then there exists an m-element subset T of $\{1, \ldots, n\}$ such that for each $1 \leq i \leq k$ there exists a color κ_i such that all the i-element subsets of T are colored κ_i. This extension follows from the original Ramsey theorem by an easy induction on k.

We now apply this extended Ramsey theorem to the coloring defined at the end of the last subsection. We are given m, the length of the words v and v'. For any given n, r we have a coloring of the r-subsets of $\{2, \ldots, n\}$ from a set of $\delta(r)$ colors. We now choose n so large that there exists an m-element subset J of $\{2, \ldots, n\}$ such that for $1 \leq j \leq k$, all the j-element subsets of J have the same color Δ_j.

The set $J = \{i_1, \ldots, i_m\}$ thus partitions $\{1, \ldots, n\}$ into the set of intervals

$$\mathcal{J} = \{[1, i_1), \ldots, [i_m, n+1)\}.$$

We now construct the words V, V' described at the end of 2.3. Recall

$$v = a_1 \cdots a_m$$

and

$$v' = a'_1 \cdots a'_m.$$

V is formed by placing $\alpha_{a_1, i_2 - 1}$ in the first $i_2 - 1$ positions, and, for $2 \leq j \leq m$, $\alpha_{a_j, i_{j+1} - i_j}$ in positions $i_j, \ldots, i_{j+1} - 1$. (We make the convention that $i_{m+1} = n + 1$.) V' is constructed in the identical fashion, using $\alpha_{a'_j, t}$ in place of $\alpha_{a_j, t}$.

The value of the program on V is

$$\sum_I f_I(V(I)),$$

where the summation is over all k-tuples I from $\{1,\ldots,n\}$. As each k-tuple is compatible with exactly one subset of the set of intervals $\mathcal{J}$, we can rewrite this sum as

$$\sum_{\mathcal{F}}\sum_{I} f_I(V(I)),$$

where the outer summation is over all the subfamilies $\mathcal{F}$ of $\mathcal{J}$ having k or fewer members, and where the inner summation is over all k-tuples I compatible with $\mathcal{F}$. Now, given a family $\mathcal{F}$ of these intervals, let $T_{\mathcal{F}} = \{i_{j_1} < \cdots < i_{j_r}\}$ be the set of left endpoints of $\mathcal{F}$, if $\mathcal{F}$ does not contain the interval $[1, i_1)$. If $\mathcal{F}$ contains the interval $[1, i_1)$ then we define $T_{\mathcal{F}}$ to contain i_1 as well as the left endpoints of the other intervals of $\mathcal{F}$. We set $J_{\mathcal{F}} = (j_1, \ldots, j_r)$, that is, the sequence of subscripts of $T_{\mathcal{F}}$. Observe that $r \le k$.

Now look at $\chi_{\mathcal{F}}(v(J_{\mathcal{F}}))$. If $J_{\mathcal{F}}$ does not contain 1, so that $\mathcal{F}$ contains neither $[1, i_1)$ nor $[i_1, i_2)$, then this is just the sum of the $f_I(V(I))$ over all I compatible with $\mathcal{F}$. The same is true if $\mathcal{F}$ contains $[1, i_1)$. The anomalous case is when $\mathcal{F}$ contains $[i_1, i_2)$ but not $[1, i_1)$: In this case, the sum of the $f_I(V(I))$ over the I compatible with $\mathcal{F}$ only involves the suffix of α_{a_1, i_2-1} of length $i_2 - i_1$, whereas to evaluate $\chi_{\mathcal{F}}(v(J_{\mathcal{F}}))$ we would place α_{a_1, i_2-i_1} in the interval $[i_1, i_2)$. Here is where we need the alternative function $\chi'_{\mathcal{F}}$. Let $\mathcal{U}$ denote the set of all subfamilies of $\mathcal{J}$ having k or fewer members, and that contain $[1, i_1)$ or do not contain $[i_1, i_2)$, and let $\mathcal{U}'$ denote the remaining subfamilies of $\mathcal{J}$ with k or fewer members. We conclude that the value of the program on V is

$$\sum_{\mathcal{F}\in\mathcal{U}} \chi_{\mathcal{F}}(v(J_{\mathcal{F}})) + \sum_{\mathcal{F}\in\mathcal{U}'} \chi'_{\mathcal{F}}(v(J_{\mathcal{F}})).$$

Each term in this sum represents an occurrence of $v(J_{\mathcal{F}})$ as a subword of v. Let σ be the signature of this occurrence. If σ does not begin with 1, then there is only one family $\mathcal{F}$ that gives rise to this occurrence, but if σ begins with 1, then there are three such families.

First, if σ begins with 0, then $1 \notin J_{\mathcal{F}}$, so $\mathcal{F}$ contains neither $[1, i_1)$ nor $[i_1, i_2)$, and $\sigma(\mathcal{F}) = \sigma$. Thus the underlying partition set has cardinality $|\sigma| - 1$, and so, by the monochromaticity, this occurrence contributes

$$[\varDelta_{|\sigma|-1}(\sigma)]^{(1)}(v(J_{\mathcal{F}}))$$

to the sum; the superscript (1) indicates that we are applying the first component of $\varDelta_{|\sigma|-1}(\sigma)$, since in this case $\varDelta_{|\sigma|-1}(\sigma)$ is a pair of maps. The families $\mathcal{F}$ of this form thus contribute

$$\sum_{|u|\le k} c(v,u,\sigma)\cdot[\varDelta_{|\sigma|-1}(\sigma)]^{(1)}(u)$$

to the sum. (Observe that if u has no occurrences with the signature σ, then $c(v, u, \sigma) = 0$, so we can include this term in the sum.)

