BOUNDED WIDTH POLYNOMINAL SIZE BRANCHING PROGRAMS RECOGNIZE EXACTLY THOSE LANGUAGES IN NC^1

David A. Mix Barrington
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 87-103

Bounded Width Polynomial Size Branching Programs Recognize Exactly Those Languages in NC^1

David A. Barrington[1]
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

April 2, 1987

1. Abstract

We show that any language recognized by an NC^1 circuit (fan-in 2, depth $O(\log n)$) can be recognized by a width-5 polynomial-size branching program. As any bounded-width polynomial-size branching program can be simulated by an NC^1 circuit, we have that the class of languages recognized by such programs is exactly non-uniform NC^1. Further, following Ruzzo [Ru81] and Cook [Co85], if the branching programs are restricted to be $ATIME(\log n)$-uniform, they recognize the same languages as do $ATIME(\log n)$-uniform NC^1 circuits, that is, those languages in $ATIME(\log n)$. We also extend the method of proof to investigate the complexity of the word problem for a fixed permutation group and show that polynomial size *circuits* of width 4 also recognize exactly non-uniform NC^1.

2. Definitions

Let $[w] = \{1, \ldots, w\}$. An *instruction* is a triple $\langle j, f, g\rangle$ in which j is the index of an input variable x_j and f and g are maps from $[w]$ to $[w]$. The meaning of an instruction $\langle j, f, g\rangle$ is "evaluate to f if $x_j = 1$, otherwise evaluate to g". A width-w branching program of length l (a w-BP) is a sequence of instructions $\langle j_i, f_i, g_i\rangle$ for $1 \leq i \leq l$. Given a 0-1 assignment x to the input variables $x_1, \ldots, x_n$, a

[1]This work is supported by NSF grant MCS-8304769, by US Air Force grant AFOSR-82-0326, and by an NSF Graduate Fellowship.

branching program B *yields* the function $B(\mathrm{x})$ which is the composition of the functions produced by each of the instructions. For example, let $w = 2$, let e be the identity function on w, and let (1 2) be the transposition of w's two elements. Then the program consisting of instructions

$$\langle 7, (1\ 2), e\rangle\ \langle 4, e, (1\ 2)\rangle\ \langle 8, (1\ 2), e\rangle$$

yields (1 2) if the exclusive OR of $x_7, \bar{x}_4$, and x_8 is one and yields e otherwise.

To recognize a language $L \subseteq \{0,1\}^*$ we need a family of width-w branching programs $\langle B_n\rangle$, where B_n for each positive integer n takes n Boolean inputs. Length and size can then be viewed as functions of n. We say that $\langle B_n\rangle$ recognizes L if for each n there is a set F_n of functions from $[w]$ to $[w]$, such that for all $\mathrm{x} \in \{0,1\}^n$, $\mathrm{x} \in L$ iff $B(\mathrm{x}) \in F_n$.

B is a *permutation branching program* (w-PBP) if each f_i and g_i is a permutation of $[w]$. In general we will use Greek letters for permutations.

The traditional notion (e.g., in [BDFP83]) of a width-w branching program is of a rectangular w by l array of nodes. Each node in row i is assigned an input variable and two out-edges leading to nodes in row $i+1$. Our model can be viewed this way but further requires that each node in a column be assigned the same variable. This requirement may be enforced at a cost of increasing the length polynomially and doubling the width. The traditional notion also defines the recognition of a set by a branching program in a different way, by fixing a start node (a source) and partitioning the sink nodes into accepting and rejecting. This difference may be dealt with in a similar manner. Thus both models define the same class $BWBP$ of languages recognized by bounded-width polynomial-size branching programs. Our main result is that $BWBP$ is equal to (non-uniform) NC^1.

In general we will speak of non-uniform circuits and branching programs, so that an NC^1 circuit will be simply a Boolean circuit with fan-in 2 and depth $O(\log n)$ (and hence polynomial size). Uniformity considerations will be postponed until Section 7, where we will make the appropriate definitions.

3. Previous Work

Branching programs were defined by Lee [Le59] as an alternative to Boolean circuits in the description of switching problems — he called them 'binary decision programs'. They were later studied in the Master's thesis of Masek [Ma76] under the name of 'decision graphs'.

Borodin, Dolev, Fich and Paul [BDFP83] and Chandra, Furst, and Lipton [CFL83] raised the question of the power of bounded-width branching programs. Borodin et al. noted that the class $BWBP$ contains AC^0 (languages recognized by unbounded fan-in, constant-depth, polynomial-size Boolean circuits) as well as the parity function (shown to be outside AC^0 in [FSS81] and [Aj83]). They conjectured that the majority function was not in $BWBP$, in fact that for bounded width it requires exponential length.

