ON NC^1 LANGUAGE RECOGNITION

by

Elizabeth McCarthy

B. Sc., University of New Mexico, 1985

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR

MASTER OF SCIENCE

in the Department of

Mathematics and Statistics

SIMON FRASER UNIVERSITY

December, 1989

APPROVAL

Name:	Elizabeth McCarthy
Degree:	Master of Science
Title of thesis:	On NC^1 Language Recognition

Examining Committee:

Chairperson: Dr. A.H. Lachlan

Dr. N.R. Reilly
Senior Supervisor

Dr. A.R. Freedman

Dr. A. Mekler

Dr. K. Heinrich
External Examiner

Date approved: December 6, 1989

ABSTRACT

The objective of this thesis is to give a self-contained account of some recent work in automata theory and complexity theory. This account will focus primarily on David A. Barrington's work on bounded width branching programs and the parallel complexity class NC^1. Preliminary topics necessary for a full understanding of the new work, including languages, monoids, and Boolean circuits, are presented, a detailed reconstruction of the proof of Barrington's main result is given, and related results are discussed.

ACKNOWLEDGEMENTS

Many thanks to Dr. Norman Reilly for being my senior supervisor, and for suggesting a thesis topic which couldn't have been more to my liking. Thanks to my committee members for their helpful comments. Thanks to Sylvia for doing everything which didn't absolutely have to be done by me. Thanks to Professors T. Evans and M. Neff of Emory University for helpful courses and/or seminars while I was away from SFU. Finally, thanks to ".ANN" and thanks and all my love to Luca.

TABLE OF CONTENTS

Introduction

The objective of this thesis is to give a self-contained account of some recent work in automata theory and complexity theory. The primary focus is the work of David A. Barrington on bounded width branching programs and the complexity class NC^1. A detailed reconstruction of his main result is given, and related results are also discussed.

Chapter I is divided into two sections. In the first section, we set up some notation, and recall some fundamental definitions and results concerning semigroups and languages, which will be used in the sequel. In the second section, we recall the definitions of a computable function and a recognizable language, and present an informal discussion of some ideas in complexity theory, to provide a context for later material.

Chapter II is devoted to a study of three non-uniform models of computation. In the first section we discuss the Boolean circuit family as a model of computation. In the second section, we discuss the family of branching programs as a model for language recognition, and define a complexity class, BWBP, of languages based on the model. In the final section of the chapter, we discuss the family of programs for a non-uniform deterministic finite automaton (NUDFA) as a model for language recognition, and define a complexity class, PLP, of languages based on the model. We show that PLP contains the regular languages.

In Chapter III, we establish numerous connections between the computational models discussed in Chapter II. In the first section we discuss the concept of uniformity which provides a "bridge" between non-uniform models of computation, such as those studied in Chapter Two, and uniform models of computation, such as the Turing machine. We then define the parallel complexity class NC^1 in terms of Boolean circuit families.

In section two, we present David A. Barrington's version of the branching program model, and show that, given a certain choice of definition for the language recognized by one of Barrington's branching programs, the complexity class CWBP of languages defined in terms of his modified branching program model is the same as the complexity class BWBP of languages defined in terms of the branching program model discussed in Chapter II.

In section three, we show that, given a different choice of definition for the language recognized by one of Barrington's branching programs, the complexity class CWBP of languages defined in terms of his branching program model is the same as the complexity class PLP of languages defined in terms of programs for a NUDFA over a finite monoid, as discussed in Chapter II.

In the fourth and final section, we present a detailed reconstruction of the proof of Barrington's main result that the complexity class BWBP of languages is exactly (non-uniform) NC^1.

We conclude with a discussion of some of the consequences of Barrington's result, make some further observations, and briefly summarize some further developments in this area.

Chapter I Preliminaries

In this chapter we briefly review some basic definitions and fundamental results in the theories of semigroups, languages, decidability, computability, and complexity, from which the work we will later examine in detail, evolved.

§ 1.1 *Algebraic and Linguistic Preliminaries*

We assume a knowledge of the definitions of and basic results about groups (see, e.g., [14]), semigroups, and monoids (see, e.g., [18]), and a familiarity with homomorphisms and congruences on these algebras. We also assume some knowledge of automata theory, including finite state machines and regular languages ([13]).

Definition 1.1.1. Let $w \in \mathbb{Z}^+$. The set $\{1,2,\ldots,w\}$ is denoted $[w]$. The transformation monoid, or the monoid of all functions from $[w]$ to $[w]$, is denoted M_w. A subsemigroup of a transformation monoid is called a transformation semigroup. The symmetric group, or the group of all permutations of $[w]$, is denoted S_w. We write $g \circ f$ to denote the composition of f and g, where $(g \circ f)(x) = g(f(x))$.

Proposition 1.1.2. *Every semigroup is isomorphic to a transformation semigroup. In particular, if M is a finite monoid of cardinality w, denoted $c(M) = w$, then M is isomorphic to a submonoid of M_w.*

The following definitions and results may be found in Eilenberg ([13], vol. A).

Definition 1.1.3. Let $n \in \mathbb{Z}^+$, and let $\Sigma = \{\sigma_1,\sigma_2,\ldots,\sigma_n\}$. Then Σ is called an alphabet, each σ_i is called a letter, and $u = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_k}$, where each $\sigma_{i_j} \in \Sigma$, is called a word over the alphabet Σ of length k, denoted $|u| = k$. For each $n \in \mathbb{N}$, we write Σ^n to denote the set of all words over Σ of length n. The unique word over Σ of length 0 is denoted ε. Σ^* denotes the set of all words over the alphabet Σ, which is, together with the operation of concatenation, the free monoid generated by the set Σ.

Definition 1.1.4. Let Σ be a finite alphabet. A subset $L \subseteq \Sigma^*$ is called a language over Σ.

Although there are other characterizations for it, and a great deal is known about the class of regular languages (see [13], [19]), the following is sufficient for our purposes.

Definition 1.1.5. Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is said to be regular if there is a finite monoid M, a subset B of M, and a homomorphism $\phi : \Sigma^* \rightarrow M$ such that $L = \phi^{-1}(B)$.

Proposition 1.1.6. *Let Σ and Γ be finite alphabets, let L be a regular language over Σ and let $\phi : \Sigma^* \to \Gamma^*$ be a homomorphism such that $\phi^{-1}(\varepsilon) = \varepsilon$. Then $\phi(L)$ is a regular language over Γ.*

Definition 1.1.7. Let Σ and Γ be finite alphabets, and let f be an injective function from Σ^* to Γ^*. The function f is said to be a _coding_. For each $u \in \Sigma^*$, the element $f(u) \in \Gamma^*$ is called the _encoding_ of u, and u is called the _decoding_ of the element $f(u) \in \Gamma^*$.

§ 1.2 *Decidability, Computability, and Complexity*

In this section we recall the definitions of a computable function and a recognizable language, and briefly and informally discuss some ideas in complexity theory. Knowledge of the definition of, and basic facts about the Turing machine is assumed (see [17]).

Definition 1.2.1. Let Σ and Γ be finite alphabets. A function $f : \Sigma^* \to \Gamma^*$ is said to be _computable_ if there is a Turing machine which, for each input $u \in \Sigma^*$, will halt and output f(u). A partial function $f : \Sigma^* \to \Gamma^*$ is said to be computable if there is a Turing machine which will, for each $u \in \Sigma^*$ such that f(u) is defined, halt and output f(u).

Definition 1.2.2. Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is said to be _recursive_, or _recognizable_ if there is a Turing machine, whose states are partitioned into "accepting" and "rejecting" states, which halts on every input word $u \in \Sigma^*$, and which halts in an "accepting" state if and only if $u \in L$. A language $L \subseteq \Sigma^*$ is said to be _recursively enumerable_ if there is a Turing machine which will halt in an accepting state for each $u \in \Sigma^*$, and which either halts in a rejecting state or does not halt if $u \notin L$.

We recall that we may, without loss of generality, restrict our attention to functions on $\{0,1\}^*$ and languages over $\{0,1\}$, since every finite alphabet Σ can be encoded into words over $\{0,1\}$ in such a way that a language over Σ is recognizable if and only if the encoded language over $\{0,1\}$ is recognizable. There is an analogous result for functions $f : \Sigma^* \to \Gamma^*$ (see [17]).

Once it had been established just what it means for a problem to be solvable, it seems natural to ask, of a solvable problem, "how hard is it to solve?," or, "how much time and/or space is required for its solution?" These are some of the questions which concern complexity theorists (see [12]). Given a model of computation, such as the Turing machine, we first define the "resources" of the model. The "resources" of a Turing machine include time and space. Once a Turing machine has been constructed which computes a particular function, the time the Turing machine takes to compute the function is an upper bound on the time complexity of the function, the space required to compute the function is an upper bound on the space complexity of the function, and the time and space required to compute the function is an upper bound on the simultaneous time and space complexity of the function. If a Turing machine has been constructed which takes time T to compute a function f, and it is proven that f cannot be computed by any Turing machine, in less time than T, then the function is said to have time complexity T. The space complexity and simultaneous time and space complexity of the function f are defined in an analogous manner.

Anyone who has ever attempted to construct a Turing machine which computes even a very simple function can appreciate how difficult it must be to prove that one has constructed an "optimal" Turing machine to compute a function. Complexity theorists look to other computational models for which it may be easier to prove "lower bounds" on the complexity of functions, which can then be translated into lower bounds on the complexity of the functions as based on the standard Turing machine model ([16]). In the next chapter we will study several computational models now under consideration in this context (see, e.g., [10], [16], [24], [15]).

Chapter II Non-Uniform Models of Computation

In this chapter, we study three different *non-uniform* models of computation. These models are called non-uniform because, in contrast to more familiar models of computation, such as the Turing machine, where one Turing machine works on input words of arbitrary length, these models consist of families of "components", one component for each input length, which processes input words of one length only. Because of their "non-uniformity", it is difficult to see how these models compare with models such as the Turing machine (see [20]). This "difficulty" will be addressed in Chapter three.

In section one, we look at Boolean circuit families. In section two, we look at families of branching programs. In the third and final section of this chapter we study families of programs for a non-uniform deterministic finite automaton over a finite monoid.

§ 2.1 *The Boolean Circuit Family Model of Computation*

We begin the section with a discussion of Boolean circuits, the components of the computational model by which the parallel complexity classes NC, NC^k, and AC^k will eventually be defined. A Boolean function is a function from $\{0,1\}^n$ to $\{0,1\}$, where $n \in \mathbb{Z}^+$. Let $\Omega = \{\wedge, \vee, \neg\}$ be the set of Boolean functions defined by the table below.

x	y	$x \wedge y$	$x \vee y$	$\neg x$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

We review some facts concerning Boolean functions and Boolean formulas, all of which can be found in, e.g., [26] or [23].

Definition 2.1.1. Let $X = \{x_1, x_2, \ldots, x_n, \ldots\}$ be a set whose elements are called variables. A Boolean formula is defined inductively as follows:

(*i*) x_j is a Boolean formula for all $j \in \mathbb{Z}^+$.

(ii) If α and β are Boolean formulas, then $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, and $(\neg\alpha)$ are Boolean formulas.

Since the functions $\wedge$ (and) and $\vee$ (or) are associative, we will often write $\bigwedge_{i=1}^{n} x_i$ to denote the Boolean formula $((\ldots((x_1 \wedge x_2) \wedge x_3) \wedge \ldots) \wedge x_n)$, and the case $\vee$ will be handled similarly. Let $n \in \mathbb{Z}^+$, and let $\beta(x_1, x_2, \ldots, x_n)$ be a Boolean formula containing the variables

$\{x_1,x_2,\ldots,x_n\}$. The Boolean formula β defines a (Boolean) function from $\{0,1\}^n$ to $\{0,1\}$ in the following way. For each $\mathrm{a} = \mathrm{a}_1\mathrm{a}_2\cdots\mathrm{a}_n \in \{0,1\}^n$, the value $\beta(\mathrm{a})$ is obtained by replacing for each $i \in \{1,2,\ldots,n\}$ every occurrence of the variable x_i in β with a_i, and evaluating the resulting expression according to the rules for $\wedge$, $\vee$, and $\neg$, given in the table above.

<u>Lemma</u> <u>2.1.2</u> [26]. *Let* $n \in \mathbb{Z}^+$, *and let* $f\colon \{0,1\}^n \to \{0,1\}$ *be a Boolean function. Then there is a Boolean formula* $\beta(x_1,x_2,\ldots,x_n)$ *such that* $f(a) = \beta(a)$ *for all* $a \in \{0,1\}^n$.
<u>*Proof.*</u> Let $\mathrm{a}^1,\mathrm{a}^2,\ldots,\mathrm{a}^k \in \{0,1\}^n$ be such that $\mathrm{f(a)} = 1$ if and only if $\mathrm{a} \in \{\mathrm{a}^1,\mathrm{a}^2,\ldots,\mathrm{a}^k\}$, where $1 \le k \le 2^n$. For each $j \in \{1,2,\ldots,k\}$, $\mathrm{a}^j = \mathrm{a}_1^j\mathrm{a}_2^j\ldots\mathrm{a}_n^j$, let $\beta_j(x_1,x_2,\ldots,x_n)$ be the Boolean formula $\bigwedge_{r=1}^{n} y_r$ where $y_r = x_r$ if $\mathrm{a}_r^j = 1$, and $y_r = (\neg x_r)$ if $\mathrm{a}_r^j = 0$. Let $\beta(x_1,x_2,\ldots,x_n)$ be the Boolean formula $\bigvee_{t=1}^{k} \beta_t(x_1,x_2,\ldots,x_n)$. Then for $\mathrm{b} \in \{0,1\}^n$, $\beta(\mathrm{b}) = 1$ if and only if for some $t \in \{1,2,\ldots k\}$, $\beta_t(\mathrm{b}) = 1$, that is, if and only if for all $r \in \{1,2,\ldots,n\}$, $y_r(\mathrm{b}) = 1$. By definition of β_t, $y_r = x_r$ if $\mathrm{a}_r^t = 1$, and $y_r = (\neg x_r)$ if $\mathrm{a}_r^t = 0$, so that $y_r(\mathrm{b}) = 1$ if and only if $\mathrm{b}_r = \mathrm{a}_r^t$. Therefore, $\beta_t(\mathrm{b}) = 1$ if and only if $\mathrm{b}_r = \mathrm{a}_r^t$ for all $r \in \{1,2,\ldots,n\}$, if and only if $\mathrm{b} = \mathrm{a}^t$, and $\beta(\mathrm{b}) = 1$ if and only if $\mathrm{b} \in \{\mathrm{a}^1,\mathrm{a}^2,\ldots,\mathrm{a}^k\}$, that is, if and only if $\mathrm{f(b)} = 1$. Therefore, β is the desired Boolean formula. □

Let m, $n \in \mathbb{Z}^+$, and let f be a function from $\{0,1\}^n$ to $\{0,1\}^m$. For each $i \in \{1,2,\ldots,m\}$, we define a Boolean function $\mathrm{f}_i : \{0,1\}^n \to \{0,1\}$ by $\mathrm{f}_i(\mathrm{a}) = (\mathrm{f(a)})_i$, the i-th letter of $\mathrm{f(a)}$, for all $\mathrm{a} \in \{0,1\}^n$. Then, for all $\mathrm{a} \in \{0,1\}^n$, $\mathrm{f(a)} = \mathrm{f}_1(\mathrm{a})\mathrm{f}_2(\mathrm{a})\cdots\mathrm{f}_m(\mathrm{a})$. This observation and Lemma 2.1.2 give us the next corollary.

<u>Corollary</u> <u>2.1.3</u>. *Let* m, $n \in \mathbb{Z}^+$, *and let* f *be a function from* $\{0,1\}^n$ *to* $\{0,1\}^m$. *Then there are Boolean formulas* $\beta_1(x_1,x_2,\ldots,x_n)$, $\beta_2(x_1,x_2,\ldots,x_n)$, $\ldots$, $\beta_m(x_1,x_2,\ldots,x_n)$ *such that for all* $a \in \{0,1\}^n$, $f(a) = \beta_1(a)\beta_2(a)\cdots\beta_m(a)$.
<u>*Proof.*</u> Direct application of Lemma 2.1.2 to the m functions f_i defined as in the discussion above. □

<u>*Definition*</u> <u>2.1.4</u>. Let $\Gamma = \{0, 1, \wedge, \vee, \neg\}$, where 0 and 1 are the constant Boolean functions, with values 0 and 1 respectively, and $\wedge$, $\vee$, and $\neg$ are as defined in the table. An <u>extended</u> <u>Boolean</u> <u>formula</u> is defined inductively as follows:

(*i*) 1, 0 and x_j are extended Boolean formulas for all $j \in \mathbb{Z}^+$.

