FINITE MONOIDS AND THE FINE STRUCTURE OF NC^1

David A. Mix Barrington
Denis Therien

COINS Technical Report 87-94

July, 1987

Finite Monoids and the Fine Structure of NC^1

David A. Mix Barrington[1]
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA 01003, U.S.A.

Denis Thérien[2]
School of Computer Science
McGill University
Montréal, P.Q. H3A 2K6, Canada

July 31, 1987

1. Abstract

Recently a new connection was discovered between the parallel complexity class NC^1 and the theory of finite automata, in the work of Barrington [Ba86] on bounded width branching programs. There (non-uniform) NC^1 was characterized as those languages recognized by a certain non-uniform version of a DFA. Here we extend this characterization to show that the internal structures of NC^1 and the class of automata are closely related.

In particular, using Thérien's classification of finite monoids [Th81], we give new characterizations of the classes AC^0, depth-k AC^0, and ACC, the last being the AC^0 closure of the mod q functions for all constant q. We settle some of the open questions in [Ba86], give a new proof that the dot-depth hierarchy of algebraic automata theory is infinite [BK78], and offer a new framework for understanding the internal structure of NC^1. A preliminary version of this work appeared as [BT87].

2. Introduction

Complexity theorists have only just begun a program of classifying problems according to the resources required to solve them by parallel algorithms, for example, the size and

[1] Former name David A. Barrington. Supported by NSF grant MCS-8304769 and US Air Force grant AFOSR-82-0326.

[2] Supported by NSERC grant A4546 and FCAR grant 86-EQ-2933

depth required to compute them using Boolean circuits. (The survey article of Cook [Co85] gives an overview of this program.) A new hierarchy of complexity classes has emerged, consisting of the classes NC^i and AC^i, of problems solvable by circuits of polynomial size and depth $O(\log^i n)$, with constant and unbounded fan-in respectively.

Here we consider one of the smallest of these new classes, NC^1, and its internal structure. Although problems well outside of NC^1 are still quite feasible for parallel computation, a close study of the relative difficulty of these easier problems can be very rewarding. A knowledge of how such low-level classes behave can give insights into the behavior of larger classes (where most of the analogous questions are unsolved) and can sometimes have a direct impact on those questions. (For example, the survey article of Johnson [Jo86] discusses the relevance of tightly constrained circuit complexity classes to the P versus NP question.)

The non-uniform class NC^1 is also mathematically interesting in itself. It has equivalent definitions as those languages definable by polynomial size Boolean formulas (equivalently, Boolean circuits which are trees) [Sp71] or by constant width polynomial size branching programs [Ba86]. As a constant width branching program can be thought of as a non-uniform DFA, the latter is the starting point for our investigations here. Non-uniform NC^1 contains all symmetric functions and other interesting functions such as integer multiplication and division [BCH84].

The first fact about the internal structure of NC^1 was provided by Furst et al. [FSS81] (and independently Ajtai [Aj83]) who showed that the parity function and several other NC^1 functions were outside of the class AC^0 (polynomial size, constant depth, unbounded fan-in). They introduced the notion of AC^0 reduction, which divides NC^1 into degrees like the Turing degrees of recursive function theory. Various reductions between interesting problems were found [CSV84, FKPS85] but until very recently it was not known that there were more than two degrees within NC^1, that of parity and the degree consisting of the class AC^0. The new contributions of Razborov [Ra86] and Smolensky [Sm87] are outlined in section 9.

The parity results showed that one cannot do modulo counting easily by combining AND and OR gates in a constant-depth way. Some more recent results [Ba85] suggested the difficulty of simulating an AND or OR by combining unbounded fan-in modulo counting gates in the same way, further suggesting a duality between the two kinds of gates. However, it was then shown that the ability to multiply in a non-solvable group, which might seem to be a generalization of modulo counting, was surprisingly powerful [Ba86]. (See any group theory text, such as [Za58], for the definition of a solvable group.)

In the meantime a curiously similar situation was arising in the theory of finite automata. Many finite automata can be built up in a simple way from threshold counters (essentially ANDs and ORs) and modulo counters (see section 4 for a summary of this theory or [Th81] for the actual results). The two types of counters cannot simulate one another, and combining them can produce exactly those automata which do not contain non-solvable groups.

Here we show extensive connections between the two theories. We are able immediately to use results from circuit complexity in automata theory, and we hope that the connections provide insight into both.

3. Background and Definitions

A *finite monoid* is a finite set with an associative binary operation and an identity element. Just as finite groups can be represented by groups of permutations, finite monoids can be thought of as sets of functions from some finite base set to itself, closed under composition and containing the identity function. The *word problem* for a monoid is to take an ordered sequence of elements of the monoid and find the product. We will define several specific classes of finite monoids in section 4 below.