Subsequent results appeared which could be interpreted as progress toward proving this conjecture. Chandra, Furst, and Lipton [CFL83] and Púdlak [Pu84] showed superlinear length lower bounds for arbitrary constant width. In [BDFP83] the idea was to work with width-2 and get exponential bounds. They succeeded for a restricted class of BP's, and Yao [Ya83] followed with a superpolynomial lower bound for general width-2. Shearer [Sh85] proved an exponential lower bound for the mod-3 function with general width-2.

This lower bound program has continued after the preliminary publication of these results in [Ba86]. Ajtai et al. have proved a nearly $n \log n$ size lower bound for a large class of symmetric functions [ABHKPRST86], where width is allowed to be polynomial in $\log n$ but size is defined to be length times width. Alon and Maass [AM86] give an $n \log n$ size lower bound for many natural symmetric functions where width can be as great as $n^{1/2}$.

Barrington [Ba85] introduced the notion of width used here and considered width-3 permutation branching programs. Their power was characterized as equal to that of certain depth-2 circuits of mod-2 and mod-3 gates, and it was shown that these could recognize any set in exponential length and that exponential length was required to recognize a singleton set.

Here we show that since the majority function is in NC^1, it is in $BWBP$, and thus the conjecture of [BDFP83] is false. While polynomial-size 3-PBP's have very limited power, we show that polynomial-size 5-PBP's suffice to simulate NC^1 circuits.

4. The Main Result

We say that a 5-PBP B *five-cycle recognizes* a set $A \subseteq [2]^n$ if there exists a five-cycle σ (called the *output*) in the permutation group S_5 such that $B(\mathbf{x}) = \sigma$ if $\mathbf{x} \in A$ and $B(\mathbf{x}) = e$ if $\mathbf{x} \notin A$ (e is the identity permutation). For example, using the usual cycle notation for elements of S_5, the one-instruction program $\langle 1, (12345), e\rangle$ five-cycle recognizes the set $A = \{\mathbf{x} : x_1 = 1\}$ with output (12345).

Theorem 1: Let A be recognized by a depth d fan-in 2 Boolean circuit (we will assume that the circuit consists of AND and OR gates with a bottom level of inputs and negated inputs. Then A is five-cycle recognized by a 5-PBP B of length at most 4^d.

Lemma 1: If B five-cycle recognizes A with output σ and τ is any five-cycle, then there exists a 5-PBP B', of the same length as B, which five-cycle recognizes A with output τ.

Proof: Since σ and τ are both five-cycles there exists some permutation θ with $\tau = \theta\sigma\theta^{-1}$. To get B', simply change each instruction of B, replacing each σ_i and τ_i by $\theta\sigma_i\theta^{-1}$ and $\theta\tau_i\theta^{-1}$. □

Lemma 2: If A can be five-cycle recognized in length l, so can its complement.

Proof: Let B five-cycle recognize A with output σ. Call the last instruction of B $\langle i, \mu, \nu\rangle$. Let B' be identical to B except for last instruction $\langle i, \mu\sigma^{-1}, \nu\sigma^{-1}\rangle$. Then $B'(\mathbf{x}) = e$ if $\mathbf{x} \in A$ and $B'(\mathbf{x}) = \sigma^{-1}$ if $\mathbf{x} \notin A$. Thus B' five-cycle recognizes the complement of A. □

Lemma 3: There are two five-cycles σ_1 and σ_2 in S_5 whose commutator is a five-cycle. (The commutator of a and b is $aba^{-1}b^{-1}$.)

Proof: (12345)(13542)(54321)(24531) = (13254). □

Example: Consider the 5-PBP

$$\langle 1, (12345), e\rangle \ \langle 2, (13542), e\rangle \ \langle 1, (54321), e\rangle \ \langle 2, (24531), e\rangle$$

which yields (13254) as calculated above if $x_1 = x_2 = 1$. If either x_1 or x_2 is zero, the yield is e because of cancellation. So this 5-PBP five-cycle recognizes the set $\{\mathbf{x} : x_1 = x_2 = 1\}$. We now generalize this construction to prove our theorem.

Proof of Theorem 1: Let C be a depth d, fan-in 2 circuit of AND and OR gates, with a bottom level of inputs and negated inputs. We will show by induction on d that the set recognized by C can be five-cycle recognized by a 5-PBP. If $d = 0$ C has no gates and the set A can easily be recognized by a one-instruction 5-PBP. Assume without loss of generality that the output gate of C is an AND gate. If it is an OR gate, appeal to Lemma 2. Then the set A recognized by C is an intersection $A_1 \cap A_2$, where A_1 and A_2 are recognized by circuits of depth $d-1$ and thus are five-cycle recognized by B_1 and B_2 of length at most 4^{d-1}. Let B_1 and B_2 have the particular outputs σ_1 and σ_2 as defined in Lemma 3, and B_1' and

B_2' have outputs σ_1^{-1} and σ_2^{-1} (This last is possible by Lemma 1). Let B be the concatenation $B_1B_2B_1'B_2'$. B yields e unless the input is in both A_1 and A_2, but yields the commutator of the two outputs if the input is in A. This commutator is a five-cycle, and so B five-cycle recognizes A. B has length at most 4^d. Given a circuit and a desired output, this proof gives a deterministic method of constructing the 5-PBP. □

The following result is a non-uniform version of the well-known result that regular languages can be recognized by NC^1 circuits. The proof in the uniform case essentially appears in [Sa72] and is given explicitly in [LF77].