(*ii*) If α and β are extended Boolean formulas, then $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, and $(\neg\alpha)$ are extended Boolean formulas.

<u>*Definition*</u> <u>2.1.5</u>. An <u>acyclic</u>, <u>directed</u> <u>graph</u> is a set of <u>nodes</u> together with a set of <u>arcs</u>, each arc connecting two nodes which satisfies the following:

(i) each arc is an arrow directed from one node into another; and

(ii) each arc connects two distinct nodes, and any path traced along the arcs following the direction of the arrows through the nodes will pass through a node of the graph at most once.

Definition 2.1.6. The indegree of a node in a directed graph is the number of arcs directed into the node. The outdegree of a node in a directed graph is the number of arcs directed out of the node. A node with indegree zero is called a source, and a node with outdegree zero is called a sink.

Definition 2.1.7 (see [16], [11]). A Boolean circuit on n variables is a finite, nonempty, acyclic, directed graph with nodes called gates and arcs called edges which satisfies the following.

(i) There are $k \geq n$ sources. For each i, $1 \leq i \leq n$, there is a source labelled by the variable x_i. Other sources, if any, carry one of the labels 0 and 1. The n sources which carry the variables x_i as labels are called input gates.

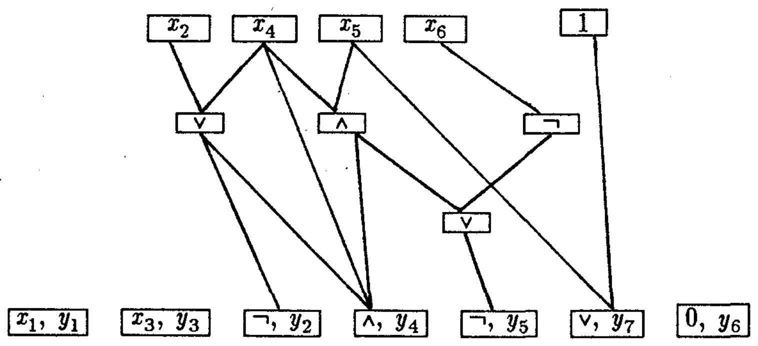
(ii) There are m sinks called output gates, and for each j, $1 \leq j \leq m$, there is an output gate labeled by the variable y_j.

(iii) All gates of indegree 1 carry the label $\neg$.

(iv) All other gates are of indegree greater than or equal to two, and each of these gates carries one of the labels $\wedge$, $\vee$.

Definition 2.1.8. The size of a Boolean circuit is the number of gates in the circuit. The depth of a Boolean circuit is the length of the longest path (i.e., greatest number of consecutive edges) from an input gate of the circuit to an output gate of the circuit. The indegree of the gate of largest indegree is the fan-in of the Boolean circuit.

Example 2.1.9. Consider the Boolean circuit below, where by convention all edges are directed down.



This Boolean circuit has 8 sources, 6 input gates, and 7 sinks or output gates. Its size is 16, its depth is 3, and it has fan-in 3.

We now describe how a Boolean circuit B with n input gates and m output gates computes a function from $\{0,1\}^n$ to $\{0,1\}^m$, where $m, n \in \mathbb{Z}^+$.

Definition 2.1.10. Let C be a Boolean circuit, and let g be a gate in C. The rank of g is the length of the longest path from a source in C to g. Sources have rank 0.

Definition 2.1.11. Let C be a Boolean circuit with n input gates. Let $a = a_1a_2\cdots a_n \in \{0,1\}^n$. For each gate g in C, the value $v_a(g)$ of the gate g on input a is defined inductively as follows:

(*i*) If g has rank 0, then $v_a(g) = a_i$ if the label of g is x_i, $v_a(g) = 1$ if the label of g is 1 and $v_a(g) = 0$ if the label of g is 0.

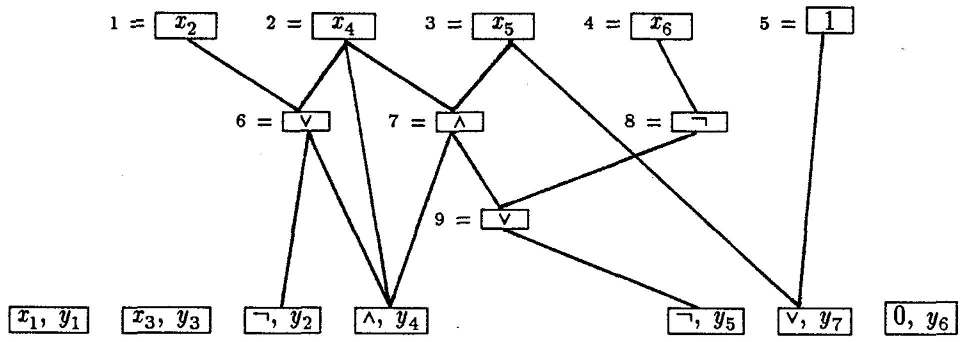
(*ii*) If g has rank k and $v_a(h)$ has been defined for each gate h in C of smaller rank, then if the label of g is $\vee$ and $h_1, h_2, \ldots, h_t$ are gates from which there are edges entering g, then $v_a(g) = v_a(h_1) \vee v_a(h_2) \vee \cdots \vee v_a(h_t)$. Similarly, if the label of g is $\wedge$, then $v_a(g) = v_a(h_1) \wedge v_a(h_2) \wedge \cdots \wedge v_a(h_t)$. If the label of g is $\neg$, then $v_a(g) = \neg v_a(h_1)$.

Definition 2.1.12. Let C be a Boolean circuit with n input gates and m output gates $o_1, o_2, \ldots, o_m$. Let $f : \{0,1\}^n \rightarrow \{0,1\}^m$ be the function defined by $f(a) = v_a(o_1)v_a(o_2)\cdots v_a(o_m)$. We call f the function computed by the Boolean circuit C. The value of the function computed by C on input a is denoted C(a).

Note: If $m = 1$, then the set $\{u \in \{0,1\}^* : B(u) = 1\}$ is called the language recognized by B. Also, we will omit the labels y_i on output gates whenever we have constructed a circuit for which a description such as "the i-th output gate from the left" makes sense. In such a case, "the i-th output gate from the left", is understood to be the output gate labelled y_i. All edges are directed downward.

If B is a Boolean circuit with n input gates and one output gate, then B computes a Boolean function. So by Lemma 2.1.2, there is a Boolean formula which corresponds to the Boolean function computed by the circuit B. If the circuit B has more than one output gate, then for each output gate G_i (labelled y_i) of B, consider the subcircuit of B consisting of G_i and all gates "above" it which lie along some path starting at G_i and traced backward along the edges of B. This subcircuit of B has $j \leq n$ input gates and one output gate G_i, and therefore computes a Boolean function. Again by Lemma 2.1.2, there is an extended Boolean formula which corresponds to the Boolean function computed by the subcircuit of B with output gate G_i. It follows from Corollary 2.1.3 that for each Boolean circuit B with n input gates and m output gates, there is a sequence of m extended Boolean formulas which corresponds to the function from $\{0,1\}^n$ to $\{0,1\}^m$ computed by the Boolean circuit B. We now provide an example to illustrate this verbal description of a method ([23], [26]) to obtain extended Boolean formulas which correspond to a given Boolean circuit.

Example 2.1.13. We first assign a number to each gate of the circuit of Example 2.1.9 which is not an output gate.



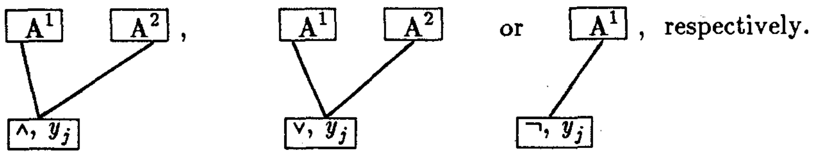
Then, $y_1 = x_1$, $y_2 = (\neg 6) = (\neg(1 \vee 2)) = (\neg(x_2 \vee x_4))$, $y_3 = x_3$, $y_6 = 0$, $y_7 = (3 \vee 5) = (x_5 \vee 1)$, $y_4 = (6 \wedge 2 \wedge 7) = ((x_2 \vee x_4)) \wedge x_4 \wedge (x_4 \wedge 3) = ((x_2 \vee x_4)) \wedge x_4 \wedge (x_4 \wedge x_5)$, and $y_5 = (\neg 9) = (\neg(7 \vee 8)) = (\neg((x_4 \wedge x_5) \vee (\neg 4))) = (\neg((x_4 \wedge x_5) \vee (\neg x_6)))$. Let $\beta_i = y_i$ for $1 \leq i \leq 7$. Then $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7$ is the desired sequence of extended Boolean formulas corresponding to the given Boolean circuit.

Example 2.1.14. From the above, we see that the Boolean circuit of Example 2.1.9 computes the function $f: \{0,1\}^6 \to \{0,1\}^7$ defined by $f(a) = \beta_1(a)\beta_2(a)\cdots\beta_7(a)$ for all $a \in \{0,1\}^n$.

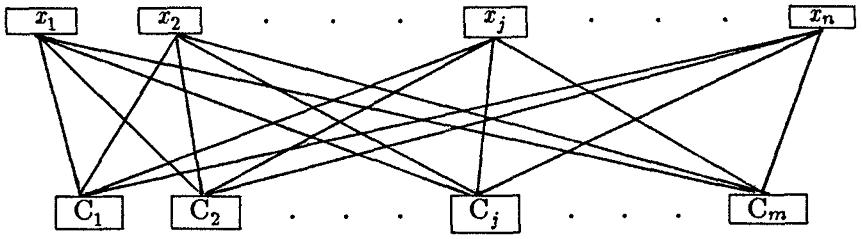
Conversely, given a sequence of extended Boolean formulas $\beta_1, \beta_2, \ldots, \beta_m$, containing variables among $\{x_1, x_2, \ldots, x_n\}$, we would like to be able to construct the corresponding Boolean circuit. Let $j \in \{1,2,\ldots,m\}$. If the Boolean formula β_j does not contain an element of Ω, then $\beta_j = 0$, or $\beta_j = 1$ or $\beta_j = x_k$, for some $k \in \{1,2,\ldots,n\}$. Then the Boolean circuit corresponding to β_j would be

$\boxed{0, y_j}$, $\boxed{1, y_j}$ or $\boxed{x_k, y_j}$, respectively. Assume that for some $l \geq 0$, if β_j contains l or fewer elements of Ω, then we can construct a Boolean circuit which corresponds to the formula β_j. Suppose β_j contains $l+1$ elements of Ω. Then $\beta_j = (\alpha_1 \wedge \alpha_2)$, or $\beta_j = (\alpha_1 \vee \alpha_2)$, or $\beta_j = (\neg\alpha_1)$, where α_1 and α_2 are Boolean formulas containing l or fewer elements of Ω. Since α_1 and α_2 contain l or fewer elements of Ω, we "have" corresponding Boolean circuits A^1 and A^2. The Boolean circuit C_j corresponding to the

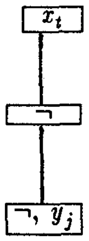
Boolean formula β_j would then be



Thus, by induction on the number of elements of Ω contained in an extended Boolean formula, we can construct the corresponding Boolean circuit, so that for each $j \in \{1,2,\ldots,m\}$, we have a Boolean circuit C_j which corresponds to the Boolean formula β_j. Rather than duplicating them, we let the Boolean circuits C_1, C_2, ..., C_m "share" input gates in the obvious way, and we obtain the Boolean circuit C which corresponds to the given sequence β_1, β_2, ..., β_m of Boolean formulas containing the variables $\{x_1,x_2,\ldots,x_n\}$, as shown below.



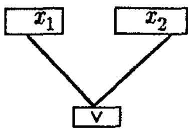
Of course, for each j, $k \in \{1,2,\ldots,m\}$ such that C_j is the circuit $\boxed{x_t, y_j}$, and the variable x_t appears in the Boolean formula β_k, we let C_j be as the circuit shown below, so that C_k can "share" the input gate labelled x_t.



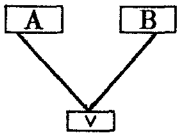
Thus, any function $f : \{0,1\}^n \rightarrow \{0,1\}^m$, where m, $n \in \mathbb{N}$, can be computed by a Boolean circuit. We now look at some specific functions on $\{0,1\}^n$ and the Boolean circuits which compute them, paying particular attention to the size and depth of these circuits.

Lemma 2.1.15. *Let $n \geq 2$. Then there is a fan-in 2 Boolean circuit C_n of depth $\lceil log_2 n \rceil$ and size $2n - 1$ which computes the function $\bigvee_{k=1}^{n} x_k : \{0,1\}^n \rightarrow \{0,1\}$.*

Proof. We proceed by induction on n. Let C_2 be the fan-in 2 Boolean circuit shown below.



The circuit C_2 computes $x_1 \vee x_2$, has size 3, which equals $2(2) - 1 = 2n - 1$, and has depth $\lceil \log_2 2 \rceil = \lceil \log_2 n \rceil = 1$, so the lemma holds for $n = 2$. Assume that for some $k \geq 2$, $n \leq k$ implies that there is a fan-in 2 Boolean circuit of size $2n - 1$ and depth $\lceil \log_2 n \rceil$ which computes the function $\bigvee_{j=1}^{n} x_j$. Let $r = \lceil \log_2(k + 1) \rceil$. It is immediate that $2^{r-1} \leq k$. This and the fact that $k + 1 \leq 2^r$ imply that both 2^{r-1} and $k + 1 - 2^{r-1}$ are less than or equal to k. By the induction hypothesis, there is a fan-in 2 Boolean circuit A of size $2(2^{r-1}) - 1 = 2^r - 1$ and depth $\lceil \log_2(2^{r-1}) \rceil = r - 1$ which computes the function $\bigvee_{j=1}^{2^{r-1}} x_j$, and there is a fan-in 2 Boolean circuit B of size $2(t) - 1$ and depth $\lceil \log_2 t \rceil$, where $t = (k + 1 - 2^{r-1})$, which computes the function $\bigvee_{j=2^{r-1}+1}^{k+1} x_j$. Let C_{k+1} be the fan-in 2 Boolean circuit shown below.