A deterministic finite automaton can be viewed as a mapping from the input element A to a monoid M whose base set is the set of states. To each input letter we assign the function on the states undergone by the automaton on reading that letter. In algebraic terms, this single map from A to M induces a homomorphism from A^* to M, which we can use to recognize a language. We say that a language is recognized by the automaton if it is the inverse image, under this homomorphism, of a subset of M. The decision problem for any regular language is thus reduced to the word problem for a finite monoid, by a purely local translation.

We define a ***non-uniform deterministic finite automaton*** on the monoid M to be a machine which takes both an input in A^n and a ***program*** and recognizes some input language as follows. The program is a sequence of instructions in $[n] \times (A \to M)$, i.e., an instruction is a number in $\{1, \ldots, n\}$ together with a mapping from the input alphabet A to M. A program of *length* $p(n)$ and an input setting yield a sequence in $M^{p(n)}$ which is multiplied out to give an element of M, the output of the NUDFA.

This notion is easily seen to be equivalent to the bounded-width branching programs defined in [Ba86] — if M is the set of all functions on $[w]$ we have a width-w branching program, and if it is a group G we have a G-permutation branching program.

We will work with the following parallel complexity classes from the general theory outlined by Cook in the survey article [Co85], in their non-uniform forms. As stated above, we define NC^1 to be the class of languages recognizable by families of Boolean circuits of depth $O(\log n)$ and fan-in two (and hence polynomial size). We define AC^0 to be the class of languages recognizable by families of circuits of constant depth, polynomial size, and unbounded fan-in.

We will further classify languages in AC^0 by the exact constant bound on the depth of the circuits. We define a Σ_k circuit to be one with k alternating levels of unbounded fan-in AND and OR gates, with the output an OR gate and a 'zeroth level' of input gates.

As our input alphabets may have more than two letters, we allow our input gates to test '$x_i \in B$' for x_i any single input letter and B any subset of the input alphabet A. We define Π_k circuits analagously, where the output is an AND gate. We define a language L to be in 'depth-k AC^0' if each $L \cap A^n$ is recognized by a constant-sized Boolean combination of Σ_k and Π_k circuits, each of polynomial size.

Sipser proved [Si83] that the hierarchy of depth-k AC^0 classes was infinite, i.e., for every k there is a language in depth-k AC^0 which is not in depth-$(k-1)$ AC^0. His definitions differed somewhat from ours, but his methods suffice to prove our hierarchy to be infinite as well. Our choice is motivated by our desire to have a single family of languages called 'depth k' which is closed under constant-sized Boolean operations. The classes Σ_k and Π_k are each closed in this way, but their union is not. Note that a depth-k language in our terminology may be expressed as a family either of Σ_{k+1} or Π_{k+1} circuits.

A language L is AC^0-reducible to L' if there is a family of circuits recognizing L, of constant depth, polynomial size, and unbounded fan-in, which contains oracle nodes for L' along with AND and OR nodes. This extends the "constant depth truth-table" reducibility of [CSV84], so that the reductions shown there are AC^0-reductions in our model as well. Thus we can define AC^0 equivalence and AC^0 degrees by analogy with classical recursion theory.

We define the class ACC (the AC^0 closure of counters) to be the set of languages reducible to one of the languages $MOD_q = \{x \in \{0,1\}^* :$ the number of ones in x is divisible by $q\}$. In [Ba86] it is shown that the word problem for any non-solvable group is complete for NC^1 under AC^0 reductions, while the word problem for any solvable group is in ACC. AC^0 is known to be distinct from NC^1 by [FSS81] and [Aj83]. In [Ba86] it is conjectured that ACC is distinct from NC^1. The recent results of Razborov [Ra86] and Smolensky [Sm87] appear to support this conjecture — for more on this see section 9.

4. The Thérien Classification

We have defined a language $L \subseteq A^*$ to be ***recognizable*** iff there exists a finite monoid M and a homomorphism ϕ from A^* to M such that $L = L\phi\phi^{-1}$, i.e., L is the image under ϕ^{-1} of a subset of M. A variant of Kleene's theorem asserts that a language is regular iff it is recognizable. Moreover, this definition allows subclasses of the regular languages to be put in correspondence with families of finite monoids. For example, Schützenberger [Sc65] has shown that a regular language is star-free (i.e., can be defined from the letters of the alphabet using boolean operations and concatenation, but not the closure operator) iff it is recognizable by a finite *aperiodic monoid*. This is a monoid in which all elements m satisfy the equation $m^t = m^{t+1}$ for some $t \geq 0$. Several other examples of such correspondences were found and a general framework, the theory of pseudo-varieties, has been developed by Eilenberg [Ei76] to discuss these results in a uniform manner.

We will say that a monoid M is *solvable* iff any subset of M which is a group is solvable. An aperiodic monoid is solvable since any subset which is a group has cardinality one. We will present a combinatorial characterization of those regular languages which are recognizable by solvable monoids, where membership of a word x is determined by counting the ways x can be factored into subwords following certain rules. This characterization will be used in the next section to algebraically classify various families of circuits.

