When Can One Finite Monoid Simulate Another?

Howard Straubing

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

1 Introduction

Let M and N be finite monoids. We want to somehow use N as a computational device that reads a sequence of inputs from M and outputs the product, in M, of this sequence. We ask what relation M must have to N for this to be possible; that is, when can N simulate M?

We also ask the related question of when a language, in particular a regular language, can be recognized by a finite monoid N.

The answers to such questions, of course, depend upon what we mean, precisely, by using N as a computational device. There is a classical answer, in which N is treated as a finite automaton, which we review in Section 2. It turns out that there are a number of ways to generalize the definitions of simulation and recognition that give new, and quite interesting, answers to the questions, and lead to a number of open problems. Moreover, both the answers and problems have important connections to topics in computational complexity theory, which we will explore.

The results presented here are all drawn from the research literature on the connections between circuit complexity and algebra. However we give them a different treatment, with a new formalism. In particular, our presentation is directed toward an audience of algebraists, and we approach the subject as a collection of questions about semigroups. We bring in computational complexity both as a motivation and a proof technique. We do not give detailed proofs of all the results cited, but we have included proofs of enough of the theorems to give the reader a good idea of the available techniques.

2 The Classical Case: Finite Automata

The simplest way to use a monoid N as a computational device is to map each input letter a to an element $\phi(a)$ of N. To the input sequence $a_1 \cdots a_n$, we then associate the product

$$\phi(a_1) \cdots \phi(a_n) \in N.$$

We say that N *simulates* a finite monoid M if and only if there is such a map $\phi : M \to N$, as well as a map $\psi : N \to M$, such that

$$\psi(\phi(m_1) \cdots \phi(m_k)) = m_1 \cdots m_k,$$

for all $m_1, \ldots, m_k \in M$. (Implicit in this definition is the requirement that $\psi(1) = 1$, since we take the product of the empty sequence in any monoid to be the identity of the monoid.)

This gives us the following, almost trivial, theorem.

Theorem 1. *Let M and N be finite monoids. N simulates M if and only if M divides N (that is, M is a homomorphic image of a submonoid of N).*

Proof. If M divides N, there is a submonoid N' of N and a homomorphism $\psi : N' \to M$ that maps onto M. For each $m \in M$, define $\phi(m)$ to be any $n \in N'$ such that $\psi(n) = m$. Since ψ is a homomorphism, we have

$$\psi(\phi(m_1) \cdots \phi(m_k)) = m_1 \cdots m_k,$$

for all $m_1, \ldots, m_k \in M$. Conversely, let N simulate M, with maps ϕ and ψ, as in the definition. Let N' be the set of all products $\phi(m_1) \cdots \phi(m_i)$, with $m_1, \ldots m_i \in M$ and $i \geq 0$. N' is a submonoid of N, and it follows directly from the definition of simulation that the restriction of ψ to N' is a homomorphism onto M. Thus M divides N. □

We write $M \prec N$ to indicate that M divides N.

Thus, for example, we cannot simulate a non-commutative monoid with a commutative one, or a nontrivial finite group with a finite aperiodic monoid, or a nontrivial finite aperiodic monoid with a finite group.

In the theory of computation, computational devices are traditionally viewed in two different ways: As computers of input-output functions, as in the definition of simulation given above, or as language recognizers. Let A be a finite alphabet. We say that a finite monoid N *recognizes* a subset L of A^* if and only if there is a map $\phi : A^* \to N$ and a subset X of N such that

$$a_1 \cdots a_k \in L \Leftrightarrow \phi(a_1) \cdots \phi(a_k) \in X.$$

Of course, ϕ extends to a unique homomorphism from A^* into N, so we can just as well say $w \in L$ if and only if $\phi(w) \in X$, where ϕ also denotes this homomorphism. As is very well known, a monoid N recognizes a language L if and only if $M(L)$, the syntactic monoid of L, divides N. (See, for example, Pin [15] or Eilenberg [10].) When we use a monoid N to compute in this fashion, we are treating the elements of N as the states of a finite automaton whose transition function is given by the multiplication in N. In particular, a language is recognized by a finite monoid if and only if the language is regular.

3 1-Simulation

Suppose now that our map from input letters to monoid elements is allowed to vary as we scan an input word. That is, suppose that we are given a family $\{\phi_{i,n}\}_{1\leq i\leq n}$ of maps from a fixed input alphabet A to a finite monoid N, so that to each input string

$$w = a_1 \cdots a_n$$

we associate the monoid element

$$\Phi(w) = \phi_{1,n}(a_1) \cdots \phi_{n,n}(a_n).$$

A family of maps of this kind is called a *1-program.* Usually we will call Φ a 1-program, and refer to the maps $\{\phi_{i,n}\}$ as the program maps.

We will say that a language $L \subseteq A^*$ is *1-recognized* by N if there exists $X \subseteq N$ such that for all $w \in A^*$,

$$w \in L \Leftrightarrow \Phi(w) \in X.$$

If we take for our input alphabet a finite monoid M, then we say that M is *1-simulated* by N if and only if there is a map $\psi : N \to M$ such that for all sequences $m_1, \ldots m_k$, from M,

$$\psi(\phi_{1,k}(m_1) \cdots \phi_{k,k}(m_k)) = m_1 \cdots m_k.$$

Let us mention now a technical point in the definition of 1-recognition. We have required that the accepting set X be the same for all input lengths. Suppose that we relax this requirement, and allow different accepting sets $X_n \subseteq N$ for different input lengths. It is easy to show that if L is recognized by a 1-program Φ over N in this sense, then it is recognized in the original sense by a 1-program Φ' over a direct product of $2^{|N|} - 2$ copies of N. The accepting set for Φ' is the cartesian product

$$A_1 \times \cdots \times A_{2^{|N|}-2},$$

where the A_i are the distinct nonempty proper subsets of N. Similar remarks apply to the modification of the simulation model so as to allow the function $\psi : N \to M$ to be replaced by a separate function ψ_n for each input length n : We can always reduce this to the more restrictive model if we replace N by a direct product of copies of N. The result of these observations is that all the theorems stated below, in this and in subsequent sections, remain true whichever of the two recognition or simulation models we use.

Since we place no restriction on the family of program maps, we can 1-recognize uncountably many languages with very small monoids. However, the next theorem shows that we gain almost no additional power in simulation, and almost no additional ability to recognize *regular* languages.

Theorem 2. *Let M and N be finite monoids. If N 1-simulates M, then M divides a direct product of finitely many copies of N.*

Proof. Let $A_M = \{a_m : m \in M\}$ be a finite alphabet in one-to-one correspondence with M. Let $\nu : A_M^* \to M$ be the unique homomorphism that maps a_m to m for all $m \in M$. If N 1-simulates M, then there is a 1-program Φ over N such that for all $w, w' \in A_M^*$ with $|w| = |w'|$, $\Phi(w) = \Phi(w')$ implies $\nu(w) = \nu(w')$. Let $\{\phi_{i,n}\}$ be the family of program maps.

An *identity* of N over A_M is a pair $(w, w') \in A_M^* \times A_M^*$ such that for every homomorphism $\theta : A_M^* \to N$, $\theta(w) = \theta(w')$. We claim that for every such identity, $\nu(w) = \nu(w')$. To see that this implies the result, consider the homomorphism

$$\Theta : A_M^* \to N \times \cdots N$$

formed by taking the direct product of all homomorphisms $\theta : A_M^* \to N$ (there are only finitely many such θ). The claim about identities implies that ν factors through Θ; that is, there is a homomorphism μ from the image of Θ onto M such that $\mu \circ \Theta = \nu$. Thus $M \prec N \times \cdots \times N$.