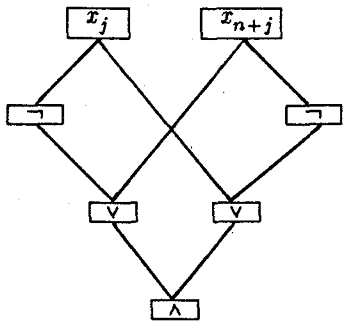
The circuit C_{k+1} computes $A(x_1, x_2, \ldots, x_m) \vee B(x_{m+1}, x_{m+2}, \ldots, x_{k+1})$, where $m = 2^{r-1}$, which is equal to $\left(\bigvee_{j=1}^{2^{r-1}} x_j\right) \vee \left(\bigvee_{j=2^{r-1}+1}^{k+1} x_j\right) = \bigvee_{j=1}^{k+1} x_j$. The circuit C_{k+1} has size $1 + (2^r - 1) + (2t) - 1 = (2^r - 1) + 2(k + 1 - 2^{r-1}) = 2(k + 1) - 1$. The circuit C_{k+1} has depth 1 plus the maximum of the depths of A and B, which implies that C_{k+1} has depth $1 + r - 1 = r = \lceil \log_2(k + 1) \rceil$. Therefore, the lemma holds for all $n \geq 2$. ◫

Corollary 2.1.16. *For all $n \geq 2$, there is a fan-in 2 Boolean circuit of size $2n - 1$ and depth $\lceil \log_2 n \rceil$ which computes the function $\bigwedge_{k=1}^{n} x_k$.*

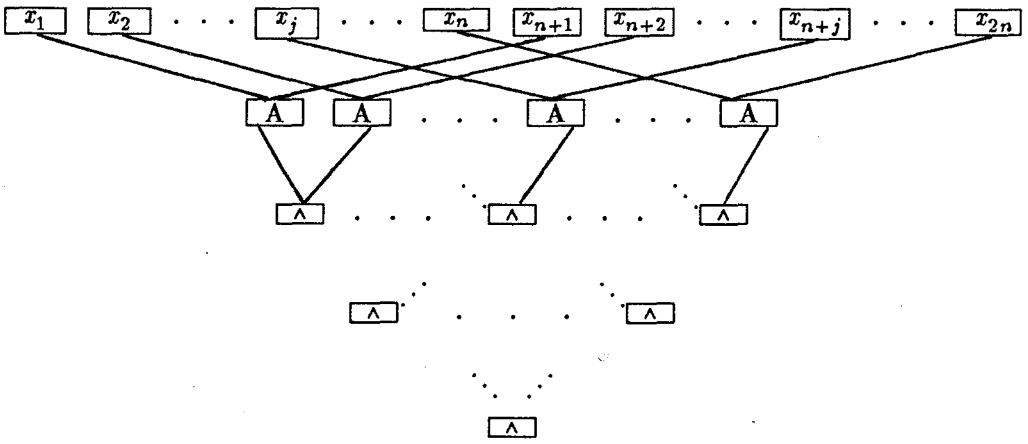
Lemma 2.1.17. *Let $n \in \mathbb{Z}^+$. Then there is a fan-in 2 Boolean circuit C of size $8n - 1$ and depth $\lceil \log_2 n \rceil + 3$ such that for all $u, v \in \{0,1\}^n$, $C(uv) = 1$ if and only if $u = v$.*

Proof. Let $u, v \in \{0,1\}^n$, where $n \in \mathbb{Z}^+$. Then $u = u_1 u_2 \cdots u_n = v_1 v_2 \cdots v_n = v$ if and only if $u_i = v_i$ for all $i \in \{1,2,\ldots,n\}$. Let $i \in \{1,2,\ldots,n\}$. Consider the Boolean formula $\beta_i(x_i, x_{n+i}) = ((\neg x_i) \vee x_{n+i}) \wedge (x_i \vee (\neg x_{n+i})) = x_i \leftrightarrow x_{n+i}$. For $u, v \in \{0,1\}^n$, β_i is such that $\beta_i(u_i, v_i) = 1$ if and only if $((\neg u_i) \vee v_i) \wedge (u_i \vee (\neg v_i)) = 1$, that is, if and only if $((\neg u_i) \vee v_i) = 1$ and $(u_i \vee (\neg v_i)) = 1$. If $u_i = 0$ and $(u_i \vee (\neg v_i)) = 1$, then $v_i = 0$. If $u_i = 1$ and $((\neg u_i) \vee v_i) = 1$, then $v_i = 1$. Therefore, $\beta_i(u_i, v_i) = 1$ implies that $u_i = v_i$. Clearly $u = v$ implies that $\beta_i(u_i, v_i) = 1$, so we have that $\beta_i(u_i, v_i) = 1$ if and only if $u_i = v_i$. Let $\beta(x_1, x_2 \ldots x_{2n})$ be the Boolean formula $\bigwedge_{j=1}^{n} \beta_j$. Then $\beta(uv) = 1$ if and only if $\beta_j(u_j, v_j) = 1$ for all $j \in \{1,2,\ldots,n\}$, if and only if $u_j = v_j$ for all $j \in \{1,2,\ldots,n\}$, if and only if $u = v$.

For each $j \in \{1,2,\ldots,n\}$ let C_j be the Boolean circuit shown below.



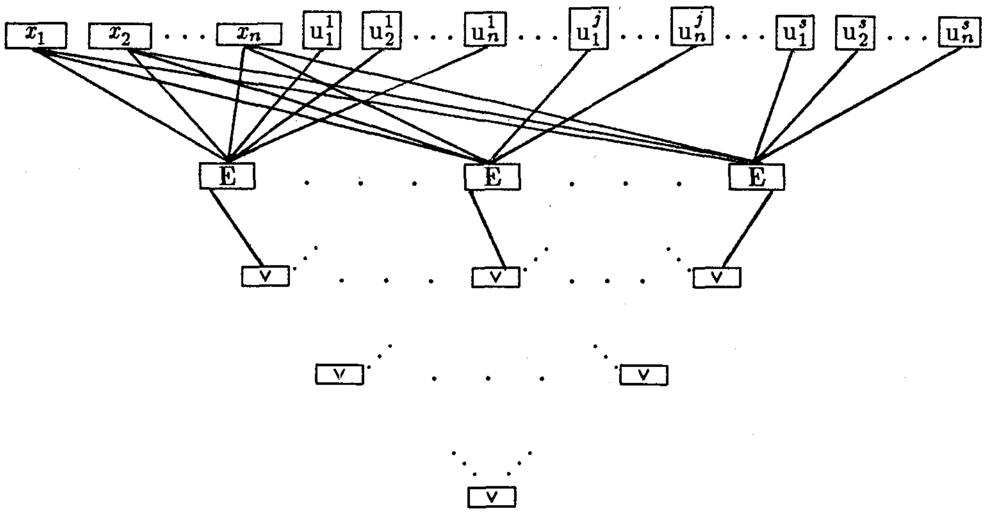
Clearly, C_j is the Boolean circuit which corresponds to the Boolean formula $\beta_j(x_j,x_{n+j})$. Let A be the Boolean circuit C_j excluding input gates, and let C be the Boolean circuit below.



The intention is, of course, that each pair of input gates labeled x_j and x_{n+j} be input gates for the "partial" circuit A, and that the output gates of the n copies of A be connected by "and" gates. The fan-in 2 Boolean circuit C computes the function from $\{0,1\}^{2n}$ to $\{0,1\}$ which corresponds to the Boolean formula $\bigwedge_{j=1}^{n} \beta_j(x_j,x_{n+j})$. Therefore, for all uv in $\{0,1\}^{2n}$, $C(uv) = 1$ if and only if $\beta_j(u_j, v_j) = 1$ for all $j \in \{1,2,\ldots,n\}$, that is, if and only if $u = v$. The Boolean circuit C_j has depth 3, so by Corollary 2.1.16, C has depth $\lceil \log_2 n \rceil + 3$. C_j has size 7, so again by Corollary 2.1.16, C has size $7n + (n - 1) = 8n - 1$. Therefore, C is the desired Boolean circuit. ◫

Corollary 2.1.18. Let $n \in \mathbb{Z}^+$, and let S be a nonempty subset of $\{0,1\}^n$, where S has cardinality s. Then there is a fan-in 2 Boolean circuit C of size $7sn + n - 1$ and depth $\lceil log_2 n \rceil + \lceil log_2 s \rceil + 3$ such that for all u in $\{0,1\}^n$, $C(u) = 1$ if and only if $u \in S$.

Proof. Let $S = \{u^1,u^2,\ldots,u^s\}$, $u^j = u_1^j u_2^j \cdots u_n^j$. Let E denote the circuit of Lemma 2.1.17, excluding input gates. Consider the fan-in 2 Boolean circuit C shown below.



The intention is, of course, that for each $j \in \{1,2,\ldots,s\}$, the input gates and the sources corresponding to the element $u^j \in S$ be "input gates" for the "partial circuit" E, and that the output gates of the s copies of E be connected by "or" gates. By Lemmas 2.1.15 and 2.1.17, C has depth $\lceil \log_2 s \rceil + 3 + \lceil \log_2 n \rceil$ and size $s(8n-1) - (s-1)n + s - 1 = 7sn + n - 1$. Let $u \in \{0,1\}^n$. It is clear that $C(u) = 1$ if and only if $E(uu^k) = 1$ for some $u^k \in S$, if and only if by Lemma 2.1.17, $u = u^k$ for some $u^k \in S$; that is, if and only if $u \in S$. Therefore, C is the desired Boolean circuit. □

We would like to show next that the "problem" of multiplication in a finite transformation monoid M_w can be "solved" by Boolean circuits. Of course, the first thing we need is a way of representing an element of M_w as a word over the alphabet $\{0,1\}$.

Definition 2.1.19 ([3]). Let $w \in \mathbb{Z}^+$, and let f be a function from $[w]$ to $[w]$. The Boolean representation $\bar{f}$ of the function f is the word

$$\bar{f} = \left((a_{ij})_{j=1}^{w}\right)_{i=1}^{w} \in \{0,1\}^{w^2},$$

where $a_{ij} = 1$ if $f(i) = j$, and otherwise $a_{ij} = 0$. We often write (a_{ij}) to simplify notation.

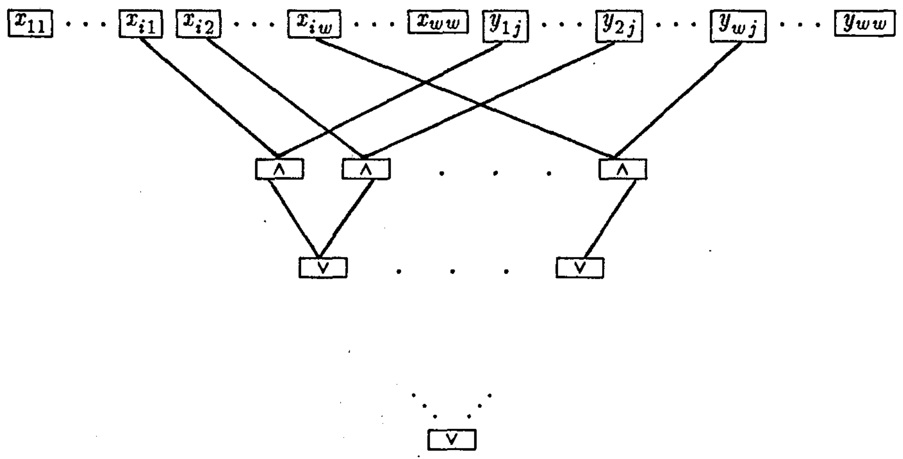
Lemma 2.1.20. *Let $w \in \mathbb{Z}^+$, and let f and g be functions from $[w]$ to $[w]$, and let $\bar{f} = (a_{ij})$ and $\bar{g} = (b_{ij})$ be their Boolean representations. We define $\bar{g} \circ \bar{f}$ to be the word (c_{ij}), where for each pair $i, j \in [w]$, $c_{ij} = \bigvee_{k=1}^{w} (a_{ik} \wedge b_{kj})$. Then $\bar{g} \circ \bar{f} = \overline{g \circ f}$.*

Proof. Note that $c_{ij} = 1$ if and only if for some $k \in [w]$, $a_{ik} \wedge b_{kj} = 1$; that is, if and only if for some $k \in [w]$, $a_{ik} = b_{kj} = 1$. Since $\bar{f}$ and $\bar{g}$ are the Boolean representations of the functions f and g, this is the case if and only if $f(i) = k$ and

$g(k) = j$; that is, if and only if $g(f(i)) = (g \circ f)(i) = j$. Therefore, $(c_{ij}) = 1$ if and only if $(g \circ f)(i) = j$, which implies that (c_{ij}) is the Boolean representation of $g \circ f$, and that $\bar{g} \circ \bar{f} = \overline{g \circ f}$. □

<u>Lemma</u> <u>2.1.21</u>. *Let $w \in \mathbb{Z}^+$. There is a fan-in 2 Boolean circuit of depth $\lceil \log_2 w \rceil + 1$ and size $2w^3 + w^2$, which, given input $\bar{f}\bar{g}$, where $\bar{f}$, $\bar{g}$ are the Boolean representations of functions $f, g : [w] \to [w]$, will output $\overline{g \circ f}$.*

<u>*Proof*</u>. We obtain the desired fan-in 2 Boolean circuit C as follows. First we need a row of $2w^2$ input gates. The first w^2 input gates are labelled by the variables x_{ij} in the order $((x_{ij})_{j=1}^{w})_{i=1}^{w}$, and the other w^2 input gates are labelled by the variables y_{ij}, in the same order. Let $i, j \in [w]$. For each $k \in w$, we connect the w pairs of input gates labelled by the variables x_{ik} and y_{kj} by an "∧" gate. We then connect these w "∧" gates by "∨" gates. By Lemma 2.1.15, this can be done using $w - 1$ fan-in 2 "∨" gates in depth $\lceil \log_2 w \rceil$. It is clear from Lemma 2.1.20, that the partial circuit we have thus far constructed is the fan-in 2 Boolean circuit of size $2w^2 + 2w - 1$ and depth $\lceil \log_2 w \rceil + 1$ which corresponds to the Boolean formula which defines c_{ij}. Therefore, this partial circuit outputs $(\overline{g \circ f})_{ij}$ on input $\bar{f}\bar{g}$, where $\bar{f}$ and $\bar{g}$ are the Boolean representations of functions $f, g : [w] \to [w]$. We complete the circuit by following this construction for the rest of the pairs $i, j \in [w]$, being careful that the ij-th output gate is the output gate of the partial circuit which computes $(\overline{g \circ f})_{ij}$. We obtain the fan-in 2 Boolean circuit C below, where the intention is of course that the circuit contains $w^2 - 1$ additional copies of the partial circuit which corresponds to $(\overline{g \circ f})_{ij}$.

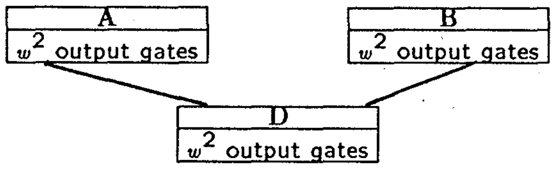


By Lemma 2.1.20, C computes $\overline{g \circ f}$ on input $\bar{f}\bar{g}$ where $\bar{f}$ and $\bar{g}$ are the Boolean representations of $f, g : [w] \to [w]$. C is a fan-in 2 Boolean circuit of size $2w^2 + w^2(2w - 1) = 2w^3 + w^2$, and depth $\lceil \log_2 w \rceil + 1$. Therefore, C is the desired Boolean circuit. □

We say that the Boolean circuit C in Lemma 2.1.21 computes the product of 2 elements of the monoid M_w.

Lemma 2.1.22. *Let $w \in \mathbb{Z}^+$. For each $k \geq 2$, there is a fan-in 2 Boolean circuit of size $(k - 1)2w^3 + w^2$ and depth $\lceil log_2 k\rceil(\lceil log_2 w\rceil + 1)$ which computes the product of k elements of the monoid M_w.*

Proof. Multiplication in the monoid M_w is associative, so the order in which the compositions are computed does not matter. We proceed by induction on k. The case $n = 2$ is Lemma 2.1.21. Assume that for some $n \geq 2$, $2 \leq k \leq n$ implies that there is a fan-in 2 Boolean circuit which computes the product of k elements of M_w of the right size and depth. Let $k = n + 1$. As in Lemma 2.1.15, if $r = \lceil \log_2(n + 1)\rceil$, both 2^{r-1} and $n + 1 - 2^{r-1}$ are less than or equal to n. By the induction hypothesis, there is a fan-in 2 Boolean circuit A of size $(2^{r-1} - 1)2w^3 + w^2$ and depth $(r - 1)(\lceil\log_2 w\rceil + 1)$ which computes the product of 2^{r-1} elements of M_w. Also by the induction hypothesis, since $n + 1 \leq 2^r$, there is a fan-in 2 Boolean circuit B of size $(n - 2^{r-1})2w^3 + w^2$ and depth t, where t is at most the depth of A, which computes the product of $n + 1 - 2^{r-1}$ elements of the monoid M_w. Using the output gates of the circuits A and B as input gates for the circuit D of Lemma 2.1.21, we obtain the fan-in 2 Boolean circuit C of size equal to the sum of the sizes of the circuits A, B, and D, less the $2w^2$ gates which are both output gates for A or B, and input gates for D, and depth equal to the maximum of the depths of A and B, plus the depth of the circuit D, shown below.