There are two essentially different kinds of counting which can be done by a finite state machine. For $t \geq 0$, threshold-t counting gives us the relation $\theta_{t,1}$, where $i\theta_{t,1}j$ iff $i = j$ or both $i \geq t$ and $j \geq t$. For $q \geq 1$, modulo-q counting gives the relation $\theta_{0,q}$ where $i\theta_{0,q}j$ iff $i \equiv j \pmod{q}$. We can combine these two kinds of counting to get an arbitrary congruence $\theta_{t,q}$ of finite index on the natural numbers N, where $i\theta_{t,q}j$ iff $i\theta_{t,1}j$ and $i\theta_{0,q}j$. Note that the monoid $N/\theta_{t,q}$ is a group iff $t = 0$ and aperiodic iff $q = 1$.

For any languages $L_0, L_1 \subseteq A^*$ and $a \in A$, define the language $[L_0, a, L_1, k]_{t,q}$ to be all those words x having j distinct factorizations of the form $x = x_0 a x_1$ where $x_0 \in L_0$, $x_1 \in L_1$, and $j\theta_{t,q}k$. Thus membership of x in this language is determined by counting (with respect to $\theta_{t,q}$) those occurrences of the letter a that occur in the context of a prefix in L_0 and a suffix in L_1.

If $\mathbf{L}$ is any family of languages over an alphabet A, let $(\mathbf{L})$ denote its Boolean closure. For each $t \geq 1$ and $q \geq 0$, we define a hierarchy of families $\mathbf{M}^i_{t,q}$ by:

$$\begin{aligned} \mathbf{M}^0_{t,q} &= (A^*) \\ \text{and for } i > 0,\ \mathbf{M}^i_{t,q} &= (\{[L_0, a, L_1, k]_{t,q} | \\ &\qquad L_0, L_1 \in \mathbf{M}^{i-1}_{t,q}, k \in N\}) \end{aligned}$$

The following results can be found in [Th81]:

Theorem 1: A language L is recognizable by a finite aperiodic monoid iff L is in $\mathbf{M}^i_{t,1}$ for some $i, t \geq 0$ iff L is in $\mathbf{M}^i_{1,1}$ for some $i \geq 0$.□

Theorem 2: A language L is recognizable by a finite solvable group iff L is in $\mathbf{M}^i_{0,q}$ for some $i \geq 0$, $q \geq 1$.□

Theorem 3: A language L is recognizable by a finite solvable monoid iff L is in $\mathbf{M}^i_{t,q}$ for some $i, t \geq 0$, $q \geq 1$ iff L is in $\mathbf{M}^i_{1,q}$ for some $i \geq 0$, $q \geq 1$.□

We can interpret these results as giving a way to construct arbitrary solvable monoids using the two types of counters as our basic building blocks. The method of connecting counters together is essentially a two-sided version of the series connection or wreath product of automata (see, e.g., [Ei76]). "Context" when speaking of finite automata generally refers only to the prefix, but in analyzing parallel computations we will want to consider the two-sided context of a letter.

If we restrict our building blocks to be modulo counters, we generate exactly the solvable groups. If we restrict to the threshold counters (or even to threshold-1 counters) we generate exactly the aperiodic monoids. In this sense the two types of counters are "orthogonal". Monoids which contain non-solvable groups are not representable at all in this framework.

5. Our New Results

We extend this classification of monoids to a classification of languages in NC^1, to wit:

Theorem 4: A language is in AC^0 iff it can be recognized by a family of polynomial length programs for a non-uniform DFA on some aperiodic monoid.

This theorem extends the central result of Chandra et al. [CFL83] that the word problem for a semigroup is in AC^0 iff the semigroup is "group-free", i.e., aperiodic.

Theorem 5: A language is in ACC iff it can be recognized by a family of polynomial length programs for a non-uniform DFA on some solvable monoid.

We include for comparison:

Theorem 6: [Ba86] A language is in NC^1 iff it can be recognized by a family of polynomial length programs for a non-uniform DFA on some finite monoid. □

Proof of Theorem 4: First, we show that the word problem for an aperiodic monoid is in AC^0 — this clearly implies that we can determine the output of the poly-length program on the non-uniform DFA in AC^0. Let L be a language in $\mathbf{M}_{1,1}^{\ell}$ for some ℓ. If $\ell = 1$ we must check the input string for membership in a constant number of languages of the form A^*aA^*, and for each of these we simply use one OR gate of fan-in $O(n)$ to search for the letter a. Thus we need a constant-size Boolean combination of poly-size ORs and we are in depth 1 AC^0, using our definitions.