Theorem 2: If $A \subseteq [2]^n$ is recognized by a w-BP B of length l, A is recognized by a fan-in 2 circuit of depth $O(\log l)$, where the constant depends on w.

Proof: Recall our notion of recognition — we say that B accepts $\mathbf{x}$ if $B(\mathbf{x})$ is in some arbitrary subset of the functions from $[w]$ to $[w]$. We can represent such a function f by w^2 Boolean variables telling whether $f(i) = j$ for each i and j. The composition of two such functions so represented may be computed by a fixed circuit whose size depends only on w. Our circuit for A will have a constant-depth section to find the function yielded by each instruction of B, a binary tree of composition circuits, and a constant-depth section at the top to determine acceptance given the function yielded by B.□

Corollary: The classes of languages $BWBP$ and non-uniform NC^1 are identical. They equal the class of languages recognized by polynomial-length 5-PBP's.

5. Non-Uniform DFA's

The intuition that the width of a branching program corresponds to the number of possible configurations of an automaton appears to be wrong. We can formalize this notion to some extent. Define a *non-uniform DFA* (NUDFA) to be a k-state automaton with a two-way, read-only input tape and a one-way *program tape*. The latter contains instructions, where a single instruction is "move right" (on the input tape), "move left", or a state transition function from $[k] \times \{0, 1\}$ to $[k]$, which causes the automaton to enter a new state depending on the current state and the visible input character. NUDFA's with k states and k-BP's simulate each other — BP length corresponds to program tape length up to a multiplicative factor of $O(n)$.

One can define an NUNFA in the same way, by allowing the state transition function in an instruction to be nondeterministic, i.e., an arbitrary subset of $[k] \times$

$\{0,1\} \times [k]$. Then the familiar subset construction can be carried out showing that NUNFA's have the same power (for a given length) as NUDFA's (a 2^k-state NUDFA can simulate a k-state NUNFA). This shows, given the corresponding definition of nondeterministic branching programs, that nondeterministic $BWBP$ is also non-uniform NC^1. However, other definitions of nondeterministic branching programs are possible — see for example Meinel [Me86].

Our intuition must accept the fact that a 5-state NUDFA can count in polynomial time, as all symmetric functions are in NC^1. It can also divide, as by [BCH84] integer division is in non-uniform NC^1.

This notion is examined in much more detail by Barrington and Thérien in [BT87], where NUDFA's provide a link between well-studied classes of finite automata and subclasses of NC^1 defined in terms of Boolean circuits.

6. Boolean Circuits of Constant Width

We define width for Boolean circuits so as to allow nodes at any level to access the inputs without penalty, and examine the consequences of our main result for constant-width circuits in this model. It is easy to show [Ho83] that constant width for branching programs is equivalent to constant width for circuits, but here we go into more detail in an attempt to get the best possible simulations.

In particular, we show that width w branching programs (using the definitions of [Ba86]) can be simulated by circuits of width $\lceil \log w \rceil + 1$ and length multiplied by a constant depending only on w (this is a slight improvement of a result of Hoover [Ho83], who simulated width w BP's in the [BDFP83] model by circuits of width $\lceil \log w \rceil + 4$.) In particular, width 5 branching programs can be simulated by width 4 circuits (improving the result cited in [Jo86]), so that width 4 polynomial circuits can recognize all of NC^1 and thus everything recognized by circuits of constant width and polynomial size.

We choose the following definition of a width-w circuit from the many equivalent ones. A circuit is a rectangular array of nodes, consisting of l rows of w nodes each. Each node has one or two edges entering it which must be from either inputs or nodes on the immediately previous row. Possible node types are EQUALS (unary), NOT (unary), AND (binary), and OR (binary). The unary EQUALS node simply passes its input though unchanged, like a piece of wire, and is introduced so that each row will have the same number of nodes. Edges carry Boolean values, and nodes send out the appropriate value calculated from their input or inputs.

This is equivalent to other definitions which allow wires (edges) to jump over intermediate levels but count them as part of the width for those levels. (See, for example, [Jo86].) Perhaps the most natural first definition of width would charge for access to the inputs, but this would lead to a class far too restricted to be interesting.

Note that for defining the class of functions calculable using width w and length $O(f(n))$, we have a lot of latitude in our definitions. We will think of the inputs as being accessed by unary AND-x_i, AND-$\bar{x}_i$, OR-x_i, or OR-$\bar{x}_i$ gates — any other use of x_i can be simulated by these in a constant number of rows. We will also assume that only one input variable is accessed by a given row of nodes — this can be enforced by replacing one row by up to w rows.

Proposition: A Boolean circuit of width w and length l may be simulated by a branching program of width 2^w and length l.