If σ begins with 1, then $\mathcal{F}$ contains either $[1, i_1)$, $[i_1, i_2)$, or both. Suppose first that $\mathcal{F}$ contains $[1, i_1)$ but not $[i_1, i_2)$. Then $\sigma(\mathcal{F})$ is obtained from σ by changing the initial 1 to 10, and then coalescing adjacent 0's, if there are any. (For example, if $\mathcal{F} = \{[1, i_1), [i_2, i_3), [i_4, i_5)\}$ with $m > 5$, then $J_{\mathcal{F}} = \{1, 2, 4\}$, $\sigma = 11010$, and $\sigma(\mathcal{F}) = 101010$. On the other hand, if $\mathcal{F} = \{[1, i_1), [i_3, i_4), [i_4, i_5)\}$, then $J_{\mathcal{F}} = \{1, 3, 4\}, \sigma = 10110$, and $\sigma(\mathcal{F}) = 10110$.) Let us denote by $[\sigma]$ the signature obtained from σ in this way. The families $\mathcal{F}$ of this form contribute

$$\sum_{|u| \leq k} c(v, u, \sigma) \cdot \Delta_{|[\sigma]|-1}([\sigma)])(u)$$

to the sum.

Suppose now that σ begins with 1, and that $\mathcal{F}$ contains $[i_1, i_2)$ but not $[1, i_1)$. Then $\sigma(\mathcal{F}) = 0\sigma$. The families $\mathcal{F}$ of this form contribute

$$\sum_{|u| \leq k} c(v, u, \sigma) \cdot [\Delta_{|\sigma|}(0\sigma)]^{(2)}(u)$$

to the sum.

Finally, if $\mathcal{F}$ contains both $[1, i_1)$ and $[i_1, i_2)$, then $\sigma(\mathcal{F}) = 1\sigma$, and so these families contribute

$$\sum_{|u| \leq k} c(v, u, \sigma) \cdot \Delta_{|\sigma|}(1\sigma))(u)$$

The result of this analysis is that the value of the program on $\mathcal{V}$ is the sum over all words u of length no more than k of

$$\sum_{\sigma \in 0(0+1)^*} c(v, u, \sigma) \cdot [\Delta_{|\sigma|-1}(\sigma)]^{(1)}(u),$$

and

$$\sum_{\sigma \in 1(0+1)^*} c(v, u, \sigma) \cdot P_{\sigma,u},$$

where

$$P_{\sigma,u} = \Delta_{|[\sigma]|-1}([\sigma)])(u) + [\Delta_{|\sigma|}(0\sigma)]^{(2)}(u) + \Delta_{|\sigma|}(1\sigma))(u).$$

Now, the value of the program on V' is precisely the same expression, except that v is replaced everywhere by v'. But by assumption, $c(v, u, \sigma) \equiv c(v', u, \sigma)$ (mod (s, p)) for all u of length no more than k, and all signatures σ, and thus the two values are identical. This completes the proof.

3 Proof of Theorem 6

To prove Theorem 6, we will first show that if a language is defined by a boolean combination of generalized Σ_1-sentences, then it is recognized by a family of k-programs over a finite commutative monoid. We then show that if a regular language L is recognized by a family of k-programs over a finite commutative monoid M, then we can simulate multiplication in the syntactic monoid of L by a family of k-programs over a direct product of copies of M. We then apply Theorem 8 to a subsemigroup of the syntactic monoid of L—this will enable us to define the classes of the syntactic congruence, and hence L itself, as boolean combinations of generalized Σ_1-sentences with regular numerical predicates.

Lemma 9. *Let $s \geq 0, p > 0$. $L \subseteq A^*$ is defined by a boolean combination of generalized Σ_1-sentences of modulus (s,p) if and only if L is recognized by a family of k-programs over a finite commutative monoid M (written additively) that satisfies the identity $(s+p) \cdot x = s \cdot x$.*

Proof. We first prove the 'only if' direction; in fact this is the only direction we need for the proof of Theorem 6. Suppose first that L is defined by a single generalized Σ_1-sentence of modulus (s,p). If $\mathbf{a} = (a_1, \ldots, a_k) \in A^k$, then we abbreviate the formula $\bigwedge_{i=1}^{k} Q_{a_i} x_i$ by $Q_{\mathbf{a}}\mathbf{x}$. We can rewrite the quantifier-free part of the sentence in such a manner that the sentence has the form

$$\exists^{(i,s,p)}\mathbf{x} \bigvee_{\mathbf{a} \in A^k} (Q_{\mathbf{a}}\mathbf{x} \wedge R_{\mathbf{a}}\mathbf{x}),$$

where each $R_{\mathbf{a}}\mathbf{x}$ is a numerical predicate. Let M be the additive monoid of $\mathbf{N}_{s,p}$. If $I \in \{1, \ldots, n\}^k$, then we define $f_I(\mathbf{a})$ to be 1 if $R_{\mathbf{a}}(I)$ holds in strings of length n, and 0 otherwise. (Recall that $R_{\mathbf{a}}$ depends only on the components of I and the length of the string.) We take $\{i\}$ to be the set of accepting values. Obviously this family of k-programs recognizes L. If L is a boolean combination of generalized Σ_1-sentences, then it is recognized by a family of k-programs over a direct product of copies of $\mathbf{N}_{s,p}$.