Assume by induction that we have an circuit of depth $\ell - 1$ and polynomial size for any language in $\mathbf{M}_{1,1}^{\ell-1}$. L is then the Boolean combination of languages of the form $L_0 a L_1$, with L_0 and L_1 in $\mathbf{M}_{1,1}^{\ell-1}$. Each of these conditions can be checked with a linear-sized OR (to search for the a) of depth $\ell - 1$ circuits (to test the prefix and postfix of the a). It is easily shown that a polynomial size OR of depth $\ell - 1$ circuits in our definition is depth ℓ, so that the circuit for L is depth ℓ and polynomial size. This closes the induction and proves the first half of the theorem.

For the second half of the proof, we must show that an arbitrary AC^0 circuit can be simulated by an NUDFA over an aperiodic monoid. For each $k \geq 1$ we will define an aperiodic monoid M_k: to any circuit C of depth k and any input string x we then associate a string $\hat{C}_x$ of elements of M_k (roughly preserving circuit size as string length) such that the product of the string in the monoid tells whether the circuit accepts the input.

Our monoids M_k will each be generated by the set $A = \{0, 1, <_\wedge, >_\wedge, <_\vee, >_\vee\}$. The $<$ and $>$ symbols of each type will serve as parentheses as we denote the circuit by a Boolean formula in infix notation (with a constant-sized bound on the nesting of parentheses). Without loss of generality we assume that our circuit is a tree and that all inputs to a gate at level i come from gates at level $i-1$. For each gate g we define a string $\hat{g}$ in A^* and we let $\hat{C}_x$ be the string encoding the output gate of C. The encoding of each gate is given level by level. If g is at level 1 then $\hat{g}$ is the concatenation of the $0-1$ values entering the gate. If g is of level $i > 1$, $g_1, \ldots, g_r$ are the OR gates of level $i-1$ feeding into g, and $h_1, \ldots, h_s$ are the AND gates at level $i-1$ feeding into g, then $\hat{g}$ is defined as

$$<_\vee \hat{g_1} >_\vee <_\vee \hat{g_2} >_\vee \ldots <_\vee \hat{g_r} >_\vee <_\wedge \hat{h_1} >_\wedge \ldots <_\wedge \hat{h_s} >_\wedge$$

The order of the input gates in the encoding is unimportant, as AND and OR are commutative operations.

We define the following subsets of A^*:

$$\begin{aligned}
VALID_1 &= (0+1)^+ \\
VALID_i &= (<_\vee VALID_{i-1} >_\vee + <_\wedge VALID_{i-1} >_\wedge)^+ \\
\\
OR_1 &= VALID_1 \cap A^* 1 A^* \\
\overline{OR_1} &= VALID_1 \setminus OR_1 \\
\overline{AND_1} &= VALID_1 \cap A^* 0 A^* \\
AND_1 &= VALID_1 \setminus \overline{AND_1} \\
\\
OR_i &= VALID_i \cap \\
&\quad (A^* <_\vee OR_{i-1} >_\vee A^* + A^* <_\wedge AND_{i-1} >_\wedge A^*) \\
\overline{OR_i} &= VALID_i \setminus OR_1 \\
\overline{AND_i} &= VALID_i \cap \\
&\quad (A^* <_\vee \overline{OR_{i-1}} >_\vee A^* + A^* <_\wedge \overline{AND_{i-1}} >_\wedge A^*) \\
AND_i &= VALID_i \setminus \overline{AND_i}
\end{aligned}$$

The intended interpretation is as follows. $VALID_i$ is the set of strings which represent valid encodings for gates at level i. (Of course, it includes encodings which do not have all their OR gates before their AND gates.) OR_i and AND_i are those strings representing accepting OR gates and AND gates at level i, respectively. If the output gate of a depth-k circuit C is an OR then C accepts x iff $\hat{C}_x \in OR_k$; similarly if it is an AND gate. Note that except for the bound on nesting depth, this is essentially the "formula value problem" language introduced by Lynch [Ly77] and recently shown to be in NC^1 by Buss [Bu87].

Now let M_k be the direct product of the syntactic monoids of OR_k and AND_k. This monoid is aperiodic because all the languages defined above are star-free — this is easy to see by induction given the case of $VALID_i$, which will follow from a lemma in the next section. (proved using a lemma of Brzozowski and Knast [BK78]). The NUDFA program to simulate C is constructed in such a way that it produces on input x the value in M_k of the string $\hat{C}_x$.□