Proof: Use the 2^w nodes in each instruction to represent the possible settings of the w Boolean variables on each level of the circuit. By our assumption, we access only a single input variable and thus the new state depends only on that variable and the old state. □

The simulation in the other direction is less straightforward. It is easy to simulate a w-BP by a $2w$-circuit, or even a $w+2$-circuit, by storing the branching program state in unary, i.e., in w gates exactly one of which will be on. We can improve matters by storing the state in binary.

Theorem 3: A branching program of width w and length l may be simulated by a Boolean circuit of width $\lceil \log w \rceil + 1$ and length $O(l)$, where the constant depends on w. That is, for any such branching program B there is such a circuit C which inputs a Boolean vector $\mathbf{x}$ and a state $s \in [w]$ and outputs $B(\mathbf{x})(s)$.

Proof: Without loss of generality let $w = 2^m$ be a power of two. To simulate an instruction $\langle j, f, g, \rangle$ it suffices to simulate one where either f or g is the identity, so without loss of generality we'll assume that $g = e$ and that the problem is to do f if x_j is on and the identity otherwise.

Note that we need only simulate a set of functions which generates under composition the entire set of functions from $[w]$ to $[w]$. (To simplify the proof we will renumber the elements of $[w]$ as $\{0 \ldots w-1\}$.)

Lemma: The functions from $[w]$ to $[w]$ are generated by: (1) the transpositions f_i, for $0 \leq i < m$, defined by $f_i(0) = 2^i$, $f_i(2^i) = 0$, and $f_i(j) = j$ otherwise; (2) the permutations g_i for $0 \leq i < m$ defined by $g_i(j) = j + 2^i$ for $j < 2^i$, $g_i(j) = j - 2^i$

for $2^i \leq j < 2^{i+1}$, and $g_i(j) = j$ otherwise; and (3) the function h defined by $h(0) = 0, h(1) = 0$, and $h(j) = j$ otherwise.

Proof: We will show that the f_i and g_i generate the permutations of $[w]$, by induction on m. This will suffice, as any function which is not one-to-one may easily be made up out of permutations and copies of h. For $w = 2$ the permutations of [w] are clearly generated by f_0. We must show how to generate any permutation of $[w] = [2^m]$, assuming that the f_i and g_i for $i < m - 1$ generate all permutations of $[w/2]$. By conjugation with g_{m-1}, we can make all permutations of the elements $\{w/2, \ldots, w-1\}$. Using these permutations as necessary among the high-numbered and low-numbered elements as necessary, we can use f_{m-1} to swap highs for lows as necessary to generate an arbitrary permutation of $[w]$. □

Proof of Theorem 3: (continued) We will encode the state by m bits L_0, L_1, ..., L_{m-1} with the state encoded being $s = \sum_i L_i 2^i$. For each of the functions f_i, g_i, and h, we must exhibit constant-width circuit sections which perform that function on s if x is on and leave s unchanged if x is off.

In the case of each f_i and g_i, we want to change L_i if necessary but leave all the other L_j unchanged. L_i must be changed to $L_i \oplus y$ for an appropriate y which is an AND of x and other L_j's. This is doable in constant width, using one extra node along with the first m, as follows. First compute $\bar{y}$ using successive ORs, maintaining all the L_i's. Then AND $\bar{y}$ with L_i and save the result. Now, using the space for L_i, compute $\bar{L}_i \wedge y$ by a NOT and successive ANDs. As $L_i \oplus y = (L_i \wedge \bar{y}) \vee (\bar{L}_i \wedge y)$, we can now get the new L_i with one OR step. The width-4 circuit in Figure 1 illustrates this method for the transposition f_2 or (0 4) with $m = 3$. In this example, the bit L_2 is to be changed iff the input x is on and both L_1 and L_0 are off. Here y is $x \wedge \bar{L}_1 \wedge \bar{L}_0$, and we calculate the new L_2 as $(L_2 \wedge \bar{y}) \vee (\bar{L}_2 \wedge y)$.

In the case of the function h, we want to change L_0 by the assignment $L_0 := L_0 \wedge y$, where y is the OR of $\bar{x}$ and all the other L_i's. The other L_i's are not changed. This is easily doable in width $m + 1$, by using one extra column to compute y by successive ORs and then ANDing it in at the end. □

Comparing this result with that of [Ho83], we see that our definition of BP width leads to a closer relationship between BP width and circuit width than does the [BDFP83] model. We conclude by summarizing the main consequence of Theorem 3 for bounded-width circuit complexity.

Corollary: The class of languages recognizable by circuits of constant width and polynomial size equals the class of those recognizable with width 4 and polynomial size, as both are NC^1.

Proof: Languages recognized by constant-width polynomial-size circuits are clearly in NC^1 by the Proposition above and Theorem 2. Any language in NC^1 is five-cycle recognized by a family of width-5 polynomial-size branching programs by Theorem 1. If we use Theorem 3 to simulate this branching program with any fixed input state, membership in L may be determined easily from the output.