To prove the claim concerning identities, suppose to the contrary that (w, w') is an identity of N such that $\nu(w) \neq \nu(w')$. There is an integer $p > 0$, such that for every $n \in N$, n^p is an idempotent. We form a new pair of words $(\overline{w}, \overline{w'})$ by replacing each letter a by the word aa_1^{2p-1}. Observe that $\overline{w}$ and $\overline{w'}$ are the images of w and w' under a homomorphism from A_M^* into itself, and thus $(\overline{w}, \overline{w'})$ is also an identity of N. Furthermore, we have $\nu(\overline{w}) = \nu(w) \neq \nu(w') = \nu(\overline{w'})$. If $\overline{w}$ and $\overline{w'}$ do not have the same length, suppose $\overline{w}$ is the shorter of the two strings. Since the lengths differ by a multiple of p, we have $|\overline{w}a_1^{kp}| = |\overline{w'}|$ for some $k > 0$. We also have, for any homomorphism $\theta : A_M^* \to N$, $\theta(a_1^p) = e$, for some idempotent $e \in N$, so $\theta(\overline{w}) = ne$ for some $n \in N$, and thus $\theta(\overline{w}a_1^{kp}) = \theta(\overline{w})$. We also have $\nu(\overline{w}a_1^{kp}) = \nu(\overline{w})$. Thus we could have assumed at the outset that $|w| = |w'|$, and we make this assumption now.

Let $m = |w| = |w'|$. Let $n > 0$. Color each pair (i, j) with $1 \leq i < j \leq n$ by the map $f : A_M \to N$ defined by

$$f(a) = \phi_{i,n}(a)\phi_{i+1,n}(a_1) \cdots \phi_{j-1,n}(a_1),$$

for all $a \in A_M$. It follows from Ramsey's Theorem that if n is large enough, there exists a set $\{1 \leq i_1 < \cdots < i_m \leq n\}$ such that all (i_k, i_{k+1}) have the same color f. Let $F : A_M^* \to N$ be the homomorphism that extends f. Let $w = b_1 \cdots b_m$, and $w' = b_1' \cdots b_m'$, where the b_i and the b_i' belong to A_M. We set

$$W = a_1^{i_1-1} b_1 a_1^{i_2-i_1-1} b_2 \cdots b_m^{n-i_m},$$

$$W' = a_1^{i_1-1} b_1' a_1^{i_2-i_1-1} b_2 \cdots b_m'^{n-i_m}.$$

By the monochromaticity,

$$\Phi(W) = n_1 f(b_1) \cdots f(b_m) n_2 = n_1 F(w) n_2,$$

and

$$\Phi(W') = n_1 f(b_1') \cdots f(b_m') n_2 = n_1 F(w') n_2,$$

for some $n_1, n_2 \in N$.

Since (w, w') is an identity of N, this gives $\Phi(W) = \Phi(W')$, and thus since N simulates M, $\nu(W) = \nu(W')$. But $\nu(W) = \nu(w) \neq \nu(w') = \nu(W')$, a contradiction.□

Theorem 2 implies, for example, that we cannot 1-simulate a noncommutative finite monoid with a noncommutative one, a nontrivial aperiodic monoid with a group, or a nontrivial group with an aperiodic monoid. In this sense, 1-simulation gives no more power than the classical simulation discussed in Section 2.

We can use the same ideas to characterize the *regular* languages 1-recognized by a finite monoid N. If A is a finite alphabet and $L \subseteq A^*$ is regular, then the syntactic monoid $M(L)$ is finite. Let μ_L denote the syntactic morphism of L. The set $\{\mu_L(A^k) : k > 0\}$ is a finite subsemigroup of the semigroup of subsets of $M(L)$, and thus contains an idempotent. So there exists $t > 0$ such that $\mu_L(A^t) = \mu_L(A^{2t})$, that is, $\mu_L(A^t)$ is a subsemigroup of $M(L)$. Let B be the finite alphabet whose letters are the elements of A^t, and let $\eta_L : B^* \to M(L)$ be the homomorphism defined by restricting μ_L to words over A whose length is divisible by t.

Theorem 3. *Let $L \subseteq A^*$ be a regular language 1-recognized by a finite monoid N, and let B and η_L be as above. Then every monoid in $\eta_L(B)$ divides a direct product of copies of N.*

Proof. From the definition of the syntactic morphism and the finiteness of M_L, there exist $u_1, \ldots, u_k, v_1, \ldots, v_k \in A^*$ with the following property: The value of $\mu_L(w)$ is entirely determined by the k-tuple of bits whose i^{th} bit is 1 if $u_i w v_i \in L$, and 0 otherwise. Since L is 1-recognized by N, there is a 1-program Φ over N, and a subset X of N, such that for all $w \in A^*$, $w \in L$ if and only if $\Phi(w) \in X$. As usual, let $\{\phi_{i,n}\}$ be the family of program maps. Suppose we are given a sequence

$$s_1, \ldots, s_m$$

of elements of the semigroup $\mu_L(A^t)$. Each s_i is the image under η_L of some $b_i \in B$. We will define a 1-program Ψ with program maps $\{\psi_{i,m}\}$ and input alphabet B over the monoid N^k, as follows: Let

$$u_i = x_1^{(i)} \cdots x_{|u_i|}^{(i)},$$

and

$$v_i = y_1^{(i)} \cdots y_{|v_i|}^{(i)},$$

where each $x_j^{(i)}, y_j^{(i)} \in A$. Let $b \in B$ be the word $a_1 \cdots a_t \in A^t$. We set the i^{th} component of $\psi_{1,m}(b)$ to

$$\prod_{r=1}^{|u_i|} \phi_{r,mt+|u_i|+|v_i|}(x_r^{(i)}) \cdot \prod_{r=1}^{t} \phi_{r+|u_i|,mt+|u_i|+|v_i|}(a_r).$$

We set the i^{th} component of $\psi_{m,m}(b)$ to

$$\prod_{r=1}^{t} \phi_{r+(m-1)t+|u_i|,mt+|u_i|+|v_i|}(a_r) \cdot \prod_{r=1}^{|v_i|} \phi_{r+mt+|u_i|,mt+|u_i|+|v_i|}(y_r^{(i)}).$$

If $1 < j < m$, we set the i^{th} component of $\psi_{j,m}$ to

$$\prod_{r=1}^{t} \phi_{r+(j-1)t+|u_i|,mt+|u_i|+|v_i|}(y_r^{(i}).$$

It follows that the i^{th} component of $\Psi(s_1, \ldots, s_m)$ is $\Phi(u_i w v_i)$, where w is the word obtained by writing $b_1 \cdots b_m$ in terms of the alphabet A. Given $(n_1, \ldots, n_k) \in N^k$, we map it first to the bit vector that has 1 in component i if and only if $n_i \in X$, and thence to the corresponding element of $M(L)$, if one exists. In this manner $\Psi(s_1, \ldots, s_m)$ is mapped to $s_1 \cdots s_m$. It follows that N^k simulates the semigroup $\mu_L(A^t)$, and thus every submonoid of this semigroup. The conclusion now follows from Theorem 2. □

Let $q > 1$. The language $MOD_q \subseteq \{0,1\}^*$ consists of all bit strings in which the number of 1's is divisible by q. The language $AND \subseteq \{0,1\}^*$ is $\{1^n : n \geq 0\}$.

Corollary 4. *(a) AND is not 1-recognized by any finite group.*

(b) Let $p > 1$ be prime. If MOD_p is 1-recognized by a finite monoid N then N contains a group of order p. In particular, no finite aperiodic monoid 1-recognizes MOD_p.

Proof. (a) The syntactic morphism of AND maps $\{0,1\}^q$, for all $q > 0$, onto the syntactic monoid $U_1 = \{0,1\}$ (considered as a multiplicative submonoid of the integers). By the theorem, if AND is 1-recognized by a finite monoid N, U_1 divides a direct product of copies of N, and thus N cannot be a group, since any divisor of a direct product of finite groups is itself a group.

(b) The syntactic monoid of MOD_p is the cyclic group G_p of order p, and the syntactic morphism μ maps 1 to a generator of this group, and 0 to the identity. It follows that for all $t \geq p-1$, $\mu(\{0,1\}^t) = G_p$, and thus, by the theorem, G_p divides a direct product of copies of N, which implies the result. □

The main theorems of this section were adapted from Barrington and Straubing [5].