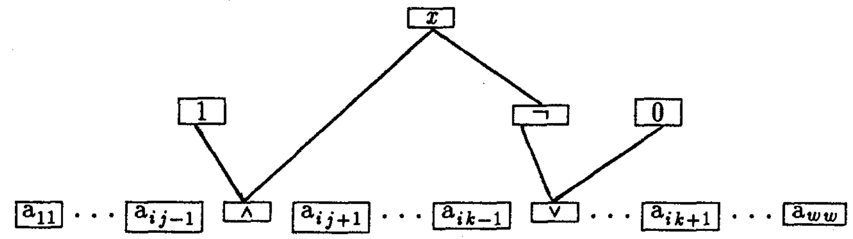
B has depth less than or equal to the depth of A, and by Lemma 2.1.21 the circuit D has depth $\lceil\log_2 w\rceil + 1$, so $(r - 1)(\lceil\log_2 w\rceil + 1) + \lceil\log_2 w\rceil + 1 = r(\lceil\log_2 w\rceil + 1)$ is the depth of C. Therefore, C has depth $\lceil\log_2(n + 1)\rceil(\lceil\log_2 w\rceil + 1)$. Since A has size $(2^{r-1} - 1)2w^3 + w^2$, B has size $(n - 2^{r-1})2w^3 + w^2$, and D has size $2w^3 + w^2$, the circuit C has size $(n)2w^3 + w^2$. Thus C is a fan-in 2 Boolean circuit of the right size and depth, which clearly computes the product of $n + 1$ elements of the monoid M_w. □

Lemma 2.1.23. *Let $w \in \mathbb{Z}^+$ and let $f, g \in M_w$ be such that for some i, $f(i) = j$ and $g(i) = k$, where $j \neq k$, and for $m \neq i$, $f(m) = g(m)$. Then there is a fan-in 2 Boolean circuit of size $w^2 + 4$ and depth 2 which will output $\overline{f}$ on input 1, and will output $\overline{g}$ on input 0.*

Proof. Without loss of generality, $j < k$. By definition of their Boolean representations,

$$\bar{f} = a_{11}\cdots a_{ij-1}1a_{ij+1}\cdots a_{ik-1}0a_{ik+1}\cdots a_{ww}, \text{ and}$$
$$\bar{g} = a_{11}\cdots a_{ij-1}0a_{ij+1}\cdots a_{ik-1}1a_{ik+1}\cdots a_{ww}.$$

Consider the fan-in 2 Boolean circuit C shown below.



C is a fan-in 2 Boolean circuit of size $w^2 + 4$ and depth 2. On input 1, C outputs $C(1) = a_{11}\cdots a_{ij-1}1a_{ij+1}\cdots a_{ik-1}0a_{ik+1}\cdots a_{ww} = \bar{f}$, and on input 0, C outputs $C(0) = a_{11}\cdots a_{ij-1}0a_{ij+1}\cdots a_{ik-1}1a_{ik+1}\cdots a_{ww} = \bar{g}$. □

Corollary 2.1.24. *Let $w \in \mathbb{Z}^+$. For any f, $g \in M_w$, there is a fan-in 2 Boolean circuit of size at most $w^2 + 3w + 1$ and depth 2 which outputs $\bar{f}$ on input 1, and outputs $\bar{g}$ on input 0.*

Proof. By definition of the Boolean representations $\bar{f}$ and $\bar{g}$, there are at most w pairs j, $k \in [w]$, $j < k$, such that $a_{ij} \neq b_{ij}$ and $a_{ik} \neq b_{ik}$, where $\bar{f} = (a_{rt})$, and $\bar{g} = (b_{rt})$ r, $t \in [w]$. Using Lemma 2.1.23 in the obvious way, we construct a fan-in 2 Boolean circuit of depth 2 and size at most $w^2 + 3w + 1$ which outputs $\bar{f}$ on input 1, and outputs $\bar{g}$ on input 0. □

Now that we have a sufficient understanding of Boolean circuits, we are in a position to show how a *family* of Boolean circuits constitutes a model of computation, which entails showing how such a family can be said to "compute a function," and how such a family can be said to "recognize a language." Prior to doing so we will make some observations about them.

Recalling how a Boolean circuit computes a function, it makes sense to think of the depth of the circuit as being the time required by the circuit to compute the function. Additionally, we note that in general, in the course of a computation performed by a Boolean circuit, several "operations" are performed simultaneously. This being the case, the Boolean circuit is said to compute the function f "in parallel." So-called parallel models of computation, those which make use of many "processors" to perform different parts of a computation simultaneously, are the subject of much current research ([11]). One obvious motivation for the interest in such models is that there are many computational problems such that the time required for their solution could be drastically reduced should a "parallel computer" be available. The *Boolean circuit family*, as defined below, is one such parallel model of computation.

Definiton 2.1.25 ([11]). Let $\mathcal{C} = (C_n)_{n=0}^{\infty}$ be a family of Boolean circuits such that for each $n \in \mathbb{N}$, the Boolean circuit C_n has n input gates. Then the function computed by $\mathcal{C}$ is the function $\mathcal{C} : \{0,1\}^* \to \{0,1\}^*$ defined by $\mathcal{C}(w) = C_k(w)$, where k is the length of w. A function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be computable by a Boolean circuit family if there is a Boolean circuit family $\mathcal{C} = (C_n)_{n=0}^{\infty}$ such that for each $n \in \mathbb{N}$, the Boolean circuit C_n has n input gates, and is such that for all $w \in \{0,1\}^n$, $C_n(w) = f(w)$.

Definition 2.1.26 ([16]). Let $\mathcal{C} = (C_n)_{n=0}^{\infty}$ be a family of Boolean circuits such that for each $n \in \mathbb{N}$, the Boolean circuit C_n has n input gates and one output gate. Then the language recognized by $\mathcal{C}$ is the language $L \subset \{0,1\}^*$ defined by $L = \{w \in \{0,1\}^* : \mathcal{C}(w) = 1\}$. A language $L \subset \{0,1\}^*$ is said to be recognizable by a Boolean circuit family if there is a Boolean circuit family $\mathcal{C} = (C_n)_{n=0}^{\infty}$ such that for each $n \in \mathbb{N}$, the Boolean circuit C_n has n input gates and one output gate, and is such that $\{w \in \{0,1\}^n : C_n(w) = 1\} = L \cap \{0,1\}^n$.

Given Definitions 2.1.25 and 2.1.26, we notice that a Boolean circuit family computes a function $f : \{0,1\}^* \to \{0,1\}^*$ if and only if for each n in $\mathbb{N}$, there exists a $k \in \mathbb{Z}^+$ such that $f(w) \in \{0,1\}^k$ for all $w \in \{0,1\}^n$, and that as it stands, *every* language over $\{0,1\}$ is recognizable by a Boolean circuit family ([20])! However, we are not concerned here with the *absolute* computational power of Boolean circuit families. What interests us in our study of Boolean circuit families is what they are able to do when "resource bounds" are imposed on them. Before we can discuss this further, we need the following definitions and results concerning the rate of growth of functions, all of which can be found in Davis [12].

Definition 2.1.27. Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$. If there are positive integers n_0 and c such that for all $n \geq n_0$, $f(n) \leq cg(n)$, we write $f(n) = O(g(n))$. If, in addition, $g(n) = O(f(n))$, we write $O(f(n)) = O(g(n))$, and we say that the functions f and g have the same rate of growth. If not, we say that the function g grows faster than the function f.

Proposition 2.1.28. *Let f and g be functions from $\mathbb{N}$ to $\mathbb{R}$.*
(i) $O(f(n)) = O(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \beta$ for some $\beta > 0$.
(ii) If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then the function f grows faster than the function g.
(iii) If $p(n)$ is a polynomial of degree r, then p and n^r have the same rate of growth. The polynomial p grows faster than n^m if $m < r$, and n^m grows faster

than p *if* $m > r$.

(*iv*) *If* $p(n)$ *is a polynomial of degree* $r \geq 1$, *then* p *grows faster than* $\log_2 n$.

(*v*) *If* $k > 1$, *then the function* k^n *grows faster than any polynomial.*

(*vi*) $\lceil \log_2 n \rceil$ *and* $\log_2 n$ *have the same rate of growth.*

(*vii*) *If* $a > 0$ *and* $b \geq 0$, *then* $a\lceil \log_2 n \rceil + b$ *and* $\lceil \log_2 n \rceil$ *have the same rate of growth.* ◻

Definition 2.1.29 ([11]). Let $\mathcal{C} = (\mathrm{C}_n)_{n=0}^{\infty}$ be a family of Boolean circuits, where the circuit C_n has n input gates. Let z and d be functions from $\mathbb{N}$ to $\mathbb{R}$. For each $n \in \mathbb{N}$, let $\sigma(n)$ be the size of the circuit C_n, and let $\delta(n)$ be the depth of the circuit C_n. Clearly, σ and δ are functions from $\mathbb{N}$ to $\mathbb{N}$. We say that the family $\mathcal{C}$ has <u>size</u> $\mathrm{z}(n)$ if $\sigma(n) = \mathrm{O}(\mathrm{z}(n))$, and we say that $\mathcal{C}$ has <u>depth</u> $\mathrm{d}(n)$ if $\delta(n) = \mathrm{O}(\mathrm{d}(n))$.

Example 2.1.30. Let $\mathcal{C} = (\mathrm{C}_n)_{n=1}^{\infty}$ be the family of Boolean circuits such that for all $n \in \mathbb{Z}^+$, C_n is the circuit of Lemma 2.1.15 for inputs of length n. Then $\sigma(n) = 2n - 1 = \mathrm{O}(n)$, and $\delta(n) = \lceil \log_2 n \rceil = \mathrm{O}(\log_2 n)$, so that the family $\mathcal{C}$ of Boolean circuits has size n and depth $\log_2 n$.

The "resources" of a Boolean circuit family are its size and depth. As remarked previously, our interest is in the capabilities of Boolean circuit families of a given size and depth. Once resource bounds are imposed, it is clear that this will limit the scope of Definitions 2.1.25 and 2.1.26 to some extent. It would be even better if we could limit the definitions so that every function (language) which is defined to be computable (recognizable) by a Boolean circuit family will actually be computable (recognizable) by a *Turing machine*, which is, after all, the *definition* of a computable function (recognizable language) ([17]). This issue will be addressed in Chapter Three.

§ 2.2 *Bounded Width Branching Programs*

In this section we discuss the computational model by which the class BWBP of languages is defined. We begin with a discussion of branching programs, which are the components of the model. Here we will be reasonably brief, because this model will be encountered again in Chapter three.

Definition 2.2.1 ([10]). A <u>branching program</u> B (abbreviated BP) on n variables is a finite acyclic directed graph which satisfies:

(*i*) B has one source which is called the input node of B;

(*ii*) all sinks of B carry one of the labels 0, 1;

(iii) all other nodes are labeled by a variable in $\{x_1,x_2,\ldots,x_n\}$;

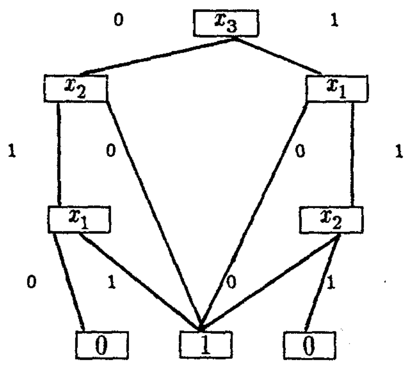
(iv) there are at most two edges leaving each node of B. One edge carries the label 0, the other edge carries the label 1. If there is only one edge leaving a node of B, that edge is understood to carry both labels 0 and 1.

Definition 2.2.2 ([10]). A directed graph is said to be levelled if it can be partitioned into levels $L_1, L_2,\ldots, L_k$, such that for all $i \in \{0,1,2,\ldots,k\}$, every edge leaving a node on level i enters a node on level $i + 1$.

Note that a graph is levelled if and only if every node ν in the graph is such that every path from a source to ν is of length equal to the rank of ν.

Definition 2.2.3. Let B be a branching program on n variables. The length of the longest path in B from the input node to a sink of B is called the length of B. The number of nodes in B is called the size of B. If B is levelled, then the width of B is the number of nodes on the level of B on which there is the largest number of nodes.

Example 2.2.4. Consider the branching program B below. By convention, all edges are directed down.



The branching program B has size 8 and length 3. Note that B is not levelled, so the width of B is not defined.

We now describe how a branching program B on n variables computes a Boolean function from $\{0,1\}^n$ to $\{0,1\}$. Given $\mathrm{a} = \mathrm{a}_1\mathrm{a}_2\cdots\mathrm{a}_n \in \{0,1\}^n$, starting at the input node labelled, say, x_i, if a_i is 1 (0), follow the edge labelled 1 (0) leaving the input node and entering a node labelled x_j. Then the edge leaving the node labelled x_j which is labelled a_j is followed. This process continues until a sink is reached ([10]). Then B(a) is defined to be the label of the sink reached when B is given input a and "evaluated" as described. The branching program B of Example 2.2.4 computes the Boolean function from $\{0,1\}^3$ to $\{0,1\}$ given by $B(010) = 0$, $B(111) = 0$, and for all other $\mathrm{a} \in \{0,1\}^3$, $B(\mathrm{a}) = 1$.

The set $\{u \in \{0,1\}^n : B(u) = 1\}$ is called the language recognized by the branching program B on n variables. We note that a language $L \subseteq \{0,1\}^n$ is recognizable by a branching program if and only if the Boolean function $f : \{0,1\}^n \to \{0,1\}$ defined by $f(u) = 1$ if and only if $u \in L$ is computable by a branching program.

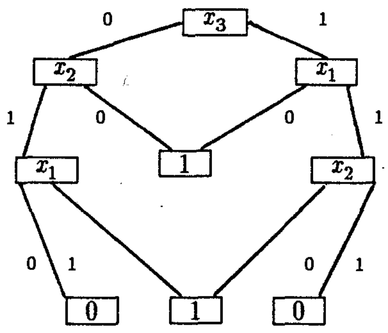
If a branching program is levelled, it has the advantage that the level of a node represents the time required to reach the node in the course of a computation, viewing the source node as level 0.

Proposition 2.2.5 ([9]). *Let B be a branching program on n variables of size s and length l. Then there is a levelled branching program B' of length l and size at most s^2 such that $B'(a) = B(a)$ for all $a \in \{0,1\}^n$.*

Proof. Throughout this proof, a "path" means a path starting at the source node ν_1. Let B be a branching program on n variables of length l and size s. We first create an array of nodes with s columns and $l + 1$ rows, the top row being row 0, each row containing all of the nodes $\nu_1, \nu_2, \ldots, \nu_s$ of B. For each path of length k, $0 \leq k \leq l - 1$, in B ending at node ν_j, and each edge from ν_j to ν_t labelled 0 (1) in B, we add an edge labelled 0 (1) from node ν_j in row k of the array to the node ν_t in row $k + 1$ of the array. We then remove all nodes of the array with indegree and outdegree zero. It is clear that this construction will yield a levelled branching program B′ of size at most $s(l + 1) \leq s^2$, since $l + 1 \leq s$. It is also clear that this construction will yield a branching program B′ which will compute the same function as that computed by B. □

Proposition 2.2.5 shows that any Boolean function which is computable by a branching program is computable by a levelled branching program, which allows us to restrict our attention to levelled branching programs.

Example 2.2.6. Following the procedure in the proof of Proposition 2.2.5, a levelled version of the branching program of Example 2.2.4 would be as shown below.



This branching program is levelled, has size 9, length 3, width 3, and computes the same function as that computed by the branching program in Example 2.2.4.

As it stands, a levelled branching program B on n variables, of length l and width w could have sinks on level k where $k < l$. Let m be the least integer such that there is a sink labelled 0 on level m. Consider the following. "String together" a chain of $l - m$ nodes labelled by a variable x_i, delete all edges entering the sink labelled 0 from nodes on level $m - 1$, connect these nodes to one end of the chain by copies of the deleted edges, and then connect the other end of the chain to the sink labelled 0. For each sink labelled 0 on level t, where $m < t < l$, delete the sink and connect all nodes on level $t - 1$, from which an edge entered the sink, to the node in the chain of nodes labelled x_i from which there is a path of length $l - t$ to the sink labelled 0 on level l. This moves all the sinks labelled 0 on level $t < l$ down to level l, without changing the length of B or the function computed by B. If we follow this procedure for sinks labelled 1, we obtain a branching program of length l, of size less than or equal to $s + 2(l - 2)$, and of width less than or equal to $w + 2$, which computes the same function as that computed by the branching program B ([3]).

The discussion above shows that any function computable by a levelled branching program is computable by a levelled branching program where all sinks occur on the bottom level. This allows us, without loss of generality, to restrict our attention to such branching programs. From now on, by "a branching program", we will mean one of this type.

We now show how a family of branching programs can be said to compute a Boolean function on $\{0,1\}^*$, and how a family of branching programs can be said to recognize a language over the alphabet $\{0,1\}$.

Definition 2.2.7 ([10]). Let $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ be a family of branching programs, where for each $n \in \mathbb{N}$, B_n is a branching program on n variables. The function computed by $\mathfrak{B}$ is the Boolean function defined by $\mathfrak{B}(u) = B_n(u)$, where $u \in \{0,1\}^n$. The language recognized by $\mathfrak{B}$ is the language $\{u \in \{0,1\}^* : \text{if } |u| = n, \text{ then } B_n(u) = 1\}$.

We conclude the section by defining the "resources" of a family of branching programs, and defining a complexity class of languages recognizable by branching program families with bounded resources.