Proof of Theorem 5: This is similar to the proof of Theorem 4, with the addition of modulo counters and MOD-q gates. In the first half of the proof, if L is in $\mathbf{M}^{\ell}_{t,q}$ we will need a MOD-q gate at each level to add up the number of factorizations found at the previous level. In the second half, we must add new symbols $<_{\oplus_q}$ and $>_{\oplus_q}$ to our alphabet A to represent a MOD-q gate. We redefine $VALID_i$ to include concatenations of strings of the form $<_{\oplus_q} VALID_{i-1} >_{\oplus_q}$, and define languages $(MOD-q)_i$ simultaneously with AND_i and OR_i. Note that these new languages $VALID_i$ are still star-free, by the lemma in the next section. $(MOD-q)_i$ will consist of all strings in $VALID_i$ which have an appropriate number of accepting gates (i.e., substrings of the form $<_\vee OR_{i-1} >_\vee$, $<_\wedge AND_{i-1} >_\wedge$, or $<_{\oplus_q} (MOD-q)_{i-1} >_{\oplus_q}$) modulo q. Determining membership in $(MOD-q)_i$ thus requires precisely $\theta_{0,q}$ counting of letters (the terminal $>_\vee$'s, $>_\wedge$'s or $>_{\oplus_q}$'s) in a level $i-1$ context. By induction, each of these languages is therefore seen to be in $\mathbf{M}^{i}_{1,q}$ and thus solvable. We omit any further details.□

6. The Depth Hierarchies

The class AC^0 is parametrized by the depth of the circuits, giving the depth-k hierarchy shown to be infinite by Sipser [Si83]. Given the correspondence just proved between AC^0 and aperiodic monoids, it is natural to look for a hierarchy of subclasses of the aperiodic monoids corresponding to depth-k AC^0.

The answer turns out to be the dot-depth hierarchy of Brzozowski [CB71, Ei76]. The star-free languages can be built up by concatenation and Boolean operations, and dot-depth measures the minimum number of times concatenation must be used in this process. More precisely, we define the class $\mathbf{D}^k$ of dot-depth k languages as the union of the classes $\mathbf{D}^k_m$ for all m, defined as follows (using notation from the last section):

$$\begin{aligned}
\mathbf{D}^0_m &= (A^*) \\
\mathbf{D}^1_m &= (\{A^*aA^* | a \in A\}) \\
\mathbf{D}^k_m &= (\{L_0a_1L_1\ldots a_rL_r | a_i \in A, L_i \in \mathbf{D}^{k-1}_m, r \leq m\})
\end{aligned}$$

A monoid has dot-depth k iff all the languages recognized by it do. This definition, designed for use with monoids, differs slightly from the standard semigroup-oriented one [Ei76, BK78]. There are several competing definitions of dot-depth 1 for monoids, all of which lead to the same definitions of dot-depth i for $i \geq 2$. (See, for example, [PS81].)

Every language in any $\mathbf{D}^k_m$ is star-free by construction, and since $\mathbf{M}^k_{1,1} = \mathbf{D}^k_1$ all star-free languages belong to some $\mathbf{D}^k$. The dot-depth thus parametrizes aperiodic monoids.

We have made many choices in the exact definitions of both the dot-depth and depth-k AC^0 hierarchies, which were taken in order to make the following theorem as simple as possible. Analagous theorems for different choices of the definitions can be proved by the same methods, though the exact correspondence between levels of the hierarchies may suffer.

Theorem 7: A language is recognized by a family of polynomial length NUDFA programs on a monoid of dot-depth k iff it is in depth-k AC^0.

Proof: We merely have to keep track of the depth in the proof of Theorem 4. In the first half we showed that any language in $\mathbf{M}^1_{1,1}$ (which equals $\mathbf{D}^1_m$) was in depth-1 AC^0. A similar induction can then show that any language in $\mathbf{D}^\ell_m$ is in depth ℓ — we need a polynomial size OR gate to search over the possible partitions of the input.

In the second half, it suffices to observe that given the following lemma, the languages $VALID_k$, AND_k, and OR_k each have dot-depth k by induction on their definitions. This lemma also proves that each language $VALID_k$ is star-free, as claimed in the previous section. □

Lemma: The language $VALID_k$ defined above has dot-depth k.

Proof: It is fairly easy to see that $VALID_1$ has dot-depth 1:

$$(0+1)^+ = A^* \setminus ((\bigcup_{a \notin \{0,1\}} A^*aA^*) \cup (A^* \setminus \bigcup_{a \in A} A^*aA^*))$$

We proceed by induction for $k > 1$. Brzozowski and Knast [BK78] prove an analogous result for a similar inductively defined language with only one type of parentheses — their methods essentially suffice here. Consider a recognizer for $VALID_k$ which reads the input string from left to right and maintains a stack of the left parentheses it has seen which have not yet been matched. It rejects the input if (a) it sees a right parenthesis which does not match the left parenthesis on the top of the stack, (b) it sees a right parenthesis when the stack is empty, (c) it sees a left parenthesis when the stack already has size $k-1$, (d) it completes a pair of depth $k-1$ parentheses without an intervening 0 or 1, (e) it sees a 0 or 1 when the stack depth is not exactly $k-1$, or (f) it reaches the end of the string without ever seeing a 0 or 1. The reader may verify that the input is in $VALID_k$ iff none of these things happen. Since the stack has only a constant number of possible contents for a particular k, this recognizer is a finite state machine. Following the argument of Brzozowski and Knast, we can describe the language $\overline{VALID_k}$ of strings rejected by this machine as a union of several expressions of dot-depth k, using the inductive hypothesis that each $VALID_i$ for $i < k$ has dot-depth i. The details are tedious but straightforward, and are omitted. Note that the modification of $VALID_k$ to allow any constant number of parenthesis types does not affect the proof. □