4 k-Simulation

We may think of the program maps of a 1-program as being a sequence of instructions. Each instruction has access to one letter of the input string, and emits a monoid element according to the value of the letter that was read. In this section we enhance the computational power of our programs by allowing each instruction to have access to more than one letter of the input. As we shall see, interesting new phenomena arise. Let $k > 0$, and let A be a finite alphabet. A *k-program Φ* over a finite monoid N consists of a family of maps $\{\phi_{I,n}\}$ from A^k into N, where each I is a k-tuple over $\{1, \ldots, n\}$. If $w = a_1 \cdots a_n \in A^n$ and $I = (i_1, \ldots, i_k)$, then we denote by w_I the k-tuple $(a_{i_1}, \ldots, a_{i_k}) \in A^k$, and we set

$$\Phi(w) = \prod_{I \in \{1,\ldots,n\}^k} \phi_{I,n}(w_I),$$

where $n = |w|$, and where the product is taken in lexicographic order of the I. Observe that if $k = 1$, then this coincides with the definition of 1-programs given in the preceding section. We define k-simulation of one monoid by another, and k-recognition of a language by a finite monoid, accordingly. The remarks in the preceding section, concerning k-recognition with a single set of accepting values as opposed to different sets of accepting values for different length inputs, apply here as well.

It turns out that nonsolvable finite groups have special computing powers when we use k-programs with $k > 1$.

Theorem 5. *Let G be a finite simple non-abelian group. There exists $k > 0$ such that any finite monoid M with $|M| < |G|$ can be k-simulated by G.*

The theorem is a fairly easy consequence of the next lemma, due to Maurer and Rhodes. If $n > 0$ and S is a semigroup, then a *polynomial* over S in n variables is a word over the alphabet $S \cup \{x_1, \ldots, x_n\}$. Such a polynomial w gives rise to a function, which we also denote by w, from S^n into S, evaluated by substituting the n arguments for the respective variables and multiplying in S.

Lemma 6. *Let G be a finite simple non-abelian group, and let $n > 0$. Then every function from G^n into G is realized by a polynomial over G.*

For the proof, see Maurer and Rhodes [14] or Straubing [19].

We now prove Theorem 5. Let $\iota : M \to G$ be an injective mapping, and let $\nu : G^2 \to G$ be any map that extends the multiplication in M, in the sense that, for any $m_1, m_2 \in M$,

$$\nu(\iota(m_1), \iota(m_2)) = \iota(m_1 m_2).$$

By Lemma 6, ν is represented by a polynomial w_2 over G. We claim that for each $n > 0$ there is a map $\nu_n : G^n \to G$ that extends n-ary multiplication in M, in the sense that for all $m_1, \ldots, m_n \in M$,

$$\nu_n(\iota(m_1), \ldots, \iota(m_n)) = \iota(m_1 \cdots m_n)$$

and that is represented by a polynomial w_n over G whose length is no more than $n^{1+\log_2 |w_2|}$ for sufficiently large n. We show this by induction on n. We already have the claim for $n = 2$, and we can set $w_1(x)$ to be x. Now suppose that $n > 2$, and the claim is true for all values less than n. If n is even, we have

$$\iota(m_1 \cdots m_n) = \nu(\nu_{\frac{n}{2}}(\iota(m_1), \ldots, \iota(m_{\frac{n}{2}})), \nu_{\frac{n}{2}}(\iota(m_{1+\frac{n}{2}}), \ldots \iota(m_n))).$$

We form the polynomial w_n by substituting the polynomials $w_{\frac{n}{2}}(x_1, \ldots, x_{\frac{n}{2}})$ and $w_{\frac{n}{2}}(x_{1+\frac{n}{2}}, \ldots, x_n)$ for the variables in w_2. The resulting polynomial has length no more than $|w_2| \cdot |w_{\frac{n}{2}}|$. If n is odd, then we can use the above argument to construct w_{n+1}, since $(n+1)/2 < n$. We can now set

$$w_n = w_{n+1}(x_1, \ldots, x_n, \iota(1)),$$

which gives $|w_n| = |w_{n+1}|$. Thus for all $n > 2$, we have

$$|w_n| \leq |w_2| \cdot |w_{\lceil \frac{n}{2} \rceil}|,$$

which implies

$$|w_n| \leq |w_2|^{\lceil \log_2 n \rceil} = |w_2| \cdot n^{1+\log_2 |w_2|},$$

which is less than $n^{2+\log_2 |w_2|}$ for sufficiently large n. This proves the claim.

Since $|w_1| = 1$, this implies that there exists $r > 0$ such that $|w_n| < n^r$ for all $n \geq 1$. It remains to show how to convert the sequence of polynomials $\{w_n\}$ into a k-program over G. We can write

$$\iota(m_1 \cdots m_n) = w_n(\iota(m_1), \ldots, \iota(m_n)) = f_1(m_{i_1}) \cdots f_p(m_{i_p}),$$

where the f_j are functions from M into G determined by the polynomial, and where for all j, $1 \leq i_j \leq n$, and $p \leq n^r$. We can form a sequence of $(r+1)$-tuples over $\{1, \ldots, n\}$:

$$(\mathbf{s}_1, i_1), \ldots, (\mathbf{s}_p, i_p),$$

where $\mathbf{s}_k$ denotes the k^{th} element of $\{1, \ldots, n\}^r$ in lexicographic order, and $(\mathbf{s}, j)$ denotes the $(r+1)$-tuple obtained by adjoining j as the last component to $\mathbf{s}$. We thus define, for $I \in \{1, \ldots n\}^{r+1}$,

$$\phi_{I,n} : M^{r+1} \to G$$

by

$$\phi_{I,n}(m_1, \ldots, m_{r+1}) = f_{i_j}(m_{r+1}),$$

if $I = (\mathbf{s}_j, i_j)$ for some j, and

$$\phi_{I,n}(m_1, \ldots, m_{r+1}) = 1,$$

otherwise. It follows that the value of Φ on the sequence $(m_1, \ldots, m_n)$ is $\iota(m_1 \cdots m_n)$. Since ι is injective, this shows G $(r+1)$-simulates M, completing the proof of Theorem 5.□

Theorem 5, and, especially, its application to boolean circuits, which we give in the next section, is due to Barrington [4], who rediscovered the principle of the Maurer-Rhodes theorem.

We don't really need a simple group larger than M—an only slightly more complicated argument shows that if G is *any* simple non-abelian group, then M is k-simulated by a direct product of copies of G, and that every regular language is k-recognized by G. It follows that the same is true for any finite monoid that contains a nonsolvable group, since any such monoid will have a finite simple non-abelian group as a divisor. Contrast this with Theorem 2, which implies severe restrictions of the 1-simulation power of finite groups.

It is widely conjectured that the nonsolvability is essential to this special computing power.

Conjecture 7. *If a finite monoid M is k-simulated by a solvable group G, then M is itself a solvable group, and every prime divisor of $|M|$ divides $|G|$.*

A finite monoid in which every group is solvable is called a *solvable monoid.*

Conjecture 8. *If a finite monoid M is k-simulated by a finite solvable monoid N, then M is solvable, and every prime divisor of the cardinality of a group in M divides the cardinality of some group in N.*

In the next section we shall see that these conjectures arise naturally from problems in the complexity theory of boolean circuits.

A *pseudovariety* of finite monoids is a class of finite monoids closed under division and finite direct products. We will say that a pseudovariety $\mathbf{V}$ of finite monoids is a *k-program variety* if any finite monoid that can be k-simulated, for some k, by a member of $\mathbf{V}$ is itself a member of $\mathbf{V}$. Theorem 5 says that if a k-program variety contains a nonsolvable group, then it contains all finite monoids. The two conjectures above say that $\mathbf{G}_{sol}$, the pseudovariety of finite solvable groups, and $\mathbf{M}_{sol}$, the pseudovariety of finite solvable monoids, are k-program varieties.

The following theorem lists some proper subvarieties of the pseudovariety of all finite monoids that are known to be k-program varieties. Parts (a)-(c) follow from results in circuit complexity—we shall say something about their proofs in the next section. Part (d) is, in essence, due to Maciel, Péladeau and Thérien [13], and, independently, to Straubing [20]. Part (e) follows from results of Barrington, Straubing and Thérien [6].