For the converse, suppose L is recognized by a family of k-programs over a finite commutative monoid M that satisfies the identity $(s+p) \cdot x = s \cdot x$. As usual, we denote the program maps by f_I. Given a string $w \in A^*$, and $m \in M$, let $c_m(w) \in \mathbf{N}_{s,p}$ denote the number, modulo (s,p), of k-tuples I such that $f_I(w) = m$. L is then a finite union of finite intersections of languages of the form

$$L_{c,m} = \{w \in A^* : c_m(w) = c\},$$

where $c \in \mathbf{N}_{s,p}$. (The union is over all $\mathbf{N}_{s,p}$-linear combinations of the elements of M whose values lie in the accepting set of the family of k-programs, and each intersection is over all the elements m of M.) Thus it suffices to find a generalized Σ_1-sentence for each $L_{c,m}$. For each $\mathbf{a} \in A^k$ and $I \in \{1, \ldots, n\}^k$, let $S_{\mathbf{a}}(I)$ be true if and only if $f_I(\mathbf{a}) = m$. S depends only on the components of I and the length of the input string, and is hence a numerical predicate. $L_{c,m}$ is then defined by the sentence

$$\exists^{(i,s,p)}\mathbf{x} \bigvee_{\mathbf{a} \in A^k} (Q_{\mathbf{a}}\mathbf{x} \wedge S_{\mathbf{a}}\mathbf{x}),$$

and thus L is defined by a boolean combination of such sentences. This completes the proof.

If $L \subseteq A^*$ and $w \in A^*$, then the quotient languages $w^{-1}L$ and Lw^{-1} are defined by

$$w^{-1}L = \{v \in A^* : wv \in L\},$$

and

$$Lw^{-1} = \{v \in A^* : vw \in L\}.$$

Lemma 10. *Let $L \subseteq A^*$, $w \in A^*$. If L is recognized by a family of k-programs over a finite commutative monoid M, then $w^{-1}L$ and Lw^{-1} are recognized by families of k-programs over M.*

Proof. If a language K is recognized by a family of k-programs over a monoid M, then the language K^ρ obtained by reversing all the strings in K is also recognized by a family of k-programs over M—we obtain the new program maps by re-indexing the old program maps. Thus we need only prove the lemma for Lw^{-1}, because $w^{-1}L = ((L^\rho)w^{-1})^\rho$. Further, we may assume that w consists of a single letter a, since if $|w| > 1$, we obtain the quotient Lw^{-1} by repeated application of this operator. Let $\{g_I\}$ denote the collection of program maps recognizing elements of L of length $n+1$. To each $I \in \{1, \ldots, n+1\}^{k+1}$ we associate a k-tuple $r(I) \in \{1, \ldots, n\}^k$ as follows: If $n+1$ is not a component of I, then $r(I) = I$. If $n+1$ is a component of I, then we choose for $r(I)$ an arbitrary element of $\{1, \ldots, n\}^k$ whose set of components is identical to the set of components of I that are less than $n+1$. For each $J \in \{1, \ldots, n\}^k$, we set $f_J(v(J))$ to be the sum, over all I such that $r(I) = J$, of $g_J((va)(J))$. Clearly, $f_J(v(J))$ depends only on the sequence $v(J)$ and not on the other letters of v, so this is well-defined. The f_J, together with the same set of accepting values used for the g_I, give a family of k-programs over M that recognize La^{-1}.

Note that the proof of the preceding lemma used the commutativity of M in a crucial way. The lemma is true for noncommutative monoids if we increase the size of the tuples used in the new program from k to $k + |w|$.

We now proceed to the proof of Theorem 6. Let $L \subseteq A^*$ be a *regular* language defined by a boolean combination of generalized Σ_1-sentences of modulus (s, p). By Lemma 9, L is recognized by a family of k-programs over a finite commutative monoid M that satisfies an identity of the form $(s+p) \cdot x = s \cdot x$.

Let us recall here the definition of the *syntactic monoid* of L, denoted $M(L)$. (See Pin [19] or Eilenberg [8].) Two words w_1 and w_2 in $A^*$are said to be congruent with respect to L if and only if for all $u, v \in A^*$, $uw_1v \in L$ if and only if $uw_2v \in L$. It is easy to see that this is a congruence on A^*, and that the index of the congruence is finite if and only if L is regular. $M(L)$ is the monoid of congruence classes, and the map that takes a word to its congruence class is called the *syntactic morphism* of L, denoted μ_L. Observe that each congruence class is a boolean combination of sets of the form $u^{-1}Lv^{-1}$, and that if L is regular, there are only finitely many sets of this form. It follows from Lemma 10 that each congruence class, *i.e.,* each set of the form

$$L_m = \{w \in A^* : \mu_L(w) = m\}$$

is recognized by a family of k-programs over a direct product of copies of M.