Definiton 2.2.8. Let f, h, and g be functions from $\mathbb{N}$ to $\mathbb{R}$, and let $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ be a family of branching programs. Let σ, λ, and ω be functions from $\mathbb{N}$ to $\mathbb{N}$, where $\sigma(n)$ is the size of the branching program B_n, $\lambda(n)$ is the length of the branching program B_n, and $\omega(n)$ is the width of the branching program B_n. We say that the family $\mathfrak{B}$ has size $f(n)$ if $\sigma(n) = O(f(n))$, we say that $\mathfrak{B}$ has length $h(n)$ if $\lambda(n) = O(h(n))$, and we say that $\mathfrak{B}$ has width $g(n)$ if $\omega(n) = O(g(n))$.

Definition 2.2.9 ([3], [10]). BWBP is the class of languages $L \subseteq \{0,1\}^*$ such that L is recognizable by a family $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ of branching programs of bounded width, that is, for which there is a $w \in \mathbb{Z}^+$, such that for all $n \in \mathbb{N}$, $\omega(n) \leq w$, and polynomial size (or, equivalently, polynomial length, given bounded width).

§ 2.3 *Programs For a NUDFA over a Finite Monoid*

In this section we discuss our final non-uniform computational model, which was clearly inspired by finite state machines for regular languages. In this case, the model does not consist of a family of components, but it is a single machine along with a family of programs, one for each input length.

Definition 2.3.1 ([3], [5]). Let M be a finite monoid, and Σ be a finite alphabet. A non-uniform deterministic finite automaton $N(M,\Sigma)$ over M with input alphabet Σ is a machine, which, for each $n \in \mathbb{N}$, given an input word in Σ^n and an *n*-program P_n (as defined below), produces an element of M as output.

A non-uniform deterministic finite automaton will hereafter be referred to as a NUDFA, and "the NUDFA N over M with input alphabet Σ," will be denoted $N(M,\Sigma)$.

Definition 2.3.2 ([5]). Let M be a finite monoid. An *n*-program P_n for a NUDFA $N(M,\Sigma)$ of length l is a sequence of instructions $\mu_1, \mu_2, \ldots, \mu_l$. For each $k \in \{1,2,\ldots,l\}$, $\mu_k = \langle i_k, f_k \rangle$, where $i_k \in \{1,2,\ldots,n\}$, and f_k is a function from Σ to M. The *n*-program P_n defines a function from Σ^n to the monoid M as follows. For each $u = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^n$, $P_n(u) = \mu_1(u)\mu_2(u)\cdots\mu_l(u)$, where for each $k \in \{1,2,\ldots,l\}$, $\mu_k(\sigma) = f_k(\sigma_{i_k})$. For each $m \in M$, there is a "constant" 0-program P_0^m with no instructions such that $P_0^m(\varepsilon) = m$, where $\{\varepsilon\} = \Sigma^0$.

Thus, given an input word $u = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^n$ and the *n*-program P_n, the NUDFA $N(M,\Sigma)$ produces the output $P_n(u) \in M$. We now describe how a NUDFA, together with a family of *n*-programs, constitutes a model of computation, which, as usual, entails showing how a NUDFA can be said to compute a function, and how a NUDFA can be said to recognize a language.

Definition 2.3.3. Let $N(M,\Sigma)$ be a NUDFA, and let $\mathfrak{P} = (P_n)_{n=0}^{\infty}$ be a family of programs for $N(M,\Sigma)$. The function computed by $N(M,\Sigma)$ with family of *n*-programs $\mathfrak{P}$ is the function N from Σ^* to M defined by $N(u) = P_n(u)$, where $u \in \Sigma^n$.

Definition 2.3.4 ([5]). Let $L \subseteq \Sigma^*$, where Σ is a finite alphabet. The language L is said to be recognizable by a family of n-programs for a NUDFA over a finite monoid if and only if there is a finite monoid M and a family $\mathfrak{P} = (P_n)_{n=0}^{\infty}$ of n-programs for the NUDFA $N(M,\Sigma)$, such that for each $n \in \mathbb{N}$, there is a subset A_n of M such that $P_n(u) \in A_n$ if and only if $u \in L \cap \Sigma^n$.

Proposition 2.3.5. *Every regular language is recognizable by a family of n-programs for a NUDFA over a finite monoid.*

Proof. Let Σ be a finite alphabet, and let $L \subseteq \Sigma^*$ be a regular language. Since L is regular, there is a finite monoid M, a subset B of M, and a homomorphism $\overline{\phi}$ from Σ^* to M such that $\overline{\phi}(u) \in B$ if and only if $u \in L$. Let $\phi : \Sigma \to M$ be the restriction of $\overline{\phi}$ to Σ. Let P_0^m be the 0-program for the NUDFA $N(M,\Sigma)$ such that $P_0^m(\varepsilon) = \overline{\phi}(\varepsilon) = m$. Then $P_0(\varepsilon) = m \in B$ if and only if $\varepsilon \in L \cap \Sigma^0$ (since L is regular). For each $n \in \mathbb{Z}^+$, let P_n be the n-program for the NUDFA $N(M,\Sigma)$ with sequence of instructions $\mu_1, \mu_2, \ldots, \mu_n$, where for each k, $1 \leq k \leq n$, the instruction $\mu_k = \langle k, \phi\rangle$. Let $n \in \mathbb{Z}^+$, $u = u_1u_2\cdots u_n \in \Sigma^n$. Then $P_n(u) \in B$ if and only if $\mu_1(u)\mu_2(u)\cdots\mu_n(u) \in B$; that is, if and only if $\phi(u_1)\phi(u_2)\cdots\phi(u_n) \in B$. Since the function $\phi : \Sigma \to M$ is the restriction of the homomorphism $\overline{\phi}$ to Σ, $\phi(u_1)\phi(u_2)\cdots\phi(u_n) = \overline{\phi}(u_1u_2\cdots u_n) = \overline{\phi}(u)$. Thus, $P_n(u) = \overline{\phi}(u)$. Since $\overline{\phi}(u) \in B$ if and only if $u \in L$, it follows that $P_n(u) \in B$ if and only if $u \in L \cap \Sigma^n$. Therefore, the finite monoid M, and the family $\mathfrak{P} = (P_n)_{n=0}^{\infty}$ of n-programs for the NUDFA $N(M,\Sigma)$ are such that L is recognizable by the family $\mathfrak{P}$ over $N(M,\Sigma)$. As L was an arbitrary regular language, it follows that every regular language is recognizable by a family of n-programs for a NUDFA over a finite monoid. $\square$

Example 2.3.6. The non-regular language $L = \{\sigma^n\tau^n : n \in \mathbb{N}\}$ ([13]) over the alphabet $\Sigma = \{\sigma, \tau\}$ is recognizable by a family of programs for a NUDFA over a finite monoid. Let M be the monoid $\{0,1\}$, where 1 is the identity of M and 0 is a zero. Then L is recognizable by the family $\mathfrak{P}$ of n-programs over the NUDFA $N(M,\Sigma)$, where $\mathfrak{P} = (P_n)_{n=0}^{\infty}$ is defined as follows. Let P_0^1 be the 0-program such that $P_0^1(\varepsilon) = 1$. Let $\bar{0}$ denote the constant function from Σ to M with value 0. Let $f, g : \Sigma \to M$ be defined by $\sigma \mapsto 1$, $\tau \mapsto 0$ and $\sigma \mapsto 0$, $\tau \mapsto 1$ respectively. For odd n, let P_n be the n-program $\langle 1, \bar{0}\rangle$, and for $n = 2k$, $k \geq 1$, let P_n be the n-program $\langle 1, f\rangle, \langle 2, f\rangle, \ldots, \langle k, f\rangle, \langle k+1, g\rangle, \langle k+2, g\rangle, \ldots, \langle n, g\rangle$. Let $\rho \in \Sigma^n$. If $n = 0$, then $P_0(\rho) \in \{1\}$ if and only if $\rho = \varepsilon \in L \cap \Sigma^0$. If n is odd, then $P_n(\rho) \in \{1\}$ if and only if $\rho \in \emptyset = L \cap \Sigma^n$. If $n = 2k$ for some $k \geq 1$, then $\rho = \rho_1\rho_2\cdots\rho_k\rho_{k+1}\rho_{k+2}\cdots\rho_n$, and $P_n(\rho) \in \{1\}$ if and only if $f(\rho_1)f(\rho_2)\cdots f(\rho_k)g(\rho_{k+1})g(\rho_{k+2})\cdots g(\rho_n) = 1$. A product of elements of M is the identity 1 of M if and only if each element in the product is 1, so that $P_n(\rho) = 1$ if and only if $f(\rho_i) = 1$ for all $i \in \{1,2,\ldots,k\}$ and $g(\rho_j) = 1$ for all $j \in \{k+1, k+2,\ldots,n\}$. But $f(\rho_i) = 1$ if and only if $\rho_i = \sigma$, and $g(\rho_j) = 1$ if and only if $\rho_j = \tau$, together imply that $P_n(\rho) \in \{1\}$ if and only if $\rho = \sigma^k\tau^k \in L \cap \Sigma^n$. Therefore, L is recognizable by the family $\mathcal{P}$ of n-programs for the NUDFA $N(M,\Sigma)$.

We now define the "resources" of a family of programs for a NUDFA over a finite monoid so that we will be able to define a complexity class of languages recognizable by a family of n-programs with bounded resources.

Definition 2.3.7. Let $\mathcal{P} = (P_n)_{n=0}^{\infty}$ be a family of programs for a NUDFA $N(M,\Sigma)$, let f be a function from $\mathbb{N}$ to $\mathbb{R}$, and let $\lambda : \mathbb{N} \to \mathbb{N}$ be such that $\lambda(n)$ is the length of P_n. We say that the family $\mathcal{P}$ has length $f(n)$ if $\lambda(n) = O(f(n))$.

Definition 2.3.8. Let PLP denote the following class of languages. A language L over a finite alphabet Σ is in PLP if and only if there is a finite monoid M, and a coding ζ from Σ^* to $\{0,1\}^*$ such that the language $\zeta(L)$ is recognizable by a polynomial length family of n-programs for the NUDFA $N(M,\{0,1\})$.

Proposition 2.3.9. *Every regular language is in PLP.*

Proof. Let L be a regular language over the alphabet $\Sigma = \{\sigma_1,\sigma_2,\ldots,\sigma_m\}$ where $c(\Sigma) = m$, $m \geq 3$. Let $\bar{\alpha}$ be the coding from Σ^* to $\{0,1\}^*$ which is uniquely determined by the function $\alpha : \Sigma \to \{0,1\}^*$ given by $\alpha(\sigma_i) = 0^{i-1}10^{m-i}$, the word over $\{0,1\}$ which has only one 1 as the i-th letter, for $1 \leq i \leq m$. Since $\bar{\alpha}$ is injective, $\bar{\alpha}$ is such that $\bar{\alpha}^{-1}(\varepsilon) = \varepsilon$. By Proposition 1.1.6, $\bar{\alpha}(L)$ is a regular language over $\{0,1\}$. The proof of Proposition 2.3.5 shows that every regular language is recognizable by a family of NUDFA programs of length n, a polynomial, so there is a finite monoid M such that $\bar{\alpha}(L)$ is recognizable by a polynomial length family of n-programs for the NUDFA $N(M,\{0,1\})$. $\square$

Chapter III Non-Uniform Models of Computation and NC^1

In this chapter we establish relationships between the various non-uniform models of computation defined in Chapter Two, and, finally, establish a relationship between language classes defined by these models and the class NC^1 of languages.

In Section one, we introduce the concept of uniformity. This concept enables us both to define the parallel complexity class NC, and its subclasses NC^k, and AC^k, and to address the difficulty in comparing non-uniform models of computation with more familiar uniform models.

In Section two, we present some of D. A. Barrington's work on bounded width branching programs. Specifically, we show how he redefined the branching program, and selected a suitable definition of the language recognized by one of his branching programs, under which the language class BWBP is preserved.

In Section three, using yet another definition of the language recognized by one of Barrington's branching programs, we exhibit the relationship between the language classes BWBP and PLP.

In Section four, we begin with a reconstruction of the proof of Barrington's result that BWBP = (nonuniform) NC^1, from which it follows that PLP, the class of languages recognizable by polynomial length families of n-programs for a NUDFA over a finite monoid is also NC^1. We state some consequences of this work, and give a sampling of some further research which was inspired by it.

§ 3.1 *Uniform Boolean Circuit Families and the Parallel Complexity Class NC*

In the Section 2.1 we mentioned that Boolean circuit families cannot compute all computable functions, and yet they can compute some functions which are not computable. This makes it difficult to compare them with uniform models of computation such as the Turing machine. There are two basic approaches which make comparison possible ([20]).

One approach, suggested by Borodin ([8]) is to impose a uniformity condition which restricts the class of Boolean circuit families under consideration to those families $\mathcal{C}$ which are such that there is a Turing machine which can, given an input word of length n, generate a description of the circuit C_n in $\mathcal{C}$, using a certain amount of time and space. If this is the case, then there is most certainly a Turing machine which can do this and then "decode the description" and simulate the circuit C_n on that input word. Limiting the class of Boolean circuit families to those which are uniform, in the sense of Borodin, thus guarantees that they compute only functions which are *computable.*

The other approach (see [20], [24]) is to "make Turing machines non-uniform", meaning allowing a Turing machine to work on inputs of only one length, by giving the Turing machine some extra information (such as a description c_n of the n-th circuit in a family $\mathcal{C}$ of Boolean circuits) for each input length n along with the input word w (of length n). If, given this information, the Turing machine computes $\mathcal{C}(w)$ for all $w \in \{0,1\}^*$, we say that the Turing machine "non-uniformly" computes $\mathcal{C}$. Ordinarily we would say that the function computed by the Turing machine is the function g from the set
$\{uw : u = c_n, \mid w \mid = n \text{ for some } n \in \mathbb{N}\}$ to $\{0,1\}^*$ defined by $g(uw) = \mathcal{C}(w)$.

We now give some definitions which will enable us to give some uniformity conditions on Boolean circuit families. We will then be in a position to define the parallel complexity class NC, and its subclasses NC^k and AC^k.

Definition 3.1.1 (see [26], [21], [11]). Let C be a fan-in 2 Boolean circuit of size s with n input gates and 1 output gate. Let each of the s gates g_i of C, $i \in \{0,1,\dots,s-1\}$, be assigned an "address"--the binary representation $\hat{\imath}$ of the integer i, with enough zeros added at the right end to form a string of length $\lceil \log_2 s \rceil$; reserving $\hat{0}$ for the output gate, and reserving $\hat{q}$ where $1 \leq q \leq n$ for the input gates labelled x_q. If g_i has indegree two, let $\hat{g}_i = \hat{\imath}\tau\hat{\jmath}\hat{k}$, where g_j and g_k are the gates of C which fan into the gate g_i, and $\tau \in \{\wedge, \vee\}$ is the label of the gate g_i. Of course, if g_i has indegree 1, then $\hat{g}_i = \hat{\imath}\neg\hat{\jmath}$, where g_j fans into g_i. If g_i has indegree 0 and is not an input gate, then $\hat{g}_i = \hat{\imath}\tau$, where $\tau \in \{0,1\}$. If g_i is an input gate, then let $\hat{g}_i = \hat{\imath}x$. The standard encoding $\hat{c}$ of the Boolean circuit C is then $\hat{c} = \hat{g}_0\hat{g}_1\cdots\hat{g}_{s-1}$.

We note that the sequence $(\hat{c}_n)_{n=0}^{\infty}$ of standard encodings of a Boolean circuit family $\mathcal{C} = (C_n)_{n=0}^{\infty}$ is one example of a "description" of the family $\mathcal{C}$. Although it is only defined for Boolean circuits with one output gate, it is clear how one would generalize the standard encoding for more general Boolean circuits with fan-in greater than two, and more than one output gate.

Definition 3.1.2 ([21]). Let g be a gate in a fan-in 2 Boolean circuit C, where for each gate ν in C, each edge entering ν is assigned a different label $Z \in \{L, R\}$. Let $p = Z_1Z_2\cdots Z_n \in \{L, R\}^*$. The gate $g(p)$ determined by p and g is defined inductively as follows.

(i) For $n = 0$, $g(p)$ is the gate g.

(*ii*) For $n \geq 1$, g(p) is the gate which has an edge entering gate $g(Z_1Z_2\cdots Z_{n-1})$ with label L (R) if $Z_n = L$ (if $Z_n = R$).