Theorem 9. *The following are k-program varieties.*

(a) The pseudovariety $\mathbf{Ap}$ *of finite aperiodic monoids.*

(b) The pseudovariety $\mathbf{G}_p$ *of p-groups, for a fixed prime p.*

(c) The pseudovariety $\mathbf{M}_p$ *of monoids all of whose groups are in* $\mathbf{G}_p$*, for a fixed prime p.*

(d) The pseudovariety $\mathbf{J}$ *of finite $\mathcal{J}$-trivial monoids.*

(e) The pseudovariety $\mathbf{G}_{nil}$ *of finite nilpotent groups.*

5 Circuit Complexity

A *circuit* with n inputs is a directed acyclic graph with $2n+2$ source nodes and a single sink node. The source nodes are labeled by the symbols

$$x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}, 0, 1$$

and each non-source node ν is labeled by a symmetric function $F_\nu : \{0,1\}^k \rightarrow \{0,1\}$, where k is the in-degree of the node. In all of the examples we consider, F_ν will be one of the following functions:

$$AND(x_1, \ldots, x_k) = 1 \Leftrightarrow x_1 = \cdots = x_k = 1$$

$$OR(x_1, \ldots, x_k) = 0 \Leftrightarrow x_1 = \cdots = x_k = 0$$
$$MOD_q(x_1, \ldots, x_k) = 1 \Leftrightarrow q | x_1 + \cdots + x_k$$

A circuit $\mathcal{C}$ with n inputs determines a function $f_{\mathcal{C}} : \{0,1\}^n \to \{0,1\}$, which we define as follows: Given $(a_1, \ldots, a_n) \in \{0,1\}^n$, we associate a bit to each of the nodes of the circuit, by induction on the distance from the node to a source. The source node labeled x_i is assigned the value a_i, the source node labeled $\overline{x_i}$ is assigned the value $1 - a_i$, and the source nodes labeled 0 and 1 are assigned the values 0 and 1, respectively. If ν is a node whose predecessors $\nu_1, \ldots, \nu_k$ have already been assigned values $b_1, \ldots, b_k$, then ν is assigned the value $F_\nu(b_1, \ldots, b_k)$. We set $f_{\mathcal{C}}(a_1, \ldots, a_n)$ to be the value assigned to the sink node. These values are well-defined because the underlying graph of the circuit is acyclic.

The *size* of a circuit is the number of edges it contains, and the *depth* of a circuit is the length of the longest path from the sink to a source. Ordinarily one looks at families of circuits $\{\mathcal{C}_n\}_{n \geq 0}$, where $\mathcal{C}_n$ has n inputs, and considers the language
$$L = \{w \in \{0,1\}^* : f_{\mathcal{C}_{|w|}}(w) = 1\}$$
recognized by the family. (Observe that circuits with 0 inputs make perfect sense.) Occasionally will we talk of circuits with more than one sink; such a circuit computes a function from $\{0,1\}^n$ to $\{0,1\}^m$, where m is the number of sinks.

Circuits, especially those in which the node functions are AND and OR, are a commonly studied model in computational complexity theory. One measure of the complexity of a language is the size, or depth, or some combination of these, of the smallest family of circuits required to recognize it. Now obviously, any language $L \subseteq \{0,1\}^*$ can be recognized by a family of circuits of depth 2 provided we allow the n^{th} circuit of the family to have $|L \cap \{0,1\}^n|$ AND nodes, and a single OR node at the sink. This requires, in general, that the size of the n^{th} circuit in the family grows exponentially in n. It is more interesting to consider what happens when we place a reasonable restriction on the circuit size. We define AC^0 to be the class of languages recognized by a constant-depth family of circuits with AND-nodes and OR-nodes, in which the size of the n^{th} circuit is bounded above by a polynomial in n. With such a family of circuits one can, for example, compare two integers given in binary notation, or, if we allow multiple outputs, add two numbers in binary. It is not at all clear at first if we can perform more complicated computations such as binary multiplication, or determining whether the number of bits in the input is even. One of the most important achievements of computational complexity theory is the following result, due to Furst, Saxe and Sipser [11] and, independently, to Ajtai [1].

Theorem 10. *Let* $q > 1$. $MOD_q \notin AC^0$.

This implies, by a relatively simple reduction, that multiplication cannot be performed by polynomial-size constant-depth circuit families. (See [11].)

The connection to finite monoids is given by the following theorem, due to Barrington and Thérien [7].

Theorem 11. *Let $L \subseteq \{0,1\}^*$. $L \in AC^0$ if and only if L is recognized by a k-program over a finite aperiodic monoid.*

Proof. Let $L \in AC^0$. Then L is recognized by a family of depth d circuits whose size is bounded by n^r for some constants d and r. We show by induction on d that there is a finite monoid M_d such that L is k-recognized by M_d, where k depends on d and r. If $d = 0$, then $L \cap \{0,1\}^n$ is one of the sets $\{0,1\}^n$, $\emptyset$, $\{a_1 \cdots a_n : a_i = 1\}$, or $\{a_1 \cdots a_n : a_i = 0\}$. In all four cases, L is recognized by a 1-program over the monoid $U_1 = \{0,1\}$, with accepting set $\{0\}$. In the first case, we just have all the program maps output the value 0 regardless of the input; in the second they all output the value 1. In the third case, all the program maps except the i^{th} give the value 1 on all inputs; the i^{th} map gives the value 0 on 1 and 1 on 0. In the fourth case, we just reverse these two values.

Now suppose $d > 0$ and that the claim is true for all smaller depths. We assume that the sink node is labeled AND; the proof is analogous if the sink node is OR. So

$$L \cap \{0,1\}^n = L_1 \cap \cdots L_t,$$

where $t \leq n^r$, and each L_i is recognized by a subcircuit of size no more than n^r and depth less than $d-1$. Thus each L_i is recognized by a $k(d-1,r)$-program over an aperiodic monoid M_{d-1}, with accepting set X_i.

M_d can be viewed as a monoid of right transformations on the set M_d. We embed this transformation monoid in the larger transformation monoid $\overline{M_d}$ formed by adjoining all the constant transformations on the set M_d. The underlying monoid of $\overline{M_d}$ is still aperiodic. We now extend this to the wreath product $U_1 \circ \overline{M_d}$, which is still aperiodic, because the aperiodic monoids are closed under wreath product. Recall that the elements of the wreath product have the form (γ, m), where m is a transformation in M_d, and γ is a map from M_d to U_1. We recognize $L \cap \{0,1\}^n$ by a program over this wreath product, in effect by running the programs for the L_i in succession, and using the constant transformations to reset M_d after each of these runs. More precisely, let $\{\phi_{I,n}^{(i)}\}$ be the program maps for L_i. If $I \neq (n, \ldots, n)$, then we define

$$\psi_{I,n}^{(i)}(a_1, \ldots, a_k) = (\alpha, \phi_{I,n}^{(i)}(a_1, \ldots, a_k)),$$

where $\alpha(m) = 1$ for all $m \in M_d$. If $I = (n, \ldots, n)$, then we define

$$\psi_{I,n}^{(i)}(a_1, \ldots, a_k) = (\alpha, \phi_{I,n}^{(i)}(a_1, \ldots, a_k)) \cdot (\beta, c_1),$$

where $\beta(m) = 1$ if $m \in X_i$ and $\beta(m) = 0$, otherwise; and where $c_1 : M_d \to M_d$ is the constant map to $1 \in M_d$. We now have $w \in L$ if and only if

$$(1,1) \cdot \prod_{i=1}^{t} \prod_{I \in \{1,\ldots,n\}^k} \psi_I^{(i)}(w_I) \in \{1\} \times M_d.$$

Since $t \leq n^r$ we can convert this, as in the proof of Theorem 5, to a sequence of $(k(d-1,r)+r+1)$-program maps over the underlying monoid of $U_1 \circ \overline{M_d}$.

Observe that we may have a different set of accepting values for different input lengths, depending upon whether the sink node is labeled AND or OR, but we can, as described in the note in Section 3, change this into a program over a direct product of copies of $U_1 \circ \overline{M_d}$, with a single set of accepting nodes.