7. Uniformity:

To speak of a uniform version of NC^1, it will be necessary to introduce alternating Turing machines, originally defined by Chandra, Kozen, and Stockmeyer [CKS81]. Here we will define a computation of an alternating Turing machine to be a game played by two players on a nondeterministic Turing machine which has two possible state transitions in every position. States are labelled White or Black as to which player has control of the moves from that state. For defining the class $ATIME(\log n)$, we assume that the machine has a random-access input tape which it can access only once, at the end of the computation, a worktape of size $c \log n$ for some constant c, and a clock which restricts it to running for $c \log n$ steps. The players, who are assumed to be omniscient, direct the computation of the machine until the end, when the machine reads an input bit (specified by an address written in binary on a work tape) and then enters a special "White victory" or "Black victory" state based on the value of this bit. The alternating Turing machine is said to accept an input **x** if White has a winning strategy for this game with input **x**. By standard methods these assumptions may be shown to be perfectly general.

The extended connection language of a fan-in 2 Boolean circuit consists of all strings of the form $\langle g, h, s\rangle$ where g and h are names of nodes in the circuit, $s \in \{left, right\}^{\leq \log n}$, and h is the node reached by following the path s from g. Ruzzo [Ru81] defines NC^1 circuits as those fan-in 2 depth $O(\log n)$ circuits whose extended connection language is in $ATIME(\log n)$. This has the consequence that $NC^1 = ATIME(\log n)$. We will define $ATIME(\log n)$-uniform branching programs in a natural way, and show that the class of languages recognized by $ATIME(\log n)$-uniform branching programs of polynomial size and constant width is also $ATIME(\log n)$. This will show that $BWBP = NC^1$ in the uniform as well as in the non-uniform setting.

Theorem 4: A language A is in $ATIME(\log n)$ iff it is recognized by a family of polynomial-size bounded-width branching programs B for which the language:

$$\{\langle k, i, f, g\rangle : \text{the } k\text{'th instruction of } B \text{ is } \langle i, f, g\rangle\}$$

is in $ATIME(\log n)$.

Proof: First we define a game in which White tries to prove that $B(\mathrm{x}) = f$, for some accepting f, and Black tries to refute him. At each stage of the game the log-time machine will define a range of instructions in B and a function which White claims is yielded by that range. White advances his claim by naming two functions g and h, with $f = gh$, and claiming that the first half of the range yields g and the second h. Black must choose one of these two subclaims to challenge, and this becomes White's new claim for the next stage. After $O(\log n)$ stages White will be making a claim about a single instruction, and this can be verified in $ATIME(\log n)$ by hypothesis. Each stage takes constant time, as we can let Black's sequence of choices be the index of the instruction to be checked — so each bit of this index need only be written down once.

For the converse, given a log-time machine M and game rules to make it an alternating machine, we can get an NC^1 circuit C in a standard way by creating a node for each configuration of M. Let B be the 5-PBP with output (12345), say, created from C by the method of Theorem 1 above, so that B five-cycle recognizes A. We must show that B is $ATIME(\log n)$ uniform. We define a game with input $\langle k, i, \sigma, \tau \rangle$ which White can win iff the input is a correct description of the k'th instruction. Both players, of course, know the actual circuit C and branching program B, as these are uniquely defined from M.

White at each stage will maintain a claim of the following form:

$$\langle s, \mu, k, i, \sigma, \tau \rangle$$

meaning "The subcircuit C_s of C whose top node is M-configuration s corresponds to a section B_s of B which five-cycle recognizes the language accepted by C_s with output μ. Further, the k'th instruction of B_s is $\langle i, \sigma, \tau \rangle$."

White will begin by claiming $\langle start, (12345), k, i, \sigma, \tau \rangle$ (where k, i, σ, and τ are taken from the input to the game) and refine this through $O(\log n)$ moves, each move corresponding to a step of M or to moving down one edge of C. For example, if s is an and-node B_s consists of four sections — White must state in which section the k'th instruction occurs, what its new number is, and which of s's children the section represents. Eventually s will be a final configuration of M and White's claim can be quickly decided. Black's moves during this process are to challenge any White claim which does not follow from his previous claim according to the definition of M and the procedure for creating B. Such a challenge may be decided easily in log-time, ending the game. White's moves are each only a constant number of steps if we choose an appropriate representation for the number k and don't have to rewrite it every time. □

8. Extensions and the Fine Structure of NC^1

What we have really done in Theorem 1 is to show that a certain problem is complete for NC^1 under certain reductions. The problem is to multiply together a series of elements of S_5, or equivalently to test whether a given word over S_5 is the identity. We will call this the *word problem* for S_5. Similarly we may define the word problem for any fixed group G. (We consider only finite groups, and always assume a group is represented as a permutation group.) This will correspond to the class of G-PBP's, where the permutations in each instruction must belong to G. Thus, for example, w-PBP's become S_w-PBP's.