Now consider the sets $P_t = \mu(A^t)$, for $t > 0$. These form a finite semigroup of subsets of $\mu_L(A^t)$, and thus there is an idempotent element $P_r = P_{2r} = P_r^2$. That is, $S = P_r$ is a subsemigroup of $M(L)$ such that $S^2 = S$. Let $B = A^r$, considered as a finite alphabet. We obtain a homomorphism $\nu : B^+ \to S$ simply by restricting μ_L to the strings over A whose lengths are divisible by r. Each of the sets

$$K_s = \{w \in B^+ : \nu(w) = s\},$$

where $s \in S$, is recognized by a family of k-programs (with B as the input alphabet) over a direct product of copies of M: This is because each k-tuple of positions in a word over B corresponds to r^k k-tuples of positions in the corresponding word over A, thus we can set each map in the new k-program to be the sum, over all these r^k k-tuples, of the original program maps. If we now form the direct product of these $|S|$ programs, we obtain a family of programs that simulates ν. We are thus in a position to apply Theorem 8: ν is refined by $\theta^k_{s,p} \times \lambda_{t,q}$. (Keep in mind that the underlying alphabet for this congruence is B, not A.)

What we have to show now is that each K_s is defined by a boolean combination of generalized Σ_1-sentences of modulus (s,p) with regular numerical predicates, and then argue that the same is true for each class L_m of μ_L. Since L is a union of μ_L-classes, this will complete the proof.

Each set of the form

$$\{w \in B^* : \lambda_{t,q}(w) = i\}$$

is defined by a 0-ary regular numerical predicate

$$\exists x \forall y (y \leq x \wedge x \equiv i \pmod{t,q}).$$

Let us see why this is a regular numerical predicate: If $i < t$, then $x \equiv i \pmod{(t,q)}$ is equivalent to $x = i$. We define $x = i$ by a formula that says, 'there exist $i-1$ distinct positions to the left of x, but there do not exist i distinct positions to the left of x'; for example, $x = 3$ is equivalent to

$$\exists y_1 \exists y_2 (y_1 < y_2 \wedge y_2 < x) \wedge \neg \exists y_1 \exists y_2 \exists y_3 (y_1 < y_2 \wedge y_2 < y_3 \wedge y_3 < x).$$

If, on the other hand, $i \geq t$, then $x \equiv i \pmod{(t,q)}$ is equivalent to

$$(x \geq t) \wedge (x \equiv i \pmod{q}),$$

and $x \geq t$ is expressed by a formula that says that there are t distinct positions to the left of x. Thus each set in question is defined by a first-order sentence whose atoms are of the form $x < y$ and $x \equiv i \pmod{q}$, which is, by definition, a 0-ary regular numerical predicate.

Each of the sets

$$\{w \in B^* : c(w,u,\sigma) \equiv \dot{p}mod(s,p)\},$$

where $u \in B^*$ and σ is a signature, is defined by a generalized Σ_1-sentence of modulus (s,p) with regular numerical predicates. This sentence asserts that there are $i \bmod (s,p)$ $|u|$-tuples of positions that contain the letters of u and that have gaps in the appropriate places. For example, the set

$$\{w \in B^* : c(w, aba, 01101) \equiv i \pmod{(s,p)}\}$$

is defined by the sentence

$$\exists^{i,s,p}(x_1,x_2,x_3)\phi,$$

where ϕ is

$$Q_a x_1 \wedge Q_b x_2 \wedge Q_a x_3 \wedge (x_1 > 1) \wedge (x_2 = x_1 + 1) \wedge (x_3 > x_2 + 1) \wedge (x_3 = \text{length}).$$

Each of the numerical predicates appearing in the above formula can be expressed in terms of $<$, and thus is a regular numerical predicate. (For example, $x_1 > 1$ is expressed by a formula that says there exists a position to the left of x_1, and $x_2 = x_1 + 1$ is expressed by a formula that says $x_1 < x_2$ and there exists no position to the right of x_1 and to the left of x_2.) Each congruence class of $\theta^k_{s,p} \times \lambda_{t,q}$ is a boolean combination of sets of the form

$$\{w \in B^* : \lambda_{t,q}(w) = i\}$$

and

$$\{w \in B^* : c(w,u,\sigma) \equiv i \pmod{(s,p)}\},$$

and is thus defined by a boolean combination of generalized Σ_1-sentences of modulus (s,p) with regular numerical predicates, and each K_s is a finite union of these languages.