Definition 3.1.3 ([21]). Let $\mathcal{C} = (C_n)_{n=0}^{\infty}$ be a family of fan-in 2 Boolean circuits of size $z(n)$ and depth $d(n)$ such that for each $n \in \mathbb{N}$, the circuit C_n has n input gates and one output gate, and each gate of C_n is assigned an address $\hat{k}$, where $\hat{k}$ denotes the binary representation of an integer $k \in \{0,1,\ldots,\sigma(n)\}$, with $\hat{0}$ reserved for the output gate of C_n and $\hat{q}$ reserved for the input gate of C_n labelled x_q for $1 \leq q \leq n$, as in Definition 3.1.1, and for every gate ν in C_n, each edge from μ to ν is assigned a different label from {L, R}. Let p denote an element of $\{L, R\}^*$, letting ε denote the empty string. Let $\hat{n}$, $\hat{m}$ be elements of $\{0,1\}^*$, and let τ denote an element of $\Gamma \cup \{x\}$, where x is used to label input gates as before. The extended connection language ECL($\mathcal{C}$) of the family $\mathcal{C}$ is the set of all strings $\hat{n}\hat{m}p\tau$ such that either

(*i*) $p = \varepsilon$ and the gate g in C_n with assignment $\hat{m}$ has label τ,

(*ii*) $p \neq \varepsilon$, $1 \leq |p| \leq \lceil \log_2 z(n) \rceil$, and the gate g in C_n with assignment $\hat{m}$ is such that the gate g(p) has label τ.

For each circuit C_n in a size $z(n)$ family $\mathcal{C} = (C_n)_{n=0}^{\infty}$ of fan-in 2 Boolean circuits, the ECL of the family contains, for each gate g in each circuit C_n in the family, an encoding of the label τ and address $\hat{m}$ of g in C_n, and an encoding of the address $\hat{m}$ of g along with the path and the label of each gate from which there is a path p in the circuit C_n of length less than or equal to $\lceil \log_2 z(n) \rceil$ which ends at g. We observe that the standard encoding is closely related to the subset of the ECL obtained by restricting the paths to be of length less than or equal to 1. This subset is called the direct connection language, or DCL ([8]).

Recalling the observation that the standard encoding can be generalized for a fan-in k Boolean circuit with n input gates and m output gates, we must recognize the possibility of generalizing the ECL to families of such circuits. We further note that a family $\mathcal{C} = (C_n)_{n=1}^{\infty}$ of fan-in 2 Boolean circuits of size $z(n)$ and depth $d(n)$ is "completely described" by the ECL of $\mathcal{C}$, though this fact is not as transparent as the fact that the sequence $(\hat{c}_n)_{n=1}^{\infty}$ of standard encodings of the circuits C_n in $\mathcal{C}$ gives a complete description of the family $\mathcal{C}$.

It is worth remarking that there is a Turing machine which will output the binary representation $\hat{n}$ of n when given any word of length n as input, in time $O(n^3)$. It is also worth remarking that any function which is computable by a Turing machine in space

$O((\log_2 n)^k)$ is computable in polynomial time ([20], [22]).

If the "complete description" of a circuit family is such that we can make sense out of it, then there most certainly exists a Turing machine which can make sense out of it, so it could then be used by a Turing machine to "non-uniformly" compute the function which is computed by the family $\mathcal{C}$. If, in addition, the complete description can be generated by a Turing machine, the function would then be a *computable function*, as discussed previously. Having described the standard encoding of a Boolean circuit, and the ECL of a family of Boolean circuits, we are now ready to define some uniformity conditions on Boolean circuit families. There are several others that we haven't included, and various relationships have been shown between all the various uniformity conditions ([21], [8], [26]).

Definition 3.1.4 . Let $\mathcal{C} = (C_n)_{n=0}^{\infty}$ be a family of fan-in 2 Boolean circuits of size $z(n)$ and depth $d(n)$. Then

(i) the family $\mathcal{C}$ is said to be *BC*-uniform ([8], [11]) if there is a deterministic Turing machine of space complexity $\log_2 z(n)$ which for each $n \in \mathbb{N}$, given the input $\hat{n}$, generates the standard encoding $\hat{c}_n$ of the circuit C_n, and

(ii) the family $\mathcal{C}$ is said to be *E**uniform ([21]) if the language ECL($\mathcal{C}$) over $\{\wedge,\vee,\neg,x,0,1,L,R\}^*$ can be recognized by an Alternating Turing Machine in time $O(d(n))$ and space $O(\log_2 z(n))$.

A discussion of Alternating Turing machines is beyond the scope of this thesis. Ruzzo ([21]) has suggested that the Alternating Turing machine is a generalization of the nondeterministic Turing machine, and we will leave it at that. We note that these uniformity conditions, though they are, strictly speaking, defined for families of fan-in 2 Boolean circuits with one output gate, can be generalized to apply to families of Boolean circuits with more than one output gate, and fan-in greater than 2.

Definition 3.1.5 ([20], [26], [11])

(i) NC is the class of languages over $\{0,1\}$ which are recognizable by a *BC*-uniform family of fan-in 2 Boolean circuits of size n^j and depth $(\log_2 n)^m$ for some j, $m \in \mathbb{N}$.

(ii) For each $k \geq 1$, NC^k is the class of languages over $\{0,1\}$ which are recognizable by a *BC*-uniform family of fan-in 2 Boolean circuits of size n^j and depth $(\log_2 n)^k$ for some $j \in \mathbb{N}$.

(iii) For all $k \geq 0$, AC^k is the class of languages over $\{0,1\}$ which are recognizable by a *BC*-uniform family of Boolean circuits of size n^j and depth $(\log_2 n)^k$, for some

$j \in \mathbb{N}$, with no restriction on the fan-in of the circuits in the family.

Definition 3.1.6. A *BC*-uniform family of fan-in 2 Boolean circuits of size n^j and depth $(\log_2 n)^k$ for some $j, k \in \mathbb{N}$ is called an NC^k circuit family. A *BC*-uniform family of Boolean circuits, with unrestricted fan-in, of size n^j and depth $(\log_2 n)^k$, for some j, k $\in \mathbb{N}$, is called an AC^k circuit family.

Let $\mathrm{L} \in \mathrm{NC}$. Then L is recognizable by a *BC*-uniform family $\mathcal{C} = (\mathrm{C}_n)_{n=0}^{\infty}$ of fan-in 2 Boolean circuits of size n^q and depth $(\log_2 n)^p$ for some $q, p \in \mathbb{N}$. Consider the following results, some of which have been mentioned previously.

(*i*) The binary representation of an integer n can be computed by a Turing machine given any input word of length n in time $\mathrm{O}(n^3)$;

(*ii*) For $k \geq 1$, a function computable by a Turing machine in space $\mathrm{O}(\log_2 n)^k$ is computable in polynomial time ([20], [22]);

(*iii*) A language recognizable by a Boolean circuit family of size n^q, where $q \geq 1$, is recognizable by a non-uniform Turing machine in time
$\mathrm{O}((n^q)(\log_2(n^q))^2) = \mathrm{O}((n^q)(q\log_2 n)^2) = \mathrm{O}(n^{q+2})$ ([20]).

Consider the Turing machine which, given an input word $\mathbf{w}$ of length n computes the binary representation of n, from which it generates the standard encoding of the circuit C_n of $\mathcal{C}$, and then "simulates $\mathcal{C}$" on input $\mathbf{w}$. This Turing machine recognizes the language L. The fact that $\mathcal{C}$ is *BC*-uniform, and the three results above, together imply that the Turing machine recognizes L in *polynomial time*. Thus, $\mathrm{L} \in \mathrm{P}$, from which it follows that $\mathrm{NC} \subseteq \mathrm{P}$. Since it is widely held that the class P contains all problems which can be solved using resources likely to be available ([12]), the fact that NC is contained in P means that the interest in NC is not only theoretical.

Consider the definition of the class AC^k. The absence of a fan-in restriction makes AC^k circuit families seem more powerful than NC^k circuit families. The following proposition indicates to what degree this is true.

Proposition 3.1.7 ([3]). *For all* $k \in \mathbb{N}$, $AC^k \subseteq NC^{k+1}$.

Proof. Let $n, k \in \mathbb{N}$, let $\mathrm{L} \in \mathrm{AC}^k$, and let $\mathcal{C} = (\mathrm{C}_n)_{n=0}^{\infty}$, be the AC^k circuit family which recognizes the language L. Consider the Boolean circuit C_n. If C_n has fan-in 2, then C_n is an NC^k circuit family, so that $\mathrm{L} \in \mathrm{NC}^k \subseteq \mathrm{NC}^{k+1}$. Otherwise C_n is of fan-in $m > 2$, and contains a gate ν labelled $\vee$ (or $\wedge$) with indegree m. Let $\mu_1, \mu_2, \ldots, \mu_m$ be the m gates which fan into ν. By Proposition 2.1.15, for each $m \in \mathbb{Z}^+$ there is a fan-in 2 Boolean circuit A_m of size $2m - 1$ and depth $\lceil \log_2 m \rceil$ which computes $x_1 \vee x_2 \vee \ldots \vee x_m$.

If we modify C_n by letting the gates $\mu_1, \mu_2, \ldots, \mu_m$ be input gates and ν be the output gate for the circuit A_m, we have increased the size of C_n by $m - 1$ and have increased the depth of C_n by $\lceil \log_2 m \rceil - 1$. Thus, if we repeat this construction for each gate of C_n with fan-in j such that $m \geq j > 2$, we obtain a fan-in 2 Boolean circuit B_n which recognizes the same language as does C_n, of size at most $m - 1$ times the size of C_n and depth at most $\lceil \log_2 m \rceil$ times the depth of C_n. Certainly m is less than the size of C_n. The family $\mathfrak{C}$ is in AC^k, so $\mathfrak{C}$ has size n^q for some $q \in \mathbb{N}$, and depth $(\log_2 n)^k$. This means that there exist $r_0, s, t \in \mathbb{N}$ such that for all $n \geq r_0$, C_n has size less than or equal to $s(n^q)$ and depth less than or equal to $t(\log_2 n)^k$. Consider the family $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ of fan-in 2 Boolean circuits where for each $l \in \mathbb{N}$, the circuit B_l is the modified version of the circuit C_l. For $n \geq r_0$, B_n has size less than or equal to $(s(n^q))^2$ and depth less than or equal to $\lceil \log_2 s(n^q) \rceil t(\log_2 n)^k = (\lceil \log_2 s \rceil + \lceil q \log_2 n \rceil)(t \lceil \log_2 n)^k$. By Proposition 2.1.28, $(\lceil \log_2 s \rceil + \lceil q \log_2 n \rceil)(t \lceil \log_2 n)^k = O((\log_2 n)^{k+1})$, so that $\mathfrak{B}$ has depth $(\log_2 n)^{k+1}$. Also by Proposition 2.1.28, $(s(n^q))^2 = O(n^{2q})$, so that $\mathfrak{B}$ has size n^{2q}, which is a polynomial. Thus the family $\mathfrak{B}$ is an NC^{k+1} circuit family. Since $\mathfrak{B}$ recognizes L, the language L is in NC^{k+1}. Since k and L were chosen arbitrarily, for all $k \in \mathbb{N}$, $AC^k \subseteq NC^{k+1}$. $\Box$

Recall the definitions of *BC*-uniform and *E**uniform. It has been shown ([21], [26]) that for $n \geq 2$, the classes (*BC*-uniform)NC^k and (*E**uniform)NC^k are the same, but that the class (*E**uniform)NC^1 is contained in the class (*BC*-uniform)NC^1, which suggests that *E**uniformity gives a sharper characterization of these complexity classes. This is why we included both definitions. After having taken the trouble to give uniformity conditions, we will no longer be concerned with them, safely assuming that any circuit family for which we can provide a description will be quite uniform.

From now on, we will focus on the class NC^1. For an interesting survey of problems which have been shown to be in NC, see [11], and the references therein. In Chapter II we constructed some circuits which give rise to NC^1 circuit families, so we have already seen some NC^1 functions and languages. Many interesting results concerning NC^1 have been obtained ([15], [1]). Of these results, we will consider new characterizations of this complexity class.

§ 3.2 *Barrington's Branching Programs and BWBP*

Recall from Chapter II that by "a branching program," we mean a levelled branching program where all sinks occur on the bottom level. When David Barrington began studying branching programs, he found that one of the reasons for the difficulty in comparing branching programs with other models for language recognition is that a branching program may depend on more that one variable at a given time in the computation. He made the following observation ([3]).

Proposition 3.2.1. *Let B be a levelled branching program on n variables of width w and length l, where each node on level k of B is assigned a distinct column number, an integer in $\{1,2,\ldots,w_k\}$, where w_k is the width of level k of B. Then B can be expanded to a levelled branching program B' of width at most $2w$ and length at most wl which computes the same function as does B, and is such that every level k of B' which is not the bottom level of sinks has all of its nodes labelled by the same input variable.*

Proof. We describe the expansion of level k of a branching program B on n variables of width w and length l, where $1 \leq k \leq l - 1$. Let p be the width of level k of B, where the j-th node on level k has label x_j, and let q be the width of level $k + 1$ of B, where the r-th node on level $k + 1$ has label y_r. We begin by replacing each of the labels x_1, x_2, ..., x_p of nodes on level k of B with the label x_1, and we insert a $(p - 1) \times (p + q)$ arrray of nodes between the relabelled level k and the original level $k + 1$ of B. The $p + q$ nodes in row m of the inserted array are labelled x_{m+1} for $1 \leq m \leq p - 1$.

We connect the first node labelled x_1 in the "relabelled" level k of B by an edge labelled 0 (1) to the node labelled x_2 in column i (j) of the inserted array if there is an edge labelled 0 (1) from the first node (labelled x_1) on level k to the i-th (j-th) node on level $k + 1$ of B. Of course, $i = j$ is allowed. For $s = 2, 3, \ldots p$, connect the s-th node labelled x_1 by an edge labelled 0, 1 to the $(q + s)$-th node labelled x_2 in the first row of the inserted array.

For $m = 2, 3, \ldots, p - 1$, if $n \neq q + m$, connect the n-th node labelled x_m in the $(m - 1)$-th row of the inserted array to the n-th node labelled x_{m+1} in row m of the inserted array by an edge labelled 0, 1. Connect the $(q + m)$-th node labelled x_m to the node labelled x_{m+1} in column i (j) of the m-th row of the inserted array by an edge labelled 0 (1) if there is an edge labelled 0 (1) from the m-th node on level k to the i-th (j-th) node on level $k + 1$ of B.

Connect the node labelled x_p in column $q + p$ of the bottom $(p - 1)$-th row of the inserted array by an edge labelled 0 (1) to the i-th (j-th) node in level $k + 1$ of B if there is an edge labelled 0 (1) from the p-th node on level k to the i-th (j-th) node on level $k + 1$ of B. For $n = 1, 2, \ldots, q$, connect the node labelled x_p in the n-th column of the last row of the inserted array to the n-th node in row $k + 1$ of B.

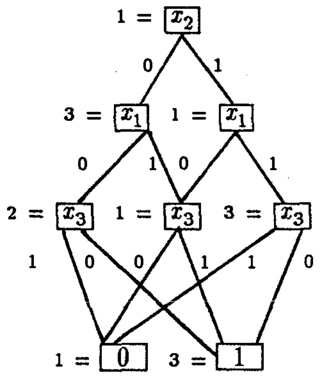
Let B′ be the levelled branching program obtained from B by expanding each level k of B, $1 \le k \le l-1$, as in the construction above, and then removing all nodes which cannot be reached by a path from the source. Clearly B′ has width at most $2w$ and length at most wl, and is such that every level but the bottom has all its nodes labelled by the same input variable. By construction, B′ computes the same function as that computed by B, since no branching occurs in B′ which does not reflect branching in B, and all branching occurring in B is reflected by branching in B′. In particular, if there is an edge labelled 0 (1) from a node x_i to a node x_p (x_q) in B, then in B′ there are edges labelled 0 and 1 from a node x_i to distinct (if $p \neq q$) nodes x_j, where x_j is either the node "next to" the node x_i or the first node on the level below the node x_i in B; from x_j there is a straight path to x_p (x_q), with no further intermediate branching, and thus the function computed by B′ is the same as that computed by B. □

We recall that a language L is in BWBP if and only if the Boolean function $f: \{0,1\}^* \to \{0,1\}$ defined by $f(u) = 1$ if and only if $u \in L$ is computable by a polynomial size family of bounded width branching programs. Proposition 3.2.1 shows that if a language over $\{0,1\}$ is recognizable by a family of branching program of bounded width and polynomial size, then it is recognizable by a bounded width, polynomial size family of branching programs where each branching program in the family is such that all the nodes on a given level are labelled by the same variable. Thus, the class BWBP is preserved, even if we change the definition of a branching program so that it excludes branching programs which contain nodes on the same level labelled by different variables. From now on, by "a branching program," we will mean a branching program such that all nodes on the same level are labelled by the same variable.