We now prove the converse. First consider a language L over an arbitrary finite alphabet A. We can encode each element of A by a string of $\lceil \log_2 |A| \rceil$ bits, and thus produce a new language L' over $\{0,1\}$. We first claim that if L is a regular language recognized, in the sense of Section 1, by a finite aperiodic monoid M, then $L' \in AC^0$. To do this, we apply the classic theorem of Schützenberger [16], which characterizes the regular aperiodic languages: L is obtained from the language A^* by repeated application of boolean operations and the concatenation operation

$$(L_1, L_2) \mapsto L_1 a L_2,$$

where $a \in A$. Observe that

$$a_1 \cdots a_n \in L_1 a L_2 \Leftrightarrow \bigvee_{i=1}^{n} [(a_1 \cdots a_{i-1} \in L_1) \wedge (a_i = a) \wedge (a_{i+1} \cdots a_n \in L_2)].$$

We can thus use the construction of L from these basic operations as the blueprint for the construction of a circuit that recognizes the strings of length n in L'. The circuit contains OR nodes of in-degree 2 for the union operation, and NOT nodes for complementation. Note, however, that we can eliminate the NOT nodes by moving them to the level of the inputs, using DeMorgan's laws. Each operation used to construct L introduces at most two new levels of nodes into the circuit, and thus the depth of the circuit is bounded above by a constant that depends on L. Since the in-degree of each node introduced is at most n, the size of the resulting circuit is $O(n^d)$, where d is the depth.

Now suppose that $L \subseteq \{0,1\}^*$ is recognized by a k-program Φ over a finite aperiodic monoid M, with accepting set $X \subseteq M$. Let A_M be an alphabet in one-to-one correspondence with M, as in the proof of Theorem 2. The set

$$K = \{a_{m_1} \cdots a_{m_r} \in A_M^* : m_1 \cdots m_r \in X\}$$

is an aperiodic regular language over the alphabet A_M, hence the language K' obtained by replacing elements of A_M by their binary encodings, is in AC^0. Now consider an circuit with n inputs and $2 \cdot \lceil \log_2 |M| \rceil \cdot n^k$ outputs, which on input $w \in \{0,1\}^n$ gives the encoded values of $a_{\phi_I(w_I)}$, where I ranges over the elements of $\{1, \ldots, n\}^k$, and where ϕ_I are the program maps, as well as the negations of these binary strings. This circuit has depth 2, and consists of $2^k \binom{n}{k}$ AND nodes of in-degree k, fed into $2 \cdot \lceil \log_2 |M| \rceil \cdot n^k$ OR nodes of in-degree $O(n^k)$. We connect the outputs of this circuit to the inputs of a circuit that recognizes the strings in K' of length $\lceil \log_2 |M| \rceil \cdot n^k$ to obtain a circuit for $L \cap \{0,1\}^n$. This shows $L \in AC^0$ and completes the proof of Theorem 11 $\square$

Theorem 10 and Theorem 11 combine to prove Theorem 9(a), that the pseudovaritety **Ap** of finite aperiodic monoids is a k-program variety. For suppose that M is k-simulated by a finite aperiodic monoid N. If M is not aperiodic, it contains a cyclic group G of cardinality $q > 1$. If we identify 1 with the generator of G and 0 with the identity of G, we obtain, by Theorem 11, a family of k-programs over N recognizing MOD_q, which contradicts Theorem 10.

We also obtain a theorem analogous to Theorem 3 on the regular languages k-recognized by aperiodic monoids; these are the same as the regular languages 1-recognized by aperiodic monoids; the proof of this is the same as that of Theorem 3.

It is worth emphasizing that Theorem 9(a), a purely algebraic statement, is proved by appeal to a result from circuit complexity. No purely algebraic-combinatorial argument, along the lines of the proof of Theorem 2, is known. Such a proof would give a new proof of the circuit lower bound Theorem 10, for if we had an AC^0 circuit family recognizing MOD_q for some $q > 0$, then could easily use Theorem 11 to construct a k-program over an aperiodic monoid that k-simulates the cyclic group of cardinality q. Thus it is of more than passing interest to prove Theorem 9(a) directly, since this would introduce a new technique for proving lower bounds for circuits.

The language MOD_2, while not in AC^0, can easily be recognized by logarithmic-depth circuit families of AND nodes and OR nodes, even if we require each node to have in-degree 2. (We will prove this below, when we show that the multiplication in any finite monoid can be simulated in this fashion; here we are simulating the multiplication in the group of order 2.) In general, we define NC^1 to be the class of languages recognized by families of circuits where each node is labeled either AND or OR, and has in-degree 2, and where there exists a constant c such that the circuit for inputs of length n has depth no more than $c \cdot \log_2 n$. Note that the size of such a circuit is necessarily bounded by a polynomial in n. With NC^1 circuit families we can perform binary multiplication (if we allow multiple outputs), determine if the majority of the bits in the input are on, and even perform binary division. (See Chandra, Stockmeyer and Vishkin [9], and Beame, Cooke and Hoover[8].) As it turns out, NC^1 is exactly the class of k-recognizable languages. This is the content of the following theorem, due to Barrington [4].

Theorem 12. *Let $L \subseteq \{0,1\}^*$. The following are equivalent:*
(a) $L \in NC^1$.
(b) L is k-recognized by every finite simple nonabelian group.
(c) L is k-recognized by some finite monoid M.

Proof. ((a) $\Rightarrow$ (b)) Let G be a finite simple nonabelian group, and choose distinct elements $\iota(0), \iota(1)$ of G. We claim that there is a polynomial w over G with n variables such that for all $x_1, \ldots, x_n \in \{0,1\}$, $w(\iota(x_1), \ldots, \iota(x_n))$ is 1 if $x_1 \ldots x_n \in L$, and 0 otherwise; and that the length of w is bounded by $r \cdot n^k$ for some constants r and k. This follows by induction on the depth of the circuit recognizing $L \cap \{0,1\}^n$: If the depth is 0, then we take for our polynomial either the constant $\iota(0)$, the constant $\iota(1)$, $\iota(x_j)$, or a polynomial v in one variable satisfying $v(\iota(x)) = \iota(1-x)$ for $x \in \{0,1\}$. Such a polynomial exists by Lemma 6. Thus for $d = 0$ the length of the polynomial is bounded by a constant K. Now consider a circuit of depth d. The sink node is labeled AND or OR. By Lemma 6, there is a polynomial $v_\wedge$ over G in two variables realizing the AND function of two variables, in the sense that

$$v_\wedge(\iota(x_1), \iota(x_2)) = \iota(AND(x_1, x_2)).$$

Similarly, there is a polynomial $v_\vee$ realizing the OR function of two variables. By induction, the two subcircuits of depth $d-1$ whose outputs are inputs to the sink are realized by polynomials w' and w'' in n variables. We obtain a polynomial realizing the whole circuit by substituting w' and w'' for the two variables in $v_\wedge$ or $v_\vee$. The resulting polynomial has length no more than

$$\max(|v_\wedge|, |v_\vee|) \cdot \max(|w'|, |w''|).$$

It follows from the induction hypothesis that this is no more than

$$K \cdot \max(|v_\wedge|, |v_\vee|)^d \leq \max(|v_\wedge|, |v_\vee|)^{c \cdot \log_2 n} \leq r \cdot n^k,$$

for some constants r and k. We now argue as in the proof of Theorem 5 that L is k'-recognized by G for some G.

$((b) \Rightarrow (c))$ is trivial.