Inside NC^1, it is most natural to define AC^0 reductions — the function f is reducible to g (written $f \leq_{AC^0} g$) if a constant-depth poly-size unbounded fan-in circuit, containing oracle nodes for g, can compute f. This notion was introduced in [FSS81] under the name of 'cp-reducibility'. They suggested further study of the degree structure (they had only just given the first proof that the structure of NC^1 was non-trivial) and conjected that majority was not reducible to parity.

This study was taken up by Fagin et al. in [FKPS84], who found many new AC^0 reducibilities among symmetric functions. Modulo the new parity lower bounds of Yao [Ya85] and Hastad [Ha86], they characterize those symmetric functions in AC^0. They show that the degree of the majority function is complete for symmetric functions and contains a large class of symmetric functions. Interestingly, no complete symmetric function exists in the projection-reducibility theory of Skyum and Valiant [SV81], by a recent result of Geréb-Graus and Szemerédi [GS86]. Chandra et al. [CSV84] prove several natural functions AC^0 equivalent to majority.

In this section we show that *solvability* of a group is the key to the applicability of the methods used earlier for the group S_5. We first give one of the many equivalent definitions (For more detail see a group theory text such as [Za58]). The commutator subgroup of G is the subgroup generated by all elements of the form $aba^{-1}b^{-1}$ for a and b in G. A group is solvable if and only if repeated taking of commutator subgroups eventually gives the trivial group. Thus a group is non-solvable if and only if it has a nontrivial subgroup whose commutator subgroup is itself. (All groups under discussion are finite.)

We first show that our earlier proof generalizes to any non-solvable group.

Theorem 5: The word problem for any fixed non-solvable group G is complete for NC^1 under AC^0 reductions.

Proof: Without loss of generality, assume that G's commutator subgroup is itself. We show that given a fan-in 2 circuit of depth d and an element a of G not equal to the identity, there is a G-PBP of length at most $(4g)^d$ which yields a if the circuit accepts the input and yields the identity otherwise. Here g is the order of G, a constant. Evaluating a G-PBP is easily seen to be in AC^0, given oracle nodes for the word problem for G. This will suffice to show completeness — the word problem is clearly in NC^1 as we can multiply two permutations in constant size and depth with fan-in two.

The proof, like that of Theorem 1, is by induction on d. The element a must have a representation as a product of at most g commutators. We carry out the proof of Theorem 1, except that we use the inductive hypothesis to produce G-PBP's yielding arbitrary non-identity elements of G instead of five-cycles. This multiplies the length by at most $4g$ instead of 4 at each step. Lemma 1 is unnecessary as for each d, we simultaneously prove the result for all a in G except the identity. □

It would be nice to have a converse to Theorem 5, but unfortunately we do not know enough about the AC^0-structure of NC^1 to prove one. By extending the methods of [Ba85], however, we can make a good start.

Theorem 6: The word problem for any fixed solvable group G is AC^0-reducible to the mod g function, where g is the order of G.

Proof: An equivalent definition of a solvable group (see, e.g., [Za58]) is one which has a series of normal subgroups $G = G_0, G_1, \ldots, G_m = \{e\}$ where each quotient group G_i/G_{i+1} is cyclic. We prove the theorem by induction on the length of this series. So assume that G has a normal subgroup N, where G/N is cyclic and the word problem for N is solvable by an AC^0 circuit containing mod g gates. Choose an element a such that the coset aN generates G/N.

We are given a product $g_1 \ldots g_k$ to evaluate. As N is normal, we can write each g_i uniquely as $a^{\epsilon_i} n_i$ with $n_i \in N$. (Converting between any two bit representations of an element of G takes constant size and depth.) Now let b_i be the product $a^{\epsilon_1} \ldots a^{\epsilon_i}$ and note that $a^{\epsilon_1} n_1 \ldots a^{\epsilon_k} n_k = (b_1 n_1 b_1^{-1}) \ldots (b_k n_k b_k^{-1}) b_k$. Each b_i depends only on the sum mod g of the appropriate ϵ_j, as the order of a in G divides g. Each term $b_i n_i b_i^{-1}$ is in N by normality, and we can calculate it in constant depth using mod g gates to get b_i. These partial terms may then be multiplied using a circuit for N. □

Theorem 6 is interesting only if the converse of Theorem 5 is true. To finish the argument, we would need to show that no single mod g function is powerful enough to be complete for NC^1. We conjecture that this is true, as it seems quite unlikely that a circuit of AND, OR, and MOD-g gates could do majority. This extends the conjecture of [FSS81] that the mod 2 function is not powerful enough to do majority. Further consequences of this conjecture are given in [BT87].