Consider the following. Let B be a branching program on n variables of width w and length l. For each $j \in \{0,1,\ldots,l\}$ we assign a different integer $k \in \{1,2,\ldots,w\}$ to each node on level j of B, reserving 1 for the source. We then think of B as an array of nodes with w columns and $l+1$ rows. Suppose that the source of B, labelled x_t, is such that the edge labelled 0 feeds into a node assigned m, and the edge labelled 1 feeds into a node assigned r. Then if $x_t = 1$, that is, when the t-th letter of the input word is 1, then the first level of B can be said to "yield" the function $f_1 \in M_w$ given by $1 \mapsto r$, and $p \mapsto p$ for all $p \in [w]$, $p \neq 1$. If $x_t = 0$, that is, when the t-th letter of the input word is 0, then the first level of B can be said to "yield" the function $g_1 \in M_w$ given by $1 \mapsto m$, and for all other $p \in [w]$, $p \mapsto p$. Once we have assigned a different integer in $[w]$ to each node on the same level of B, then each level k of B can be said to "yield" the function

f_k $(g_k) \in M_w$ if the variable x_i which labels the nodes in level $k-1$ of B is 1 (0), where $f_k(i) = j$ $(g_k(i) = j)$ when there is an edge labelled 1 (0) from the node assigned the integer i on level $k-1$ of the array into the node assigned j on level k of the array. Then for each input word in $\{0,1\}^n$, B "yields" the composition $h_l \circ \cdots \circ h_2 \circ h_1$, where for each $k \in \{1,2,\ldots,l\}$, $h_k \in M_w$ is the function "yielded" by level k of B when the variable x_i which labels the $(k-1)$-th row of B takes on the value of the i-th letter of the input word. For each word w in the language recognized by B, we put h(1), where h = B(w) is the function in M_w yielded by B on input w into an "accepting" subset of $[w]$. We may then define the language recognized by B to be the set of all words w such that on input w, B yields a function $h \in M_w$ such that h(1) is in the "accepting" subset of $[w]$.

Example 3.2.2. Consider the branching program B on 3 variables of width 3 and length 3 below, where each node of B is assigned an address.



B recognizes the language {000, 011, 101, 110}. According to the addresses assigned to the nodes of B, we see that if $x_2 = 1$, level 1 of B yields the function $f_1 \in M_3$ given by $1 \mapsto 1$, $2 \mapsto 2$, $3 \mapsto 3$, and if $x_2 = 0$, level 1 of B yields the function $g_1 \in M_3$ given by $1 \mapsto 3$, $2 \mapsto 2$, $3 \mapsto 3$. If $x_1 = 1$, level 2 of B yields the function $f_2 \in M_3$ given by $1 \mapsto 3$, $2 \mapsto 2$, $3 \mapsto 1$. Similarly, we find that g_2 is the function $1 \mapsto 1$, $2 \mapsto 2$, $3 \mapsto 2$, $f_3 : 1 \mapsto 3$, $2 \mapsto 1$, $3 \mapsto 1$, and g_3 is the function $1 \mapsto 1$, $2 \mapsto 3$, $3 \mapsto 3$. We see then that B(000) yields the composition $g_3 \circ g_2 \circ g_1 = k_1$, where $k_1(1) = k_1(2) = k_1(3) = 3$, B(001) yields $f_3 \circ g_2 \circ g_1 = h_1$, where $h_1(1) = h_1(2) = h_1(3) = 1$. B(010) yields $g_3 \circ g_2 \circ f_1 = g_3$, B(100) yields the composition $g_3 \circ f_2 \circ g_1 = k_2$, where $k_2(1) = k_2(3) = 1$, and $k_2(2) = 3$. B(011) yields $f_3 \circ g_2 \circ f_1 = f_3$, B(101) yields $f_3 \circ f_2 \circ g_1 = k_3$, where $k_3(1) = k_3(3) = 3$ and $k_3(2) = 1$. B(110) yields $h_2 = g_3 \circ f_2 \circ f_1$, where $h_2(3) = 1$ and $h_2(1) = h_2(2) = 3$, and B(111) yields $f_3 \circ f_2 \circ f_1 = h_3$, where $h_3(3) = 3$, and $h_3(1) = h_3(2) = 1$. We notice that B(w)(1) = 3 if and only if w ∈ {000, 011, 101, 110}, which is the language recognized by

B.

The discussion and the example show that we may as well think of a branching program B on n variables of width w and length l as a sequence of l "instructions," where each instruction selects one of two possible elements of M_w, depending on the value of a particular variable. The branching program B determines the composition of the yields of its instructions, each of which depends on the value assigned to a particular variable by the input word. This brings us to David A. Barrington's version of the branching program model for language recognition.

Definition 3.2.3 ([3], [4]). Let $n, w, l \in \mathbb{Z}^+$. A B-branching program B on n variables of width w and length l (abbreviated w-BBP) is a sequence of instructions $\langle j_i, f_i, g_i \rangle$, $1 \leq i \leq l$, such that for all $i \in \{1,2,\ldots,l\}$, $j_i \in \{1,2,\ldots,n\}$, and $f_i, g_i \in M_w$. Given a word $u \in \{0,1\}^n$, the instruction $\langle j_i, f_i, g_i \rangle$ "yields" f_i if the j_i-th letter of u is 1, and the instruction yields g_i if the j_i-th letter of u is 0. On input u, the w-BBP B yields the composition $h = h_l \circ \cdots \circ h_2 \circ h_1$ where h_k is the yield of the instruction $\langle j_k, f_k, g_k \rangle$ on input u; that is $B(u) = h$.

It is clear that a B-branching program B on n variables of width w and length l can be viewed as an array of nodes with w columns and $l+1$ rows, where nodes in row 0 are labelled by the variable x_{j_1}, nodes in row $k-1$ of the array are labelled by the variable x_{j_k}, and sinks in row $l+1$ are labelled by the integers $1, 2,\ldots,w$, with edges labelled 0 and 1 connecting nodes in row k to row $k+1$, according to the functions f_k and g_k contained in the instruction $\langle j_k, f_k, g_k \rangle$.

In contrast to the branching programs we looked at in Chapter II, we cannot simply label some of the sinks 0, and some of the sinks 1, and define the language recognized by a B-branching program B to be all words which determine a path in B which ends at a sink labelled 1. This is because a B-branching program has w source nodes, so an input word may define as many as w paths in B, so it could happen that for any choice of sink labels we might make, there is some word in $\{0,1\}^n$ which determines one path in B which ends at a sink labelled 0, and another path which ends at a sink labelled 1. However, for every $j \in [w]$, each $u \in \{0,1\}^n$ does determine a unique path in B, starting from j, that is, $B(u)(j)$, the sink ending the path in B determined by u starting from the node labelled x_{j_1} in row 0, column j, is well defined. This fact suggests the following as a definition of the language recognized by a B-branching program.

Definition 3.2.4 ([3]). Let B be a w-BBP on n variables, and let $A \subseteq [w]$. Then the

<u>language recognized by</u> B <u>with accepting set</u> A is the language $\{u \in \{0,1\}^n : B(u)(1) \in A\}$.

<u>Definition</u> <u>3.2.5</u>. Let $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ be a family of B-BPs such that for each $n \in \mathbb{N}$, B_n is a B-branching program on n variables of width w_n, and for each $n \in \mathbb{N}$, let $A_n \subseteq [w_n]$. Then the <u>language recognized by</u> $\mathfrak{B}$ <u>with family</u> $\mathcal{A} = (A_n)_{n=1}^{\infty}$ of <u>accepting sets</u> is the language $\{u \in \{0,1\}^* : |u| = n \text{ and } B_n(u)(1) \in A_n\}$. A language $L \subseteq \{0,1\}^*$ is said to be <u>recognizable by a family</u> of BBPs if and only if for each $n \in \mathbb{N}$, there is a B-branching program B_n on n variables of width w_n and a subset A_n of $[w_n]$ such that $B(u)(1) \in A_n$ if and only if $u \in L \cap \{0,1\}^n$. For each $k \in \mathbb{Z}^+$ there is a B-branching program B_0^k with no instructions such that $B_0^k(\varepsilon) = k$.

Let $z, w \in \mathbb{Z}^+$. We remark that if $z < w$, then any element of M_z can be extended to a function from $[w]$ to $[w]$, so that any B-branching program of width z can be thought of as a B-branching program of width w for all $w \geq z$. As with the other models we've discussed, we need to define the "resources" of a family of B-branching programs so that we can define a complexity class of languages in terms of bounds on those resources.

<u>Definition</u> <u>3.2.6</u>. Let $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ be a family of B-branching programs, where for each $n \in \mathbb{N}$, B_n is a BBP on n variables. Let f be a function from $\mathbb{N}$ to $\mathbb{R}$, and for each $n \in \mathbb{N}$, let $\lambda(n)$ denote the length of B_n, and let $\omega(n)$ denote the width of B_n. We say that the family $\mathfrak{B}$ has <u>length</u> $f(n)$ if $\lambda(n) = O(f(n))$. We say that the family $\mathfrak{B}$ has <u>bounded width</u> if for some $w \in \mathbb{Z}^+$, $\omega(n) \leq w$ for all $n \in \mathbb{N}$. By the above comment, bounded width is equivalent to constant width.

<u>Definition</u> <u>3.2.7</u>. CWBP is the class of languages over $\{0,1\}$ which are recognizable by a constant width polynomial length family of B-branching programs.

<u>Proposition</u> <u>3.2.8</u> ([3]). *CWBP = BWBP.*

<u>Proof</u>. Proposition 2.2.5 shows that any language recognizable by a polynomial length family of branching programs is recognizable by a polynomial length family of levelled branching programs. Proposition 3.2.1 shows that any language recognizable by a polynomial length bounded width family of levelled branching programs is recognizable by a polynomial length bounded width family of levelled branching programs such that all nodes on the same level are labelled by the same input variable. It is clear from discussion in this section that any language recognizable by a bounded width polynomial length family of such programs is recognizable by a constant width polynomial length family of BBPs. Therefore, BWBP is contained in CWBP. Conversely, suppose $L \subseteq \{0,1\}^n$ is recognizable by a w-BBP B of

length l with accepting set $A \subseteq [w]$. Consider the array of nodes which represents B. Let D be the set of nodes which are connected by edges to the node labelled x_{j_1} in row 0, column 1 of the array. Clearly D is a branching program of width at most w and length l, except that sinks carry labels in $[w]$. But the language recognized by B is the set of $u \in \{0,1\}^n$ such that $B(u)(1) \in A$. Therefore, when we assign the label 1 to each sink with a label in A, and assign the label 0 to all other sinks, we obtain a branching program D of width at most w and length l such that $D(u) = 1$ if and only if $u \in L$. It follows that any language recognizable by a bounded width polynomial length family of B-branching programs is recognizable by a bounded width polynomial length family of branching programs. Therefore, CWBP is contained in BWBP, and thus BWBP = CWBP. □

Having established the equivalence between the two branching program definitions, we will see in the next section how Barrington's version facilitates the establishment of a connection between branching programs and programs for a NUDFA over a finite monoid.

§ 3.3 *Branching Programs and Programs for a NUDFA over a Finite Monoid*

Recall the definition of the language recognized by a B-branching program of width w with accepting set $A \subseteq [w]$. There are, of course, other possible definitions.

Definition 3.3.1 ([3]). Let $L \subseteq \{0,1\}^n$, $n \in \mathbb{N}$, $w \in \mathbb{Z}^+$, and let B be a w-BBP on n variables. If there is a subset M of M_w such that for all $u \in \{0,1\}^n$, $B(u) \in M$ if and only if $u \in L \cap \{0,1\}^n$, then B is said to weakly recognize L. If $f, g \in M_w$ are such that $B(u) = f$ if and only if $u \in L \cap \{0,1\}^n$, and otherwise $B(u) = g$, then B is said to strongly recognize L. For each $f \in M_w$, there is a B-branching program B_0^f with no instructions such that $B_0^f(\varepsilon) = f$. A language $L \subseteq \{0,1\}^*$ is recognizable by a family of width w B-branching programs if and only if for all $n \in \mathbb{N}$, $L \cap \{0,1\}^n$ is weakly recognizable by a w-BBP.

Example 3.3.2. The B-branching program obtained from the branching program in Example 3.2.2 weakly recognizes the language {000, 011, 101, 110}, since $u \in L$ if and only if $B(u)(1) = 3$, if and only if $B(u) \in \{k_1, f_3, k_3, h_2\}$. Let B be the 2-BBP on three variables with instructions $\langle 2, f_0, f_1\rangle$, $\langle 3, f_1, f_1\rangle$ where $f_0 : 1 \mapsto 1, 2 \mapsto 1$, and $f_1 : 1 \mapsto 1, 2 \mapsto 2$. Then B strongly recognizes the language {000, 001, 100, 101}, since $B(u) = f_1$ if and only if $u \in L$, and otherwise $B(u) = f_0$. Note that B also strongly recognizes the complement of L.

Recall (Definition 2.3.4) that a language $L \subseteq \{0,1\}^n$ is recognizable by a program for a NUDFA over a finite monoid if and only if there is a finite monoid M, a subset B of M, and an n-program P_n for N(M,{0,1}) such that $P_n(u) \in B$ if and only if $u \in L$. By Proposition 1.1.2, every finite monoid M such that $c(M) = w$ is isomorphic to a submonoid of M_w, and this fact helps to show the following.

Proposition 3.3.3 ([3], [5]). *A language $L \subseteq \{0,1\}^*$ is recognizable by a family of programs for a NUDFA over a finite monoid if and only if there is a $w \in \mathbb{Z}^+$ such that for each $n \in \mathbb{N}$, $L \cap \{0,1\}^n$ is weakly recognizable by a branching program of width w.*

Proof. Let $L \subseteq \{0,1\}^*$ be recognizable by the family $\mathcal{P} = (P_n)_{n=0}^{\infty}$ of programs for the NUDFA $N(M_w,\{0,1\})$, where for each $n \in \mathbb{N}$, the subsets A_n, $A_n \subseteq M_w$ are chosen such that $P_n(u) \in A_n$ if and only if $u \in L \cap \{0,1\}^n$. Let $n \in \mathbb{N}$. If $n = 0$, let B_0^f be the B-branching program with no instructions such that if $\varepsilon \in L$ (if $\varepsilon \notin L$), then $B_0^f(\varepsilon) = f$ for some $f \in S_0$ ($f \notin S_0$). Otherwise, consider the n-program P_n with sequence of instructions $\langle j_k, f_k\rangle$, $1 \leq k \leq l$, where $f_k : \{0,1\} \to M_w$. Let B_n be the B-branching program with sequence of instructions $\mu_k = \langle j_k, f_k(1), f_k(0)\rangle$, $1 \leq k \leq l$. Since f_k is a function from $\{0,1\}$ to M_w, $f_k(1)$ and $f_k(0)$ are in M_w for $1 \leq k \leq l$, B_n is a w-BBP. Then for all $u \in \{0,1\}^n$, $u = u_1u_2\cdots u_n$, $B_n(u) \in A_n$ if and only if $h_l \circ\cdots\circ h_2 \circ h_1 \in A_n$, where for each k, h_k is the yield of the instruction μ_k. Since μ_k yields $f_k(1)$ if $u_{j_k} = 1$, and μ_k yields $f_k(0)$ if $u_{j_k} = 0$, then for all $k \in \{1,2,\ldots,l\}$, μ_k yields $f_k(u_{j_k})$. Therefore, $B_n(u) = h_l \circ\cdots\circ h_2 \circ h_1 = f_l(u_{j_l}) \circ\cdots\circ f_2(u_{j_2}) \circ f_1(u_{j_1}) = P_n(u)$ for all $u \in \{0,1\}^n$, so that $B_n(u) \in A_n$ if and only if $P_n(u) \in A_n$, that is, if and only if $u \in L \cap \{0,1\}^n$. Therefore, B_n weakly recognizes $L \cap \{0,1\}^n$. Since n was arbitrary, the language L is recognizable by the family $\mathcal{B} = (B_n)_{n=0}^{\infty}$ of width w branching programs.