We now translate this back into the alphabet A. Consider first the languages

$$L^{(r)}_s = \{w \in (A^r)^* : \mu_L(w) = s\}.$$

We write a sentence for $L^{(r)}_s$ by taking each subsentence of the defining sentence for K_s of the form

$$\exists^{(i,s,p)}(x_1,\ldots,x_k)\phi$$

and replacing it by

$$\exists^{(i,s,p)}(x_{1,1},\ldots,x_{1,r},\ldots,x_{k,1},\ldots,x_{k,r})\phi',$$

where ϕ' is obtained from ϕ in the following steps. First, we replace each atomic formula $Q_b x_i$ by

$$\bigwedge_{j=1}^{r} Q_{a_j} x_{i,j},$$

where $b \in B$ is the element $a_1 \cdots a_r$ of A^r. Second, we replace each occurrence of $x_i < x_j$ by $x_{i,r} < x_{j,r}$, each occurrence of $x_i \equiv j \pmod q$ by $x_{i,r} \equiv jr \pmod{qr}$, and every quantifier $\exists x_i$ by

$$\exists x_{i,1} \cdots \exists x_{i,r}.$$

Third, we take the conjunction of this transformed formula with

$$\bigwedge_{i=1}^{k}((x_{i,r} \equiv 0 \pmod r)) \wedge \bigwedge_{j=1}^{r-1}(x_{i,j} + 1 = x_{i,j+1})) \wedge \text{length} \equiv 0 \pmod r).$$

This gives us a defining sentence of the required type for $L_s^{(r)}$.

Finally, we show how to write a sentence of the required type for L_m. Each L_m is the union, over all $w \in A^*$ of length less than r, of the languages $L_s^{(r)}w$, where $s \cdot \mu_L(w) = m$. We already know that each $L_s^{(r)}$ is defined by a sentence of the required type, so it remains to show that if a language N is so defined, then so is Nw. It is sufficient to prove this in the case $w = a \in A$. To do this, we take the defining sentence for N, and, working from the innermost quantifers outward, replace each subformula $\exists x\psi$ by

$$\exists x(\exists y(x < y) \wedge \psi)$$

and each

$$\exists^{(i,s,p)}(x_1, \ldots, x_k)\psi$$

by

$$\exists^{(i,s,p)}(x_1, \ldots, x_k)(\exists y(x_1 < y \wedge \cdots \wedge x_k < y) \wedge \psi).$$

We take the conjunction of the resulting sentence with a sentence that says that the last letter of the word is a. This is

$$\exists^{(1,s,p)}x(\forall y(y \leq x) \wedge Q_a x).$$

(Observe that the last sentence says that the number of final letters equal to a is congruent to 1 modulo (s,p), but there is only one final letter, so the number of final letters equal to a is always 0 or 1.) This gives the required sentence, and completes the proof of Theorem 6.

4 Consequences of the Main Theorem

4.1 Elementary results on expressibility

An immediate consequence of our main theorem is the result of Maciel, Péladeau and Thérien [17], for ordinary Σ_1-sentences:

Theorem 11. *If a regular language L is defined by a boolean combination of Σ_1-sentences, then it is defined by a boolean combination of Σ_1-sentences that use only regular numerical predicates.*

Proof. This is just the case $s = p = 1$ of Theorem 6.

Theorem 12. *The language $1^* \subseteq \{0,1\}^*$ cannot be defined by a boolean combination of generalized Σ_1-sentences of modulus $(0,p)$.*

Proof. Suppose otherwise. This language is regular, so by Theorem 6, it is defined by a boolean combination of sentences with regular numerical predicates and quantifiers of the form $\exists^{(j,0,q)}$, where q is the least common multiple of the moduli in the original sentences. Let U_1 denote the monoid $\{0,1\}$ with the usual multiplication. By Theorem VII.4.2 of [22], the image of $\{0,1\}^+$ under the syntactic morphism of the language does not contain a copy of U_1. But the syntactic morphism of 1^* maps $\{0,1\}$ onto U_1, a contradiction.

As we shall see in the next subsection, the foregoing theorem is equivalent to a result of Barrington, Straubing and Thérien [4] on the power of BT-programs over finite nilpotent groups. Observe, by the way, that if $s > 0$, then we can define 1^* by the sentence $\exists^{(0,s,p)} x Q_0 x$.

Theorem 13. *Let q be prime. The language MOD_q cannot be defined by a boolean combination of generalized Σ_1-sentences of modulus (s,p) unless p is divisible by q.*

Proof. Suppose otherwise. Since MOD_q is regular, Theorem 6 implies that it is defined by a boolean combination of sentences with regular numerical predicates of modulus (s,p), with p not divisible by q. The syntactic morphism μ of MOD_q maps 0 to the identity and 1 to the generator of the cyclic group of order q, and thus maps the set of strings of length q onto the group of order q. But by Theorem VII.4.1 of [22], any group in $\mu(\{0,1\}^q)$ has cardinality dividing a product of the moduli occurring in a defining sentence, a contradiction.

4.2 $\mathcal{J}$-trivial monoids

Let M be a finite monoid, and let $x \in M$. The *two-sided ideal* of M generated by x is the set $\{mxm' : m, m' \in M\}$. Two elements x and y of M are said to be $\mathcal{J}$-equivalent if they generate the same two-sided ideal. M is said to be $\mathcal{J}$-trivial if distinct elements always generate distinct two-sided ideals. (As a simple example, every monoid that is both idempotent—*i.e.,* every element is idempotent—and commutative is $\mathcal{J}$-trivial, since if $x = mym'$ and $y = nxn'$, then $x = mnxn'm'$ and $y = nmnxn'm'n' = mn^2x(n')^2m' = mnxn'm' = x$.)