$((c) \Rightarrow (a))$ Let A_M be an alphabet in one-to-one correspondence with M, as in the proof of Theorem 2, and let $m \in M$. Let $X \subseteq M$, and consider the language

$$T = \{a_{m_1} \cdots a_{m_r} \in A_M^* : m_1 \cdots m_r \in X\},$$

as well as the language T' obtained from T by encoding each element of A_M by a string of $u = \lceil \log_2 |M| \rceil$ bits. The product of two elements of M is then represented by a function from $\{0,1\}^{2u}$ into $\{0,1\}^u$, and this function can be represented by a fixed circuit of AND and OR nodes of in-degree 2 and NOT nodes of in-degree 1. Let d be the depth of this circuit. It follows that the product of n elements of M is realized by a circuit of AND and OR nodes of in-degree 2 having depth $d \cdot (1 + \log_2 n)$, since we can move any NOT nodes to the level of the inputs. We attach to the outputs of this circuit a circuit of fixed size computing the function from $\{0,1\}^u$ into $\{0,1\}$ that determines if a bit string is an encoding of an element of X. This shows $T' \in NC^1$. We now argue as in the proof of Theorem 11, that if L is k-recognized by a M, then L is recognized by a circuit whose first two levels consist of AND nodes of constant in-degree followed by $O(n^k)$ OR nodes of in-degree $O(n^k)$, and whose subsequent levels are the circuit for T' that we have just constructed. Since we can simulate an OR node of in-degree $O(n^k)$ by a tree of OR nodes of in-degree 2 and depth $k \cdot \lceil \log_2 n \rceil$, we obtain $L \in NC^1$, completing the proof. □

Let us return to circuit families in which there is no bound on the in-degrees of the nodes, and in which the depth of the circuits in the family is constant. We will now allow these circuits to contain nodes that compute the MOD_q function, for a value of q fixed throughout the circuit family. We distinguish two ways in which this can be done. $L \subseteq \{0,1\}^*$ is said to be in $CC(q)$ if it is recognized by a polynomial-size constant-depth family of circuits in which every node is labeled by MOD_q; L is said to be in $ACC(q)$ if it is recognized by such a circuit family in which every node is labeled either MOD_q, AND or OR.

We state the following two theorems without proof. They are due to Smolensky [17].

Theorem 13. *If p is prime and $k \geq 1$, then $AND \notin CC(p^k)$.*

Theorem 14. *If p, q are distinct primes and $k \geq 1$, then $MOD_q \notin ACC(p^k)$.*

There are analogous conjectures for the case where the modulus of the circuit nodes is not a prime power.

Conjecture 15. *If $q > 1$ then $AND \notin CC(q)$. If p is a prime that does not divide q then then $MOD_p \notin CC(q)$.*

Conjecture 16. *If $q > 1$ and p is a prime that does not divide q, then $MOD_p \notin ACC(q)$.*

These are among the most outstanding unsolved problems in circuit complexity.

Now let us indicate the connections to algebra. The following theorems can be considered modular analogues of Theorem 11. Their proofs, which we omit, are similar. (See Barrington and Thérien [7] and Straubing [18].)

Theorem 17. *Let $q > 2$, $L \subseteq \{0,1\}^*$. $L \in CC(q)$ if and only if L is k-recognized by a solvable group whose cardinality divides a power of q.*

Theorem 18. *Let $q > 1$, $L \subseteq \{0,1\}^*$. $L \in ACC(q)$ if and only if L is k-recognized by a solvable monoid in which every group has cardinality dividing a power of q.*

We conclude this section by showing the equivalence between the theorems and conjectures stated above, and the results and problems cited at the end of Section 3.

Proof of Theorem 9(b). Suppose the finite monoid M is k-simulated by a p-group G. M must then be a group, because otherwise M contains an isomorphic copy of U_1, and thus AND is k-recognized by G. But since every p-group is solvable, this implies $AND \in CC(p)$, by Theorem 17, thus contradicting Theorem 13. $|M|$ cannot be divisible by a prime q different from p, because otherwise M would contain a cyclic group of order q, and MOD_q would be k-recognized by G, which implies $MOD_q \in CC(p)$, again contradicting Theorem 13. Thus M is a p-group. $\square$

Proof of Theorem 9(c). The argument is the same as the preceding; we do not need the observation about AND. □

Let P be a set of primes, and let $\mathbf{G}_{sol,P}$ denote the pseudovariety consisting of all finite groups G such that every prime divisor of $|G|$ belongs to P. Let $\mathbf{M}_{sol,P}$ denote the pseudovariety of finite monoids whose groups are all in $\mathbf{G}_{sol,P}$.

Theorem 19. *The following are equivalent.*
(a) Conjecture 15.
(b) For every set P of primes, $\mathbf{G}_{sol,P}$ is a k-program variety.

Proof. Assume Conjecture 15. Suppose M is k-simulated by $G \in \mathbf{G}_{sol,P}$. As in the proof of Theorem 9(b), we conclude M must be a group, and every prime dividing the cardinality of M must be in P. M must also be a solvable group, because if it were not, M could simulate the multiplication in a finite simple non-abelian group G, and thus by Theorem 12, k-recognize all languages in NC^1, in particular AND, contradicting the conjecture. Thus $M \in \mathbf{G}_{sol,P}$. Conversely, suppose that for every P, $\mathbf{G}_{sol,P}$ is a k-program variety. If $AND \in CC(q)$ for some q, then by Theorem 17 AND is k-recognized by a solvable group whose cardinality divides a power of q, and thus U_1 can be k-simulated by this group. This implies that $\mathbf{G}_{sol,P}$, where P is the set of prime divisors of q, is not a k-program variety, contradicting the assumption. We get the same contradiction if we suppose $MOD_p \in CC(q)$ for some prime p that does not divide q.

Theorem 20. *The following are equivalent.*
(a) Conjecture 16.
(b) For every set P of primes, $\mathbf{M}_{sol,P}$ is a k-program variety.

Proof. The proof is identical to the foregoing proof except that we do not need to reason about k-recognition of AND in either direction. □

6 $(\log^k n)$-simulation

In a k-program each instruction has access to only a fixed number of letters of the input string. If we allow the number of letters to which an instruction has access to grow with the input length, we might increase the power of the program. Of course, if we allow each instruction to access all n letters of the input, then we can simulate any monoid M by any monoid N that is as large as M—there will be no connection between the algebraic structure of M and N. However if the number of letters accessed grows slowly with M, then we find some interesting results.

Here we will look at the situation in which each instruction of the program for inputs of length n depends on $c \cdot \log^k n$ letters of the input string, for some constants c and k. If $f : \mathbf{N} \to \mathbf{N}$ is a function, then we define an $f(n)$-program over a monoid M exactly as we did in the case when f is constant: The program maps for inputs of length n are indexed by the $f(n)$-tuples over $\{1, \ldots, n\}$. The results we describe originate in work of Toda [21] on the polynomial-time hierarchy. Their interpretation in terms of circuits is due to Allender [2] and Allender and Hertrampf [3]. The semigroup-theoretic interpretation that we give here has not been published before.

First, let us look briefly at the question of $c \cdot log^k n$-programs over aperiodic monoids. If a finite monoid M is $c \cdot \log^k n$-simulated by a finite aperiodic monoid N, then by making a suitable translation between A_M and the binary alphabet $\{0,1\}$, we can simulate multiplication in M by a constant-depth circuit family where the size of the n^{th} circuit is $n^{O(\log^k n)}$—this is proved exactly like one direction of Theorem 11. Now results of Hastad [12] and Yao [22] extend Theorem 10 to the case of circuit families of this size; they prove that any constant-depth circuit family of AND nodes and OR nodes that recognizes MOD_q must have size exponential in n. Thus we conclude in this case as well that M must itself be aperiodic. Hastad also shows that for every $d > 0$ there is a language L recognized by a polynomial-size family of circuits of depth $d+1$ such that any circuit family of depth d recognizing L has exponential size. This implies (by suitably adapting the proof of Theorem 11) that there is no finite aperiodic monoid N such that every finite aperiodic monoid M is $\log^k n$-simulated by N, or by a direct product of copies of N.

It is a remarkable fact, then, that every finite aperiodic monoid can be $O(\log^k n)$-simulated by a direct product of copies of the monoid $U_1 \circ \overline{U_1} \circ G_2$, where G_2 is the cyclic group of order 2. We will prove this with circuits. We first define a *probabilistic circuit* with n inputs to be an ordinary circuit with $n+m$ inputs, for some $m > 0$. However we partition the inputs into two sets: The first n inputs, denoted $x_1, \ldots, x_n$, are called ordinary inputs, and the remaining m inputs, denoted $y_1, \ldots, y_m$, are called probabilistic inputs. Let $f : \{0,1\}^{m+n} \to \{0,1\}$ be the function computed by this circuit, and let $g : \{0,1\}^n \to \{0,1\}$ be a function. Let $\epsilon > 0$. We say that the circuit computes g with error at most ϵ if for all $(x_1, \ldots, x_n) \in \{0,1\}^n$,

$$|\{(y_1, \ldots, y_m) : f(x_1, \ldots, x_n, y_1, \ldots, y_m) \neq g(x_1, \ldots, x_n)\}| < 2^m \epsilon.$$

That is, we think of the probabilistic inputs as fair coins which we flip at the start of the computation; the probability that the circuit makes an error in computing g is at most ϵ. We also say that the circuit computes g with probability at least $1 - \epsilon$.