The dot-depth hierarchy for semigroups was proved to be infinite by Brzozowski and Knast [BK78], confirming a conjecture of Cohen and Brzozowski [CB71]. A semigroup is a set with an associative binary operation, but not necessarily with an identity. The algebraic theories of monoids and of semigroups are similar, but not identical. Brzozowski and Knast's methods suffice to prove the analogous theorem for monoids (see [St86] for background). Our connection between the depth-k and dot-depth hierarchies immediately provides a new proof of the dot-depth results (for either monoids or semigroups) using Sipser's result [Si83] for the depth-k AC^0 hierarchy. We do not get a new proof of [Si83] from the dot-depth results because they do not rule out the possibility that non-regular languages in depth-k AC^0 might be in depth-$k-1$.

Brzozowski and Knast give an example of an automaton which recognizes a language of dot-depth exactly k — it will be interesting to compare this example with the language of dot-depth exactly k provided by Sipser's theorem, and see whether either proof technique has further relevance in the other setting.

7. Branching Program Results

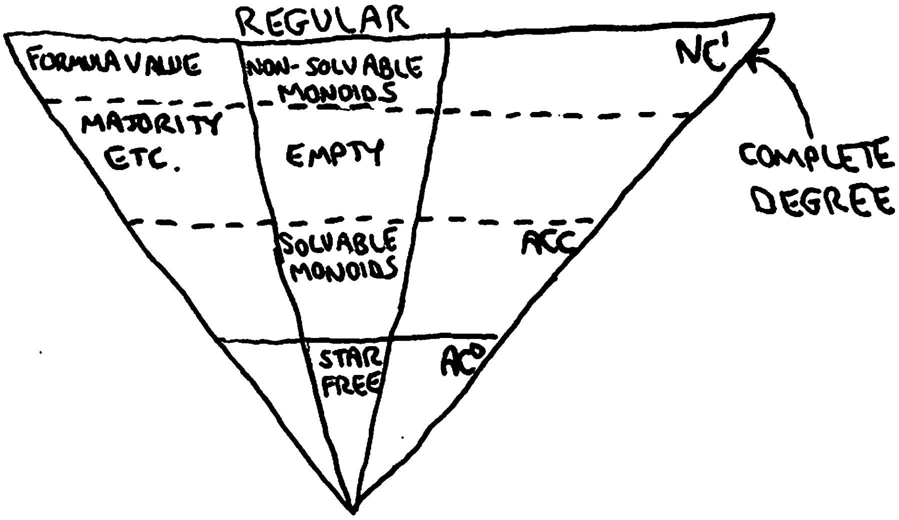
One of the most puzzling aspects of the main construction of [Ba86] was the fact that the restriction of branching programs to permutation branching programs did not appear to affect their power in the width 5 case. In [Ba85], however, it was seen that this restriction does matter in the width 3 case. Our results here begin to tell us what is really going on. Increased width increases the power of a branching program by increasing the complication of the monoid in which the individual instructions lie. For width 4 or less, this monoid is solvable, and computing power comes from the presence of modulo counters and threshold counters in the way we have outlined. Naturally, restricting to only modulo counters reduces the available power. But once we have a non-solvable group available, which can be used to simulate both threshold and modulo counting and more besides, the restriction to a group rather than a monoid is no longer important.

In [Ba86] it was asked whether width 3 or width 4 unrestricted branching programs were strictly less powerful than NC^1, given polynomial size. It was shown there that width 5 was necessary for a *permutation* branching program to recognize majority in polynomial size, modulo the conjecture that ACC is strictly contained in NC^1. We now see by Theorem 5 that the same conjecture implies that width 5 is necessary for an unrestricted branching program to do so. The extra freedom to use non-permutation instructions does not help enough to do any language outside of ACC.

8. The Fine Structure of $\mathbf{NC^1}$

We can now draw a much more complete map of NC^1 as it is structured by AC^0 reductions. The regular languages form a spine for the whole class, extending up through the complete degree. Along with its regular languages, this complete degree contains the formula value problem of Lynch [Ly77] and various related languages, as proved recently by

Buss [Bu87] A solid line on the map shows the known separation of AC^0 from NC^1 [FSS81, etc.] , and dotted lines show various conjectures — that majority is not complete and that majority is not in ACC.



Our new understanding (and the new results described in the next section) still leave a number of interesting questions. Is the degree of mod p a minimal degree? Is the structure of ACC as we conjecture it? Is majority (and thus the many problems shown equivalent to majority in [CSV84]) complete for NC^1? This last question amounts to characterizing the problems reducible to majority, a non-uniform version of the class already defined by Parberry and Schnitger [PS86] as constant depth and polynomial size in their threshold gate model.