Let A be a finite alphabet, and let $r > 0$. If $w_1, w_2 \in A^*$, then we define $w_1 \alpha_r w_2$ if and only if the set of subwords of w_1 of length no more than r is the same as the set of subwords of w_2 of length no more than r. It is easy to verify that α_r is a congruence on A^* of finite index. It follows that the quotient A^*/α_r is a finite monoid. Observe that we can identify each element of this monoid with a set of words over A of length no more than r, and that if m, x, m' are elements of this monoid, then $x \subseteq mxm'$. It follows readily that A^*/α_r is $\mathcal{J}$-trivial. The following theorem, due to I. Simon [20], says, in effect, that every $\mathcal{J}$-trivial monoid is obtained in this way.

Theorem 14. *Let M be a finite $\mathcal{J}$-trivial monoid, and let $\phi : A^* \to M$ be a homomnorphism. Then there exists $r > 0$ such that ϕ factors through the projection from A^* onto A^*/α_r.*

We will use Simon's theorem to prove the following lemma.

Lemma 15. *If a language L is recognized by a family of k-programs over a finite $\mathcal{J}$-trivial monoid M, then there exists k' such that L is recognized by a family of k'-programs over a finite idempotent and commutative monoid.*

Proof. Let $L \subseteq A^*$ be recognized by a family of k-programs over a finite $\mathcal{J}$-trivial monoid M. It follows from Theorem 14 that there exists $r > 0$ such that L is recognized by a family of k-programs over M^*/α_r, where we consider M as a finite alpahbet. In fact, the program maps are the same; we just interpret $f_I(w(I)) \in M$ as an element of the larger monoid M^*/α_r. The value of the program on $w \in A^*$ is thus determined by the set of sequences of the form

$$(f_{I_1}(w(I_1)), \ldots, f_{I_s}(w(I_s))),$$

where $s \leq r$ and $I_1 < \cdots < I_r$ in lexicographic order.

Let $M^{(j)}$ denote the Cartesian product of j copies of M, and let N be the set of subsets of $M \cup M^{(2)} \cup \cdots M^{(r)}$, with union as the operation. This makes N into a finite idempotent and commutative monoid. We define (kr)-program maps over M as follows: If the (kr)-tuple I is formed by concatenating r k-tuples $I_1, \ldots, I_r$ in lexicographic order, then we define $g_I(a_1, \ldots, a_{rk})$ to be the set of all subsequences of

$$(f_{I_1}(a_1, \ldots, a_k), f_{I_2}(a_{k+1}, \ldots, a_{2k}), \ldots, f_{I_r}(a_{(r-1)k+1}, \ldots, a_{rk}).$$

Otherwise, we set $g_I((a_1, \ldots, a_{rk})$ to be the empty set (which is the identity of the monoid N). The resulting family of (rk)-programs recognizes N.

We now ask, when can one finite monoid be simulated by a family of k-programs over another finite monoid? That is, let M and N be finite monoids, and consider a family of k-programs over N, where the input alphabet $A_M = \{a_m : m \in M\}$ is in one-to-one correspondence with M. We say that the family of k-programs *simulates* M if whenever the program gives the same value for two input sequences

$$a_{m_1} \cdots a_{m_n}$$

and

$$a_{m'_1} \cdots a_{m'_n}$$

of the same length, then

$$m_1 \cdots m_n = m'_1 \cdots m'_n.$$

Our results above imply that any finite $\mathcal{J}$-trivial monoid can be simulated by a family of k-programs over a finite idempotent and commutative monoid. The results of Furst, Saxe and Sipser cited earlier, along with Theorem 7(a), imply that if a finite monoid M is simulated by a family of k-programs over a finite aperiodic monoid, then M itself must be aperiodic. Similarly, Conjectures 2 and 3 are equivalent (via Theorem 7(b,c)) to asserting that if M is simulated by a family of k-programs over a finite solvable group (resp. solvable monoid), then M is itself a solvable group (solvable monoid).

A family of finite monoids that is closed under finite direct products, submonoids and quotient monoids is called a *pseudovariety* of finite monoids. We call a pseudovariety $\mathbf{V}$of finite monoids a *program variety* if whenever M is simulated by a family of k-programs over a monoid $N \in \mathbf{V}$, then $M \in \mathbf{V}$. Thus the discussion above shows that the pseudovariety of finite aperiodic monoids is a program variety. The question of whether the pseudovarieties of solvable groups and solvable monoids are program varieties is, of course, our central open question. Here we show:

Theorem 16. *The psuedovariety* $\mathbf{J}$ *of finite* $\mathcal{J}$*-trivial monoids is a program variety.*

Proof. Suppose M is simulated by a family of k-programs over a finite $\mathcal{J}$-trivial monoid N. Let $A_M = \{a_m : m \in M\}$ be a finite alphabet in one-to-one correspondence with M, and let $\phi : A_M^* \to M$ be the homomorphism that maps each a_m to m. By Lemma 15, each of the sets $\{w : \phi(w) = m\}$, where $m \in M$, is recognized by a family of k-programs over a finite idempotent and commutative monoid, so M itself is simulated by a family of k-programs over a direct product N' of these monoids; note that N' is itself idempotent and commutative. Since $\phi(A_M) = M = M^2$, Theorem 8 implies that ϕ is refined by $\theta_{1,1}^k \times \lambda_{t,q}$ for some $t \geq 0, q > 0$.