Unfortunately, the random restriction method of [FSS81] does not seem to extend even to parity (mod 2) gates, as the restriction of a parity gate is still a parity gate. However, there has recently been dramatic progress in this area. Razborov [Ra86] has proven the original conjecture of [FSS81] by showing that any constant depth circuit of AND and mod 2 gates computing the majority function has exponential size. Smolensky [Sm87] has extended this method to show that an AC^0 circuit with mod p gates cannot do the mod q function if p and q are distinct primes, and thus no circuit containing gates for a single prime can do majority. No limitations are yet known on the power of AC^0 circuits with mod q gates for composite q (except for prime powers, which are equivalent to their primes).

9. Open Problems and Recent Progress

We now know that poly-size bounded-width BP's give NC^1 while poly-size general BP's give L, the languages recognized by deterministic log-space Turing machines. Certainly this suggests a new attack on the problem of whether $NC^1 = L$ as this can now be phrased entirely in terms of branching programs. It would be useful to develop a lower-bound technology for width-5 PBP's, if this is possible. Even a superpolynomial lower bound for, say, the clique function would prove NC^1 different from NP.

The power of general poly-size permutation BP's (no restriction on width) was mentioned as an open problem in [Ba86a]. Cook and McKenzie [CM86] have just shown that the word problem for S_n is complete for log space under NC^1 reductions, even if the inputs and outputs are in pointwise notation (i.e., a permutation σ is given as the list of integers $\sigma(1), \ldots, \sigma(n)$). (In fact, they show that the easier problem of permutation powering with the exponent in unary is complete.) A poly-size PBP can be constructed to solve this problem, given an appropriate definition of recognition of a language by a PBP. As these PBP's can be thought of as *reversible* non-uniform log-space Turing machine computations, this suggests a comparison with work of Bennett [Be73].

The effect of non-determinism on these classes must be examined as well, suggesting possible new attacks on the problem of whether L is equal to NL, the class of languages recognized by nondeterministic log-space Turing machines. One must be careful with definitions here, as the wrong sort of non-determinism can turn a very small class into NP. For example, depth-2 poly-size unbounded fan-in Boolean circuits can only recognize Π_2-$TIME(\log n)$. But if we give such a circuit both x and y inputs and say that it 'accepts' x iff there is some y such that the circuit accepts $\langle x, y \rangle$, it can recognize any language in NP. This and similar extensions of the branching program model have recently been considered by Meinel [Me86].

We know the power of width-3 [Ba85] and width-5 PBP's — what of width-4? As S_4 is solvable, they cannot do all of NC^1 by the method used here for width-5, but we would like to prove they cannot do it at all. The conjecture of Section 8 would settle this, but 4-PBP's are a special case which might be more amenable to analysis.

Width 2 and 3 Boolean circuits are an attractive target for a lower bound proof — it would be nice to show that width 4 is necessary to do NC^1, if this is true. Can one improve Theorem 3 on simulating BP's by circuits? Of course, any bounded width BP can be simulated in width 4 with polynomial size blowup using our main result, but can the simulation be improved keeping the blowup linear?

We know that BP's without the permutation restriction require width-3 to do majority in poly-size [Ya83] and we know that width-5 suffices. Does the extra freedom to use non-permutation instructions help at all? This amounts to extending the notion of a PBP over a group to a BP over a monoid, and is taken up in [BT87]. There it is shown, modulo the conjecture of Section 8, that poly-size BP's over a monoid which contains only solvable groups cannot recognize all of NC^1, so that width 5 is necessary here as well.

The fine structure of NC^1 is another good subject for further study. Until recently we knew only that there are at least two classes under AC^0 reductions (from [FSS81] and [Aj83]) but this is more than is known about most degree theories in complexity theory. The recent work of Razborov [Ra86] and Smolensky [Sm87] has shown that there are infinitely many degrees (see the last section) and tends to support the conjecture of Section 8, but this conjecture is still open.

There seems to be no reason to believe that majority is complete for NC^1, but we are a long way from proving this. The languages AC^0 reducible to majority form the analogue of AC^0 in the threshold gate theory of Parberry and Schitger, and might be an interesting proper subclass of NC^1. Many natural problems are known to be AC^0 equivalent to majority, as shown by Fagin et al. [FKPS85] and Chandra et al. [CSV84].

AC^0-reducibility should also be compared with the projection reducibility of Skyum and Valiant [SV81] in this setting. Majority is AC^0-complete for symmetric functions, but no function is projection-complete for them [GS86].

It is also interesting that an algebraically-defined regular language such as a word problem should be complete for NC^1. The analysis in [BT87] sheds some further light on this.

The unexpected power of NUDFA's suggests some foundational questions. Placing the power to recognize a language in a program to a very simple machine seems very different than placing it in, say, the state table of a Turing machine. How different is it, and how does it relate to other known models of computation?

10. Acknowledgements

This work is part of my Ph.D. research under the direction of Mike Sipser, who suggested this topic and advised me throughout my work on it. I would like to thank him for all his help and also thank Ravi Boppana, Johan Hastad, David Johnson, Philip Klein, Tom Leighton, and Leslie Valiant for various helpful discussions. I am also grateful for a number of helpful suggestions made by the anonymous referees.