We can also consider the subset of NC^1 consisting of the symmetric functions, for which majority is AC^0 complete. It should now be possible to add to the work in [FKPS85] and elsewhere.

9. Recent Structure Results

In [FSS81] it was conjectured that majority is not AC^0 reducible to parity, and this conjecture has just been proved by Razborov [Ra86]. He shows that any constant-depth circuit of unbounded fan-in AND, OR, and PARITY gates may be approximated by a polynomial over the field $GF(2)$ of relatively low degree, and then shows that there is a symmetric function which cannot be so approximated. The result follows from the AC^0-completeness of majority for symmetric functions (e.g., [FKPS85]).

Smolensky [Sm87] has extended these techniques to polynomials over other finite fields and developed simpler methods for showing that a function must have large degree. In this way, he has proved the AC^0 independence of the mod p and mod q for p and q distinct primes, and also that the mod p functions all lie strictly below majority.

The direct consequence of this for branching programs is that majority cannot be calculated by a polynomial size M-BP where M is a monoid containing no groups other than p-groups (groups whose order is a power of p, a prime fixed for the monoid).

These new techniques offer new possibilities for showing that ACC is different from NC^1. Currently, however, they run aground in placing any limits on the power of AC^0 circuits with mod 6 gates, or equivalently both mod 2 and mod 3 gates (or mod pq, in general). In his paper, Razborov expresses an interest in extending his work to arithmetic circuits over other finite fields, but expresses doubts that his approximation technique extends directly. Arithmetic circuits over $GF(2)$ correspond directly to AC^0 circuits with parity gates, but this breaks down for other finite fields. To simulate an arithmetic circuit for $GF(q)$, one needs not only mod q gates to implement addition, but also mod $q-1$ gates to implement unbounded fan-in multiplication. All poly-size arithmetic circuits for finite fields can be simulated in ACC, however.

We can, at least, now add some new solid lines to our map — between the various mod p functions and between each of those and majority. There is no direct evidence, however, that these lines should go where we have our dotted lines. We don't know, say, that the mod 6 function is not AC^0 complete for NC^1, though everything we have found here seems to us to suggest that it is not.

10. Open Problems

We have found that in general NUDFA's over interesting varieties of monoids lead to interesting classes of languages. A clear direction to explore is the power of NUDFA's over various varieties of groups. This is taken up in [BT87a], which applies the subword counting theory of [Th83] and generalizes the methods of [Ba85] (for branching programs over S_3) to obtain results about nilpotent and solvable groups.

Much of the new fine structure theory here depends on the conjecture of [Ba86], that ACC is strictly contained in NC^1. It would to nice to prove this, either by methods like those of Razborov or Smolensky or by advancing the ideas contained here and in [BT87a]. Currently we cannot even prove that the mod 6 function is not complete for NC^1. We also have the many open questions in the fine structure theory outlined in section 8.

Our results offer some hope of using automata results such as [BK78] to prove circuit results such as [Si83] without using the random restriction technology. Could this be carried out, and even extended to get similar proofs of [FSS81], [Ya85], or [Hå86]? Do random restrictions have any more to contribute to automata theory besides the new proof of [BK78]?

One could examine other models similar to NUDFAs, for example, non-uniform stack machines. This might extend these techniques to complexity classes larger than NC^1.

Straubing [St86a] has developed a framework generalizing both circuits and branching programs, which provides new proofs of some of our results.

Finally, we would like to develop an algebraic theory to explain all this, analogous to Eilenberg's treatment of finite automata in [Ei76].

11. Acknowledgements

We would like to thank Pierre McKenzie, who first noticed the connections between our efforts and organized the meeting in Montréal where we obtained the first of these results. We would also like to thank Andrew Goldberg for providing an early translation of the Razborov paper [Ra86] and thank Ravi Boppana, Mike Sipser, Roman Smolensky, and Howard Straubing for helpful discussions. This paper was partially prepared using the facilities of the M.I.T. Mathematics Department, to whom we are grateful. The preliminary version of this work [BT87] is copyrighted by the ACM and material from it appears here with their permission.

12. References

[Aj83] M. Ajtai, 'Σ^1_1 formulae on finite structures', Annals of Pure and Applied Logic 24 (1983), 1-48.

[Ba85] D. A. Barrington, 'Width 3 Permutation Branching Programs', Technical Memorandum TM-291, M.I.T. Laboratory for Computer Science, Dec. 1985.

[Ba86] D. A. Barrington, 'Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1', Proc. 18th ACM STOC, 1986, 1-5. Journal version J. Comp. Sys. Sci., to appear.

[Ba86a] D. A. Barrington, 'Bounded width branching programs', Ph.D. thesis, Dept. of Mathematics, M.I.T., May 1986. Available as Technical Report TR-361, M.I.T. Laboratory for Computer Science.