Now let $w_1, w_2 \in A_M^*$, with $w_1 \alpha_k w_2$. We form new words w_1' and w_2' as follows: If

$$w_1 = b_1 \cdots b_s,$$

where each $b_i \in A_M$, then we set

$$w_1'' = a_1^{2k} b_1 a_1^{2k} \cdots b_s a_1^{2k},$$

and we define w_2'' analogously. We pad both of these words with enough copies of a_1 at the right-hand end to obtain words w_1' and w_2' with $w_1' \lambda_{t,q} w_2'$. It is now easy to see that if a word v of length no more than k occurs as a subword of w_1' with signature σ, then it occurs in w_2' with the same signature. Thus $w_1 \theta_{1,1}^k w_2$, so $\phi(w_1') = \phi(w_2')$, hence $\phi(w_1) = \phi(w_2)$ (because $\phi(a_1) = 1$). We have proved that ϕ factors through A_M^*/α_r, and hence M is $\mathcal{J}$-trivial.

Our definition of program varieties differs from the definition of p-varieties given by Péladeau, Straubing and Thérien [18]. Their definition is in terms of polynomial-size BT-programs rather than k-programs. Maciel, Péladeau and Thérien [17] prove that the pseudovariety of dot-depth one semigroups forms a p-variety, which in fact implies our theorem above.

4.3 Nilpotent Groups

We can define finite nilpotent groups either in terms of the lower and upper central series, or as direct products of p-groups. We refer the reader to any textbook on group theory. Here we shall need the following characterization of nilpotent groups, due to Thérien [24], building on work of Eilenberg [8] on p-groups. Let us define equivalence relations α_r^m on A^* by setting $w_1 \alpha_r^m w_2$, if for each word v of length no more than r, the number of occurrences of v as a subword of w_1 is congruent, modulo m, to the number of its occurrences as a subword of w_2. It is obvious that each α_r^m has finite index, and easy to verify that each of these equivalences is a congruence. Thérien proved:

Theorem 17. *Each of the quotient monoids A^*/α_r^m is a nilpotent group. Furthermore, if $\phi : A^* \to G$ is a homomorphism into a nilpotent group, then there exist $m, r > 0$ such that ϕ factors through the projection from A^* onto A^*/α_r^m.*

We use this to prove the analogue of Lemma 15:

Lemma 18. *If a language L is recognized by a family of k programs over a finite nilpotent group, then there exists k' such that L is recognized by a family of k'-programs over a finite abelian group.*

Proof. The proof exactly parallels that of Lemma 18. Observe that the congruence α_r^m counts occurrences of subwords modulo m, rather than simply testing for the presence of subwords. Thus we redefine the monoid N to be the set of maps from $M \cup M^{(2)} \cup \cdots \cup M^{(r)}$ into $\mathbf{Z}_m$, with pointwise addition as the operation. This makes N an abelian group, and we argue as before that L is recognized by a family of (rk)-programs over N.

It is well known that the family of finite nilpotent groups forms a pseudovariety of finite monoids, which we denote $\mathbf{G}_{\text{nil}}$.

Theorem 19. $\mathbf{G}_{\text{nil}}$ *is a program variety.*

Proof. Suppose a finite monoid M is simulated by a family of k-programs over a nilpotent group G. As in the proof of Theorem 16, it follows from Lemma 18 that M is simulated by a family of k'-programs over a finite abelian group H, which we write additively. Suppose, contrary to the statement of the theorem, that M is not a nilpotent group. If M is a non-nilpotent group, then, by a result of Barrington, Straubing and Thérien, there is a family of BT-programs over M that recognizes 1^*; if M is not a group, then M contains a copy of the monoid U_1. In either case, 1^* is recognized by a family of BT-programs over M. We now compose the BT-programs with the k' programs: The BT-program takes an input sequence

$$a_1 \cdots a_n,$$

and transforms it into a sequence

$$\Psi = (f_1(a_{i_1}), \ldots, f_s(a_{i_s}))$$

of elements of M. The k'-program queries k'-tuples of this sequence from M; thus, the output of each program map depends on a k'-tuple of positions in the original input sequence $a_1 \cdots a_n$. Observe that each k'-tuple from $\{1, \ldots, n\}$ is queried many times, but we can use the commutativity of H to add up all the resulting values in H for each k'-tuple, and thus obtain a family of k'-programs over H that recognizes 1^*. It now follows from Lemma 9 that 1^* is definable by a boolean combination of generalized Σ_1-sentences of modulus $(0, p)$ for some $p > 0$, contradicting Theorem 12.

Much the same argument shows that 1^* cannot be recognized by a family of BT-programs (regardless of size) over a finite nilpotent group, a result due to Barrington, Straubing and Thérien [4].