The following sequence of lemmas is adapted from [3].

Lemma 21. *Let $s > 0$. Both the OR and the AND function of n variables can be computed by probabilistic circuits of depth 2, with error less than 2^{-s}. The circuits consist of no more than $(n \cdot (s+1))^{2s+1}$ AND nodes on the first level, each of in-degree no more than $2s$, and a single MOD_2 gate at the output level.*

Proof. We first construct a circuit with n ordinary inputs $x_1, \ldots, x_n$ and n probabilistic inputs $y_1, \ldots, y_n$. The circuit consists of n AND nodes, each of in-degree 2 computing the values $x_i \wedge y_i$. The outputs of these nodes are fed into a single MOD_2 node. If $OR(x_1, \ldots, x_n) = 0$, then all the x_i are 0, and the output of the circuit is 1. If $OR(x_1, \ldots, x_n) = 1$, then let $x_{i_1}, \ldots, x_{i_r}$ be the ordinary inputs equal to 1. Note that $r > 0$. The circuit outputs 0 if and only if an odd number of $y_{i_1}, \dots, y_{i_r}$ are equal to 1, which occurs with probability $\frac{1}{2}$. Thus this circuit computes the negation of the OR function with probability at least $\frac{1}{2}$.

We now form the AND of s copies of this probabilistic circuit, with a new set of probabilistic inputs for each copy—that is, the resulting circuit has ns probabilistic inputs. If $OR(x_1, \dots, x_n) = 0$, then all s subcircuits output 1, so the circuit outputs 1. If $OR(x_1, \dots, x_n) = 1$ then each subcircuit outputs 1 with probability $\frac{1}{2}$, so the whole circuit outputs 1 with probability 2^{-s}. Thus this probabilistic circuit computes the negation of the OR function with error no more than 2^{-s}.

We now represent the behavior of this circuit by a polynomial in the $n \cdot (s+1)$ input variables over the field $\mathbf{Z}_2$. Each 2-input AND node computes a polynomial $x_i y_{ij}$, where $1 \leq i \leq n$ and $1 \leq j \leq t$. Each MOD_2 node outputs the value of the polynomial

$$1 + \sum_{i=1}^{n} x_i y_{ij},$$

and thus the whole circuit outputs the product of these s polynomials. The result is a polynomial of degree $2s$. We can write this polynomial as a sum of monomials, each of degree no more than $2s$. (Observe that since $\mathbf{Z}_2$ satisfies the identities $x^2 = x$ and $x + x = 0$, no variable need ever appear more than once in a monomial, and no monomial need ever appear more than once in a polynomial.) The number of distinct monomials of degree t is therefore $\binom{n \cdot (s+1)}{t} = O(n^t)$, and therefore the number of distinct monomials of degree no more than $2s$ is $O(n^{2s+1})$. If we replace each ordinary input x_i by $1 + x_i$ then we obtain a polynomial of the same degree that gives the value of $AND(x_1, \dots, x_n)$ with error probability no more than 2^{-s}. If we replace the polynomial P by $1 + P$ we obtain a polynomial of the same degree that gives the value of $OR(x_1, \dots, x_n)$ with error probability 2^{-s}. We can convert the polynomial back into a circuit in which each monomial of degree d is represented by the AND of d inputs; the outputs of these AND nodes are fed, together with a constant input 1, into a single MOD_2 gate, to compute the MOD_2 sum. (It may be that we have a monomial of degree 0, which will cancel the constant input, so the resulting circuit may or may not have a constant 1 at the input level.) □

Lemma 22. *Let $r, d, k > 0$. There exists $q > 0$ depending on r, d and k such that any family of circuits of constant depth d and size n^k with AND and OR nodes can be simulated by a family of probabilistic circuits with the following properties: Each circuit has depth 2, and consists of a single MOD_2 node whose inputs are the outputs of AND nodes; the in-degree of the MOD_2 node is $n^{O(\log^q n)}$; the in-degree of each AND node is $O(\log^q n)$; and the error probability is no more than $n^{-\log^r n}$.*

Proof. Given a circuit in the family of size n^k, we apply Lemma 21 to simulate each OR and AND node by a probabilistic circuit with error probability no more than 2^{-s}. The resulting probabilistic circuit has $n^{k+1}s$ probabilistic inputs, and has error probability no more than $n^k 2^{-s}$. We want to choose s so that

$$n^k 2^{-s} < n^{\log^r n}.$$

It suffices to choose $s > \log^{k+1} n + \log^r n$. In particular, we can set p to be the larger of $k+1$ and r, so that each AND node in the resulting probabilistic circuit has in-degree $O(\log^p n)$, and each MOD_2 node has in-degree $O(n^{2s+1}) = n^{O(\log^p n)}$.

We will now again apply the trick of representing the behavior of this probabilistic circuit by a polynomial over $\mathbf{Z}_2$ and manipulating the polynomial. Suppose we have an AND of m_1 MOD_2 nodes, where each MOD_2 node has m_2 inputs, given by polynomials over $\mathbf{Z}_2$. Let p_{ij} be the polynomial giving the j^{th} input to the i^{th} MOD_2 node. We can then represent the output of the AND node by the polynomial

$$\prod_{i=1}^{m_1}(1+\sum_{j=1}^{m_2} p_{ij}).$$

This can be rewritten as the sum of $m_2^{m_1}$ terms, each of which is a product of m_1 of the p_{ij}. When we represent the resulting polynomial by a circuit, we obtain a MOD_2 node of in-degree $m_2^{m_1}$, where each input is the output of an AND node of in-degree m_1. (We may have to add a constant input 1 to the MOD_2 node.) We apply this repeatedly to the probabilistic circuit we have constructed, switching AND nodes and MOD_2 nodes, until we obtain a circuit that consists of a tree of d levels of MOD_2 nodes, each with in-degree $n^{O(\log^{pd} n)}$ at the outputs, and trees of d levels of AND nodes, each of in-degree $O(\log^p n)$, at the inputs. We can collapse each tree of AND nodes to a single AND node of in-degree $O(\log^{pd} n)$, and collapse the tree of MOD_2 nodes to a single MOD_2 node of in-degree $n^{O(\log^{pd} n)}$. □

Remarkably, we can change the probabilistic circuit into an equivalent deterministic circuit by adding a single level of AND nodes followed by a single OR node.

Lemma 23. *Let $d, k > 0$. There exists $q > 0$ depending on r and d such that any family of circuits of constant depth d and size n^k with AND and OR nodes can be simulated by a family of circuits of depth 4 having the following structure: The output node is an OR node of in-degree n; each input to the output node is an AND node of in-degree n; each input to these AND nodes is a MOD_2 node of in-degree $n^{O(\log^q n)}$; each input to a MOD_2 node is an AND of $O(\log^q n)$ inputs.*