[BCH84] P. W. Beame, S. A. Cook, and H. J. Hoover, 'Log-depth circuits for division and related problems', Proc. 25th IEEE FOCS, 1984, 1-6. Journal version SIAM J. Comp 15 (1986), 994-1003.

[BK78] J. A. Brzozowski and R. Knast, 'The dot-depth hierarchy of star-free languages is infinite', J. Comp. Sys. Sci. 16 (1978) 37-55.

[BT87] D. A. Barrington and D. Thérien, 'Finite monoids and the fine structure of NC^1', Proc. 19th ACM STOC, 1987, 101-109.

[BT87a] D. A. Barrington and D. Thérien, 'Non-uniform automata over groups', Proc. 14th ICALP, 1987, 163-173.

[Bu87] S. R. Buss, 'The Boolean formula value problem is in ALOGTIME', Proc. 19th ACM STOC, 1987, 123-131.

[CB71] R. S. Cohen and J. A. Brzozowski, 'Dot-depth of star-free events', J. Comp. Sys. Sci. 5 (1971) 1-16.

[CFL83] A. K. Chandra, S. Fortune, and R. Lipton, 'Unbounded fan-in circuits and associative functions', Proc. 15th ACM STOC, 1983, 52-60.

[CSV84] A. K. Chandra, L. Stockmeyer, and U. Vishkin, 'Constant-depth reducibility', SIAM J. Computing 13 (May 1984) 423-439.

[Co85] S. A. Cook, 'The taxonomy of problems with fast parallel algorithms', Information and Control 64 (Jan. 1985), 2-22.

[Ei76] S. Eilenberg, *Automata, Languages, and Machines*, Vol. B (Academic Press, New York, 1976).

[FKPS85] R. Fagin, M. M. Klawe, N. J. Pippenger, and L. Stockmeyer, 'Bounded depth, polynomial-size circuits for symmetric functions', Theoretical Computer Science 36 (1985) 239-250.

[FSS81] M. Furst, J. B. Saxe, and M. Sipser, 'Parity, circuits, and the polynomial-time hierarchy', Proc. 22nd IEEE FOCS, 1981, 260-270. Journal version Math. Systems Theory 17 (1984) 13-27.

[Hå86] J. Håstad, 'Almost optimal lower bounds for small depth circuits', Proc. 18th ACM STOC, 1986, 6-20.

[Jo86] D. S. Johnson, 'The NP-completeness column: An ongoing guide', Journal of Algorithms 7:2 (June 1986), 289-305.

[Ly77] N. Lynch, 'Log space recognition and translation of parenthesis grammars', J. ACM 24 (1977) 583-590.

[PS81] J.-E. Pin and H. Straubing, 'Monoids of upper triangular matrices', Colloq. Math. Soc. Janós Bolyai 39 (semigroups) 1981, 259-272.

[PS86] I. Parberry and G. Schnitger, 'Parallel computation with threshold functions', in *Structure in Complexity Theory*, Lecture Notes in Computer Science No. 223, (New York, Springer Verlag, 1986), 272-290.

[Ra86] A. A. Razborov, 'Lower bounds for the size of circuits of bounded depth with basis $\{\&, \oplus\}$', preprint (in Russian). To appear in Mathematical Notes of the Soviet Academy.

[Sc65] M. P. Schützenberger, 'On finite monoids having only trivial subgroups', Information and Control 8 (1965) 190-194.

[Si83] M. Sipser, 'Borel sets and circuit complexity', Proc. 15th ACM STOC, 1983, 61-69.

[Sm87] R. Smolensky, 'Algebraic methods in the theory of lower bounds for Boolean circuit complexity', Proc. 19th ACM STOC, 1987, 77-82.

[Sp71] P. M. Spira, 'On time-hardware complexity tradeoffs for Boolean functions', Proc. 4th Hawaii Symposium on System Sciences (North Hollywood, Calif., Western Periodicals Co., 1971) 525-527.

[St86] H. Straubing, 'Semigroups and languages of dot-depth 2', Proc. 13th ICALP, Lecture Notes in Computer Science, No. 226, (New York, Springer Verlag, 1986) 416-423.

[St86a] H. Straubing, 'Finite monoids and Boolean circuit complexity', draft.

[Th81] D. Thérien, 'Classification of finite monoids: the language approach', Theoretical Computer Science 14 (1981) 195-208.

[Th83] D. Thérien, 'Subword counting and nilpotent groups', in L. J. Cummings, ed., *Combinatorics on Words: Progress and Perspectives*, (Academic Press, New York, 1983), 297-305.

[Ya85] A. C. C. Yao, 'Separating the polynomial-time hierarchy by oracles', Proc. 26th IEEE FOCS, 1985, 1-10.

[Za58] H. J. Zassenhaus, *The Theory of Groups*, 2nd ed. (New York, Chelsea Publ. Co., 1958).