We can further generalize the congruences $\alpha_r^{(m)}$ and α_r to allow subword counting modulo (s, p). We define the congruences $\alpha_r^{(s,p)}$ accordingly. The quotient monoids $A^*/\alpha_r^{(s,p)}$ generate the join pseudovariety $\mathbf{J} \vee \mathbf{G}_{\text{nil}}$, which is the smallest pseudovariety that contains both $\mathbf{J}$ and $\mathbf{G}_{\text{nil}}$. We believe that our main theorem implies the following statement, however we have not yet been able to prove it:

Conjecture 20. $\mathbf{J} \vee \mathbf{G}_{\text{nil}}$ *is a program variety.*

5 Conclusion

Our outstanding open problems concern k-programs over solvable groups and monoids, and our techniques apply to k-programs over commutative monoids. But solvable monoids are built from commutative monoids: Indeed, every solvable monoid divides an iterated wreath product of commutative monoids. One can also capture the solvable monoids in terms of congruences: Thérien [24] constructs a sequence of congruences $\alpha_r^{(s,p)}(k)$, $k > 0$, with $\alpha_r^{(s,p)}(1) = \alpha_r^{(s,p)}$, such that every solvable monoid divides the quotient of A^* by some $\alpha_r^{(s,p)}(k)$, and every solvable group divides the quotient of A^* by some $\alpha_r^{(0,p)}(k)$. It may be possible to approach the conjectures by induction on the length of the wreath product, or by the level k of the congruence, applying techniques like the ones we have used here at each step. As the group case seems to be simpler, the first case to study would be k-programs over $A^*/\alpha_r^{(0,p)}(2)$.

We have not discussed decidablity issues, so we conclude by mentioning an open problem that we believe can be resolved using the main results of the present paper.

Conjecture 21. *It is decidable whether a given regular language is in* $\mathbf{B}\Sigma_1{}^{(s,p)}$.

6 Bibliography

References

1. M. Ajtai, "Σ_1^1 formulae on finite structures", *Annals of Pure and Applied Logic* **24** (1983) 1–48.
2. D. Mix Barrington, K. Compton, H. Straubing, and D. Thérien, "Regular Languages in NC^1", *J. Comp. Syst. Sci.* **44** (1992) 478–499.
3. D. Mix Barrington, "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1", *J. Comp. Syst. Sci.* **38** (1989) 150–164.
4. D. Mix Barrington, H. Straubing, and D. Thérien, "Nonuniform Automata over Groups", *Information and Computation* **89** (1990) 109-132.

5. D. Mix Barrington and H. Straubing, “Superlinear Lower Bounds for Bounded-width Branching Programs”,*J. Comp. Syst. Sci.* **50** (1995) 374-381.
6. D. Mix Barrington and D. Thérien, “Finite Monoids and the Fine Structure of NC^1”, *JACM* **35** (1988) 941–952.
7. J. R. Büchi, “Weak Second-order Arithmetic and Finite Automata”, *Zeit. Math. Logik. Grund. Math.* **6** (1960) 66-92.
8. S. Eilenberg, *Automata, Languages and Machines,* vol. B,Academic Press, New York, 1976.
9. R. Fagin, M. Klawe, N. Pippenger, and L. Stockmeyer, "Bounded-depth, Polynomial-size Circuits for Symmetric Functions",*Theoret. Comput. Sci* **36** (1985) 239-250.
10. M. Furst, J. Saxe, and M. Sipser, “Parity, Circuits, and the Polynomial Time Hierarchy”, *J. Math Systems Theory* **17** (1984) 13–27.
11. R. Graham, B. Rothschild, J. Spencer, *Ramsey Theory,* John Wiley and Sons, New York, 1990.
12. V. Grolmusz and G. Tardos, “Lower Bounds for (MOD p-MOD m) Circuits”, *Proc. 39th IEEE FOCS,* 1998.
13. Y. Gurevich and H. Lewis, “A Logic for Constant-Depth Circuits”, *Information and Control,* **61** (1984) 65–74.
14. N. Immerman, “Languages That Capture Complexity Classes”, *SIAM J. Computing* **16** (1987) 760–778.
15. M. Krause and P. Pudlak, “On the Computational Power of Depth 2 Circuits with Threshold and Modular Gates”, *Proc. 26th ACM STOC,* 1994.
16. R. McNaughton and S. Papert, *Counter-Free Automata,* MIT Press, Cambridge, Massachusetts, 1971.
17. A. Maciel, P. Péladeau and D. Thérien, “Programs over Semigroups of Dot-depth One”, preprint (1996).
18. P. Péladeau, H. Straubing and D. Thérien, “Finite Semigroup Varieties Defined by Programs”, *Theor. Comp. Sci.* **180** (1997) 325-339.
19. J. E. Pin, *Varieties of Formal Languages,* Plenum, London, 1986.
20. I. Simon, "Piecewise Testable Events" in Proc. 2nd GI Conference, *Lecture Notes in Computer Science* **33** (1975) 214-222.
21. R. Smolensky, “Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity”, *Proc. 19th ACM STOC* (1987) 77–82.
22. H. Straubing, *Finite Automata, Formal Languages, and Circuit Complexity,* Birkhaüser, Boston, 1994.
23. H. Straubing, D. Thérien, and W. Thomas, “Regular Languages Defined with Generalized Quantifiers”,*Information and Computation* **118** (1995) 289-301.
24. D. Thérien, “Classification of finite monoids: the language approach”, *Theoret. Comput. Sci.* **14** (1981) 195-208.