Proof. Call the circuit that we are trying to simulate $\mathcal{C}$. Let us take the probabilistic circuit constructed in the proof of Lemma 22 to simulate $\mathcal{C}$, with error probability no more than n^{-3}, and form the AND of n copies of the circuit, with a separate set of probabilistic inputs for each copy. We call this probabilistic circuit $\mathcal{D}$. Let us now take the OR of n copies of $\mathcal{D}$, again with a separate set of probabilistic inputs for each copy. Call this probabilistic circuit $\mathcal{E}$. If $\mathcal{C}$

outputs 1 then $\mathcal{D}$ outputs 1 with error probability no more than $n \cdot n^{-3} = n^{-2}$, and 0 with error probability no more than n^{-3n}. What is the probability of error of $\mathcal{E}$? If $\mathcal{C}$ outputs 1, then $\mathcal{E}$ fails only if all the copies of $\mathcal{D}$ fail, which occurs with probability no more than n^{-2n}. If $\mathcal{C}$ outputs 0, then $\mathcal{E}$ fails if one of the n copies of $\mathcal{D}$ fails, which occurs with probability no more than $n \cdot n^{-3n} = n^{1-3n}$. In either case, the error probability is less than 2^{-n}, as long as $n \geq 2$. (The trivial case $n = 1$ can be handled separately.) Since the error probability is less than 2^{-n}, there must be some setting of the t probabilistic inputs that gives the correct answer on all 2^n settings of the ordinary inputs. (Suppose to the contrary that for each setting of the probabilistic inputs, there is a setting of the ordinary inputs that gives the wrong answer. Then the number of settings of the whole set of inputs leading to an error is at least 2^t; but the assumption about the error probability implies that for each setting of the ordinary inputs there are fewer than 2^{t-n} settings of the probabilistic inputs that give rise to an error, and thus the total number of settings giving rise to an error is strictly less than 2^t, a contradiction.) Thus we can hard-wire these settings of the probabilistic inputs into the circuit, and obtain the desired result. □

We are now ready to prove the main result of this section. The construction is similar to the argument given in the proof of Theorem 11. Recall that the wreath product is an associative operation on transformation monoids. When we write a monoid M as a factor in a wreath product, we mean the transformation monoid whose underlying set of states is M, with the natural right action of M on itself. Recall as well that $\overline{U_1}$ denotes the transformation monoid U_1 with the two constant transformations adjoined. Let M be a finite aperiodic monoid. For each $m \in M$ we define the language

$$L_m = \{a_{m_1} \cdots a_{m_r} \in A_M^* : m_1 \cdots m_r = m\}.$$

As in the proof of Theorem 11, we form the language L'_m over $\{0, 1\}$, and argue that this language is in AC^0. It follows from Lemma 23 that L'_m is recognized by a circuit $\mathcal{E}$ of the kind described in the statement of that lemma. We claim that L'_m is $O(\log^q n)$-recognized by $U_1 \circ \overline{U_1} \circ G_2$, where G_2 denotes the cyclic group of order 2. To compute the output of a single MOD_2 node in the circuit, we note that each input to this node is an AND node whose value depends on $O(\log^q n)$ input bits. We can thus compute the value of the MOD_2 node by a sequence of $O(\log^q n)$-program maps with values in G_2. At the end of the sequence we feed this value (1 or 0) to the $\overline{U_1}$ co-ordinate of the wreath product. We now repeat the same sequence of program maps (without the last instruction that feeds the computed value to the middle co-ordinate); this has the effect of resetting the G_2 co-ordinate to the identity. We repeat this procedure with all the MOD_2 nodes attached to a single AND node. Once this is completed, the state of the middle co-ordinate is the value of the AND node, and the state of the right co-ordinate is the identity of G_2. We now feed 1 to the left co-ordinate if the state of the middle co-ordinate is 0, and 0 if the state of the middle co-ordinate is 1. We then reset the state of the middle co-ordinate to the identity, using the constant transformations. We repeat this for each of the n AND gates at the third level of the circuit. At the conclusion, the state of the leftmost co-ordinate of the wreath product will be 0 if the output of $\mathcal{E}$ is 1, and 1 if the output of $\mathcal{E}$ is 0. Each instruction has accessed a $c \cdot \log^q n$-tuple of input bits, and each input bit has been accessed n^2 times, so we can recognize L'_m with a $(c \cdot \log^q n + 2)$-program over this wreath product.

To simulate M, we run the $|M|$ programs recognizing the L'_m in parallel, and thus we can simulate M with a direct product of $|M|$ copies of $U_1 \circ \overline{U_1} \circ G_2$.

The argument above proves:

Theorem 24. *Let $L \in AC^0$. There exists a constant $q > 0$ such that L is $O(\log^q n)$-recognized by $U_1 \circ \overline{U_1} \circ G_2$. Let M be any finite aperiodic monoid. There exists a constant $q > 0$ such that M is $O(\log^q n)$-simulated by a direct product of copies of $U_1 \circ \overline{U_1} \circ G_2$.*

It is relatively easy to generalize this result to circuits in which several different prime moduli appear. We state the following theorem without proof.

Theorem 25. *Let P be a nonempty set of primes. Let $M \in \mathbf{M}_{sol,P}$. Then there exist a constant $q > 0$, and a group $G \in \mathbf{G}_{sol,P}$ such that M is $O(\log^q n)$-simulated by a direct product of $|M|$ copies of $U_1 \circ \overline{U_1} \circ G$.*

It would be interesting to find proofs of these theorems that do not require probabilistic arguments. Observe that in the proof of Lemma 23 we could have just as well taken the AND of OR's of copies of our probabilistic circuit as the OR of AND's. This leads one to suspect that Theorems 24 and 25 hold with something even simpler than $U_1 \circ \overline{U_1}$ as the left-hand factor in the wreath product. We venture the following guess:

Conjecture 26. *Let P be a nonempty set of primes. Let $M \in \mathbf{M}_{sol,P}$. Then there exist a constant $q > 0$, a group $G \in \mathbf{G}_{sol,P}$, and an $\mathcal{R}$-trivial monoid R such that M is $O(\log^q n)$-simulated by $R \circ G$.*

7 References

References

1. M. Ajtai, "Σ_1^1 formulae on finite structures", *Annals of Pure and Applied Logic* **24** (1983) 1–48.
2. E. Allender, "A note on the power of threshold circuits", *Proc. 30th IEEE FOCS* (1989) 580–584.
3. E. Allender and U. Hertrampf, "Depth reduction for circuits of unbounded fan-in", *Information and Computation* **112** (1994) 217–238.
4. D. Mix Barrington, "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1", *J. Comp. Syst. Sci.* **38** (1989) 150–164.
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, H. Straubing, and D. Thérien, “Nonuniform Automata over Groups”, *Information and Computation* **89** (1990) 109–132.
7. D. Mix Barrington and D. Thérien, “Finite Monoids and the Fine Structure of NC^1”, *JACM* **35** (1988) 941–952.
8. P. Beame, S. Cook, and J. Hoover, “Log-Depth Circuits for Division and Related Problems” *SIAM J. Computing* **15** (1986) 994–1003.
9. A. Chandra, L. Stockmeyer, and U. Vishkin, “Constant-Depth Reducibility”, *SIAM J. Computing* **13** (1984) 423–439.
10. Samuel Eilenberg, *Automata, Languages and Machines,* vol. B, Academic Press, New York, 1976.
11. M. Furst, J. Saxe, and M. Sipser, “Parity, Circuits, and the Polynomial Time Hierarchy”, *J. Math Systems Theory* **17** (1984) 13–27.
12. J. Hastad, “Almost Optimal Lower Bounds for Small-Depth Circuits”, *Proc. 18th ACM STOC* (1986) 6–20.
13. A. Maciel, P. Péladeau and D. Thérien, “Programs over Semigroups of Dot-depth One”, preprint, 1996.
14. W. Maurer and J. Rhodes, “A Property of Finite Simple Non-Abelian Groups”, *Proc. Amer. Math. Soc.* **16** (1965) 552–554.
15. J. E. Pin, *Varieties of Formal Languages,* Plenum, London, 1986.
16. M. P. Schützenberger, “On Finite Monoids Having Only Trivial Subgroups”, *Information and Control* **8** (1965) 190–194.
17. R. Smolensky, “Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity”, *Proc. 19th ACM STOC* (1987) 77–82.
18. H. Straubing, “Constant-depth Periodic Circuits”, *International J. Algebra and Computation* **1** (1991) 49–88.
19. H. Straubing, *Finite Automata, Formal Languages, and Circuit Complexity,* Birkhäuser, Boston, 1994.
20. “Languages Defined with Modular Counting Quantifiers”, Proc. 15th STACS, *Lecture Notes in Computer Science* **1373**, Springer, Berlin (1998) 332-343.
21. S. Toda, “PP is as hard as the polynomial-time hierarchy”, *SIAM J. Computing* **20** (1991), 865-877.
22. A. Yao, “Separating the Polynomial Time Hierarchy by Oracles”, *Proc. 26th IEEE FOCS* (1985) 1–10.