This paper is a slightly revised version of [Ba86], which is copyrighted by the Association for Computing Machinery, and the material of [Ba86] appears with their permission. Most of this material has also appeared in my Ph.D. thesis [Ba86a], which is copyrighted by the Massachusetts Institute of Technology and used with their permission. I would like to thank the M.I.T. Mathematics Department and Laboratory for Computer Science for all of their support.

11. References

[ABHKPRST86] M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Púdlak, V. Rödl, E. Szemerédi, and G. Turán, 'Two lower bounds for branching programs', Proc. 18th ACM STOC, 1986, 30-38.

[Aj83] M. Ajtai, 'Σ_1^1 formulae on finite structures', Annals of Pure and Applied Logic 24 (1983), 1-48.

[AM86] N. Alon and W. Maass, 'Meanders, Ramsey theory, and lower bounds for branching programs', Proc. 27th IEEE FOCS, 1986, 410-417.

[Ba85] D. A. Barrington, 'Width-3 permutation branching programs', Technical Memorandum TM-293, M.I.T. Laboratory for Computer Science.

[Ba86] D. A. Barrington, 'Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1', Proc. 18th ACM STOC, 1986, 1-5.

[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.

[BDFP83] A. Borodin, D. Dolev, F. E. Fich, and W. Paul, 'Bounds for width two branching programs', Proc. 15th ACM STOC, 1983, 87-93. Journal version SIAM J. Computing 15 (1986) 549-560.

[Be73] C. H. Bennett, 'Logical reversibility of computation', IBM Journal of Research and Development, 17 (1973), 525-532.

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

[CFL83] A. K. Chandra, M. L. Furst, and R. J. Lipton, 'Multiparty protocols', Proc. 15th ACM STOC, 1983, 94-99.

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, 'Alternation', J. ACM 28 (1981) 114-133.

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

[CM86] S. A. Cook and P. McKenzie, 'Problems complete for deterministic logarithmic space', Publication 560 (Fév. 1986), Dépt. d'I.R.O., Université de Montréal.

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

[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.

[GS86] M. Geréb-Graus and E. Szemerédi, 'There are no p-complete families of symmetric Boolean functions', preprint, 1986.

[Ha86] J. Hastad, 'Almost optimal lower bounds for small depth circuits', Proc. 18th ACM STOC, 1986, 6-20.

[Ho83] H. J. Hoover 'Characterizing bounded width', manuscript, 1983.

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

[LF77] R. E. Ladner and M. J. Fischer, 'Parallel prefix computation', Proc. 1977 Intl. Conf. on Parallel Processing, 218-233. Journal version J. ACM 27 (1980) 831-838.

[Le59] C. Y. Lee, 'Representation of switching functions by binary decision programs', Bell System Technical Journal 38 (1959) 985-999.

[Ma76] W. Masek, 'A fast algorithm for the string editing problem and decision graph complexity', M.Sc. thesis, Dept. of E.E.C.S., M.I.T., May 1976.

[Me86] C. Meinel, 'Rudiments of a branching program based complexity theory', preprint, June 1986.

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

[Pu84] P. Púdlak, 'A lower bound on complexity of branching programs', Proc. Conference on the Mathematical Foundations of Computer Science, 1984, 480-489.

[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 Academy of Sciences of the USSR.

[Ru81] W. L. Ruzzo, 'On uniform circuit complexity', JCSS 22, 3 (June 1981), 365-383.

[Sa72] J. Savage, 'Computational work and time on finite machines', J. ACM 19 (1972), 660-674.

[Sh85] J. B. Shearer, personal communication, 1985.

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

[SV81] S. Skyum and L. G. Valiant, 'A complexity theory based on Boolean algebra', Proc. 22nd IEEE FOCS, 1981, 244-253. Journal version J. ACM 32 (1985) 484-502.

[Ya83] A. C. Yao, 'Lower bounds by probabilistic arguments', Proc. 24th IEEE FOCS, 1983, 420-428.

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

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

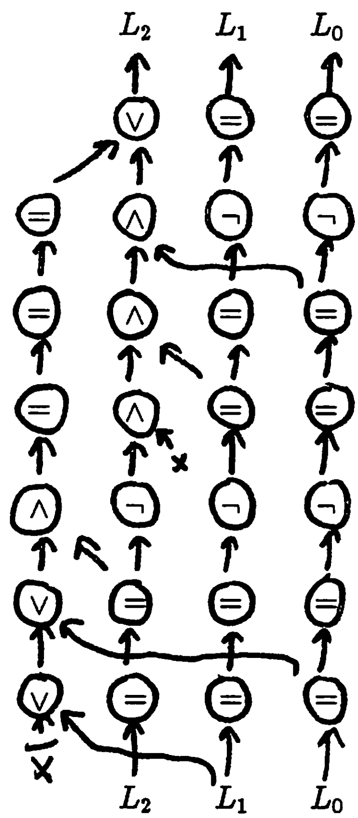

Figure 1: A width 4 circuit calculating f_2.