Conversely, let $L \subseteq \{0,1\}^*$ be recognizable by the family $\mathcal{B} = (B_n)_{n=0}^{\infty}$ of width w branching programs, and let $A_n \subseteq M_w$ be such that $B_n(u) \in A_n$ if and only if $u \in L \cap \{0,1\}^n$. Let $n \in \mathbb{N}$. If $n = 0$, let P_0^g be the 0-program for $N(M_w,\{0,1\})$ such that if $\varepsilon \in L$ (if $\varepsilon \notin L$), then $P_0^g(\varepsilon) = g$ where $g \in A_n$ ($g \notin A_n$). Otherwise, consider the w-BBP B_n with sequence of instructions $\nu_k = \langle j_k, f_k, g_k\rangle$, $1 \leq k \leq l$, where $f_k, g_k \in M_w$. For each $k \in \{1,2,\ldots,l\}$, let h_k be the function from $\{0,1\}$ to the finite monoid M_w defined by $h_k(1) = f_k$, $h_k(0) = g_k$. Let P_n be the n-program for the NUDFA $N(M_w,\{0,1\})$ with sequence of instructions $\langle j_k, h_k\rangle$; $1 \leq k \leq l$, where $j_k \in \{1,2,\ldots,n\}$. Then for $u \in \{0,1\}^n$, $u = u_1u_2\cdots u_n$, $P_n(u) \in A_n$ if and only if $h_l(u_{j_l}) \circ\cdots\circ h_2(u_{j_2}) \circ h_1(u_{j_1}) \in A_n$. By definition of h_k, h_k is the yield of the instruction ν_k on input u, so that $P_n(u) = B_n(u)$. Therefore, $P_n(u) \in A_n$ if and only if $B(u) \in A_n$, that is, if and only if $u \in L \cap \{0,1\}^n$. Again, n arbitrary implies that the finite monoid

M_w, the family $\mathcal{P} = (P_n)_{n=0}^{\infty}$ of programs for the NUDFA $N(M_w,\{0,1\})$, and the subsets $A_n \subseteq M_w$ are such that L is recognizable by a family of programs for a NUDFA over a finite monoid. ☐

<u>Corollary</u> <u>3.3.4</u>. *BWBP = PLP.*

<u>*Proof.*</u> Let L ∈ BWBP. Since BWBP =CWBP by Proposition 3.2.8, for some $w \in \mathbb{Z}^+$ there is a polynomial length family $\mathcal{B} = (B_n)_{n=0}^{\infty}$ of w-BBPs which weakly recognizes L. Then there is a family of accepting sets $(A_n)_{n=0}^{\infty}$ such that for each $n \in \mathbb{N}$, $B_n(u) \in A_n \subseteq M_w$ if and only if $u \in L \cap \{0,1\}^n$. The family $\mathcal{B}$ is polynomial size means that there exist q, c, $r_0 \in \mathbb{N}$ such that for all $n \geq r_0$, $\lambda(n) \leq cn^q$. By Proposition 3.3.3, for each $n \in \mathbb{N}$ there is an n-program P_n of the same length as B_n such that $P_n(u) \in A_n$ if and only if $u \in L \cap \{0,1\}^n$. Consider the family $\mathcal{P} = (P_n)_{n=0}^{\infty}$ of programs for $N(M_w,\{0,1\})$. Clearly L is recognized by the family $\mathcal{P}$ for the NUDFA over the finite monoid M_w with alphabet $\{0,1\}$. For all $n \geq r_0$, the length of B_n which is the length of P_n is less than or equal to cn^q, so that $\mathcal{P}$ has length cn^q. Therefore, L is recognizable by a polynomial length family of programs for a NUDFA over a finite monoid, so that L ∈ PLP. The language L was arbitrary, which implies that BWBP ⊆ PLP. A similar argument shows that PLP ⊆ BWBP, from which it follows that BWBP = PLP. ☐

In the next section we will see how families of branching programs and families of programs for a NUDFA over a finite monoid relate to families of Boolean circuits and the complexity class NC^1.

§ 4 *Bounded Width Branching Programs and NC^1*

Having modified the definition of a branching program, Barrington continued the work of others ([10]), looking at branching programs of small width, hoping that his new definition would provide some new insight. Given the new definition, one idea readily comes to mind. The idea is that of considering branching programs such that every function contained in the instructions is a permutation. In the study of such branching programs, one can employ the power of group theory.

<u>*Definition*</u> <u>3.4.1</u> ([3], [2]). Let n, w, $l \in \mathbb{Z}^+$. A <u>permutation</u> <u>branching</u> <u>program</u> B of width w and length l (abbreviated w-PBP) is a w-BBP such that every function in an instruction of B is a permutation of the set $[w]$.

Barrington investigated the power of permutation branching programs of widths 2, 3, and 4 ([3], [2]). We present here his analysis of width 5 permutation branching programs.

Definition 3.4.2 ([3], [4]). Let $L \subseteq \{0,1\}^n$, and let B be a 5-PBP. B is said to 5-cycle recognize L if there is a five cycle $\sigma \in S_5$ such that $B(u) = \sigma$ if $u \in L$, and $B(u) = e$ if $u \notin L$, where e is the identity of S_5. We call σ the output of B, and we say that B recognizes L with output σ.

Lemma 3.4.3 ([3], [4]). *Let $L \subseteq \{0,1\}^n$, and let B be a 5-PBP which recognizes L with output the 5-cycle σ. Then for any 5-cycle τ in S_5, there is a 5-PBP which recognizes L with output τ.*

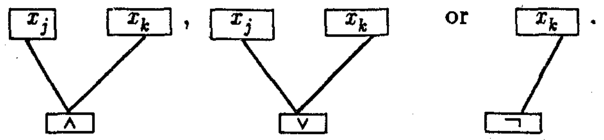
Proof. Let B be the 5-PBP with sequence of instructions $\nu_k = \langle j_k, \alpha_k, \beta_k \rangle$, $1 \leq k \leq l$, which 5-cycle recognizes L with output σ. Let $\theta \in S_5$ be such that $\theta \circ \sigma \circ \theta^{-1} = \tau$. Let $k \in \{1,2,\ldots,l\}$, and consider μ_k, where $\mu_k = \langle j_k, \theta \circ \alpha_k \circ \theta^{-1}, \theta \circ \beta_k \circ \theta^{-1} \rangle$. It is clear that the instruction μ_k yields $\theta \circ \gamma_k \circ \theta^{-1}$, where γ_k is the yield of the instruction ν_k. Let B′ be the permutation branching program of width 5 and length l with sequence of instructions $\langle j_k, \theta \circ \alpha_k \circ \theta^{-1}, \theta \circ \beta_k \circ \theta^{-1} \rangle$, $1 \leq k \leq l$. Then for $u \in \{0,1\}^n$, $B'(u) = (\theta \circ \gamma_l \circ \theta^{-1}) \circ \cdots \circ (\theta \circ \gamma_2 \circ \theta^{-1}) \circ (\theta \circ \gamma_1 \circ \theta^{-1})$, which is equal to $\theta \circ \gamma_l \circ \cdots \circ \gamma_2 \circ \gamma_1 \circ \theta^{-1}$ whenever $B(u) = \gamma_l \circ \cdots \circ \gamma_2 \circ \gamma_1$. If $u \in L$, then, since B 5-cycle recognizes L with output σ, $B(u) = \sigma$, so that $B'(u) = \theta \circ \sigma \circ \theta^{-1} = \tau$. If $u \notin L$, then $B(u) = e$, so that $B'(u) = \theta \circ e \circ \theta^{-1} = e$. Therefore, B′ is a 5-PBP of the same length as B which 5-cycle recognizes L with output τ. □

Lemma 3.4.4 ([3], [4]). *If $L \subseteq \{0,1\}^n$ is 5-cycle recognized by a 5-PBP of length l, then the complement of L is 5-cycle recognized by a 5-PBP of length l.*

Proof. Suppose that L is 5-cycle recognized by the 5-PBP B of length l with output σ, and that the last instruction of B is $\langle j_k, \alpha, \beta \rangle$. Let B′ be the 5-PBP of length l such that the first $l-1$ instructions of B′ are the first $l-1$ instructions of B, and the last instruction of B′ is the instruction $\langle j_k, \sigma^{-1} \circ \alpha, \sigma^{-1} \circ \beta \rangle$. Then for all $u \in \{0,1\}^n$, $B(u) = \gamma_l \circ \cdots \circ \gamma_1$ implies that $B'(u) = \sigma^{-1} \circ \gamma_l \circ \cdots \circ \gamma_1$. If $u \in L$, $B(u) = \sigma$, so that $B'(u) = \sigma^{-1} \circ \sigma = e$. If $u \notin L$, $B(u) = e$, so that $B(u) = \sigma^{-1} \circ e = \sigma^{-1}$, which is a 5-cycle. Therefore, B′ is a 5-PBP of length l which recognizes the complement of L with output σ^{-1}. □

Theorem 3.4.5 ([3], [4]). *Let $L \subseteq \{0,1\}^n$ be such that there is a fan-in 2 Boolean circuit C on n variables of depth d such that $C(u) = 1$ if and only if $u \in L$. Then there is a 5-PBP B of length at most 4^d which 5-cycle recognizes L.*

Proof. We proceed by induction on d. If $d = 1$, then C is one of the circuits below.



If C is the circuit with the $\wedge$ gate, then let B be the 5-PBP of length 4 with sequence of instructions $\langle j, (12345), e\rangle$, $\langle k, (13542), e\rangle$, $\langle j, (15432), e\rangle$, $\langle k, (12453), e\rangle$. Let $u \in \{0,1\}^n$. If u is an element of L, then $u_j = 1$ and $u_k = 1$, so that $B(u) = (12453) \circ (15432) \circ (13542) \circ (12345) = (13254)$. If $u \notin L$, then one of the following holds. Either

(i) $u_j = 1$ and $u_k = 0$, so that $B(u) = (15432) \circ (12345) = e$,

(ii) $u_j = 0$ and $u_k = 1$, so that $B(u) = (12453) \circ (13542) = e$, or

(iii) $u_j = 0$ and $u_k = 0$, so that $B(u) = e$.

Therefore, B recognizes L with output (13254). If C is the circuit with the $\vee$ gate, let B be the 5-PBP of length 4 with sequence of instructions $\langle j, e, (12345)\rangle$, $\langle k, e, (13542)\rangle$, $\langle j, e, (15432)\rangle$, $\langle k, (14523), \sigma \circ (12453)\rangle$, where $\sigma = (14523)$ is the inverse of (13254). It is clear that for all $u \in \{0,1\}^n$, $B(u) = \varepsilon$ if both of u_j, u_k are 0, and $B(u) = \sigma$ otherwise. Therefore, B recognizes L with output $\sigma = (14523)$. If C is the circuit with the $\neg$ gate, let B be the 5-PBP with the single instruction $\langle k, \varepsilon, (12345)\rangle$. Then it is clear that B recognizes L with output (12345). Assume that for some $k \geq 1$, if L is recognizable by a fan-in 2 Boolean circuit of depth $d \leq k$, then there is a 5-PBP of length at most 4^d which 5-cycle recognizes L. Let L be recognizable by a fan-in 2 Boolean circuit C of depth $k + 1$. We consider 3 cases

(i) C is the "and" of two fan-in 2 Boolean circuits A and D, each of depth less than or equal to k;

(ii) C is the "or" of two fan-in 2 Boolean circuits A and D, each of depth at most k;

(iii) C is the "not" of a fan-in 2 Boolean circuit A of depth k.

By the induction hypothesis, the languages L_1 and L_2 recognized by the fan-in 2 Boolean circuits A and D can be 5-cycle recognized by 5-PBPs of length at most 4^k.

Consider case (i), where $L = L_1 \cap L_2$. By Lemma 3.4.3, there is a 5-PBP B_a which recognizes the language L_1 with output $\sigma = (12345)$, and there is a 5-PBP B_d which recognizes the language L_2 with output $\tau = (13542)$, each of length at most 4^k. Also by Lemma 3.4.3, there is a 5-PBP $\overline{B}_a$ which 5-cycle recognizes the language L_1 with output $(15432) = \sigma^{-1}$, and there is a 5-PBP $\overline{B}_d$ which 5-cycle recognizes the language L_2 with output $(12453) = \tau^{-1}$ each of length at most 4^k. Let B be the 5-PBP of length at most

$4(4^k) = 4^{k+1}$ whose sequence of instructions is the sequence of the instruction sequences of B_a, B_d, $\overline{B}_a$, $\overline{B}_d$. Then, for all $u \in \{0,1\}^n$, $B(u)$ is equal to $\overline{B}_d(u) \circ \overline{B}_a(u) \circ B_d(u) \circ B_a(u)$. Let $u \in \{0,1\}^n$. Then $u \in L_1$ implies that $B_a(u) = \sigma$ and $\overline{B}_a(u) = \sigma^{-1}$; and $u \in L_2$ implies that $B_d(u) = \tau$ and $\overline{B}_d(u) = \tau^{-1}$. Therefore, $u \in L_1 \cap L_2 = L$ implies that $B(u) = \tau^{-1} \circ \sigma^{-1} \circ \tau \circ \sigma$ which equals $(13254) = \rho$. If $u \notin L_1$, then $B_a(u) = e$ and $\overline{B}_a(u) = e$, so that $B(u) = e$ or $B(u) = \tau^{-1} \circ \tau = e$. Similarly, $u \notin L_2$ also implies that $B(u) = \varepsilon$. Therefore, B is a 5-PBP of length at most 4^{k+1} which 5-cycle recognizes L with output ρ .

Consider case (*iii*). L is the complement of the language L_1, so by the induction hypothesis, L_1 is 5-cycle recognizable by a 5-PBP of length at most 4^k. By Lemma 3.4.4, the complement L of L_1 is 5-cycle recognizable by a 5-PBP B of length at most 4^k.

Consider case (*ii*), where $L = L_1 \cup L_2$. By the induction hypothesis and Lemma 3.4.4, there are 5-PBPs which 5-cycle recognize the complements of the languages L_1 and L_2, each of length at most 4^k. By case (*i*) there is a 5-PBP B of length at most 4^{k+1} which 5-cycle recognizes the language $\overline{L}_1 \cap \overline{L}_2$, where $\overline{L}_1$ and $\overline{L}_2$ are the complements of the languages L_1 and L_2. Again by Lemma 3.4.4, there is a 5-PBP of length at most 4^{k+1} which 5-cycle recognizes $L = \overline{\overline{L}_1 \cap \overline{L}_2}$, the complement of $\overline{L}_1 \cap \overline{L}_2$. ☐

Corollary 3.4.6 ([3], [21]). *$NC^1 \subseteq$ 5-PBP $\subseteq$ BWBP.*

Proof. Let $L \in NC^1$. Then there is a family $\mathcal{C} = (C)_{n=0}^{\infty}$ of fan-in 2 Boolean circuits of polynomial size and depth $\log_2 n$ which recognizes L. That $\mathcal{C}$ has depth $\log_2 n$ implies that there exist integers $k, c \in \mathbb{Z}^+$ such that for all $n \geq k$, $\delta(n) \leq c\log_2 n$. By Theorem 3.4.5, for each $n \in \mathbb{N}$, there is a 5-PBP B_n of length at most $4^{\delta(n)}$ which 5-cycle recognizes $L \cap \{0,1\}^n$. Consider the family $\mathcal{B} = (B_n)_{n=1}^{\infty}$ of 5-PBPs which recognizes L. For $n \geq k$, $\lambda(n) \leq 4^{c\log_2 n} = n^{2c}$. Therefore, $\lambda(n) = O(n^{2c})$, and the language L is recognizable by a polynomial length family of permutation branching programs of width 5. Therefore, $L \in$ 5-PBP $\subseteq$ BWBP. The language L was arbitrary, so $NC^1 \subseteq$ 5-PBP $\subseteq$ BWBP. ☐

Theorem 3.4.7 ([3], [4]). *If $L \subseteq \{0,1\}^n$ is recognizable by a branching program of width w and length l, then L is recognizable by a fan-in 2 Boolean circuit of depth $O(\log_2 l)$, where the constant depends only on w.*

Proof. Let $L \subseteq \{0,1\}^n$ be recognized by the branching program B on n variables of width w and length l with sequence of instructions $\nu_k = \langle j_k, f_k, g_k \rangle$, $1 \leq k \leq l$, where $f_k, g_k \in M_w$. B recognizes L, so there is a subset $H = \{h_1, h_2, \ldots, h_t\} \subseteq M_w$ such that for all $u \in \{0,1\}^n$, $B(u) \in H$ if and only if $u \in L$.

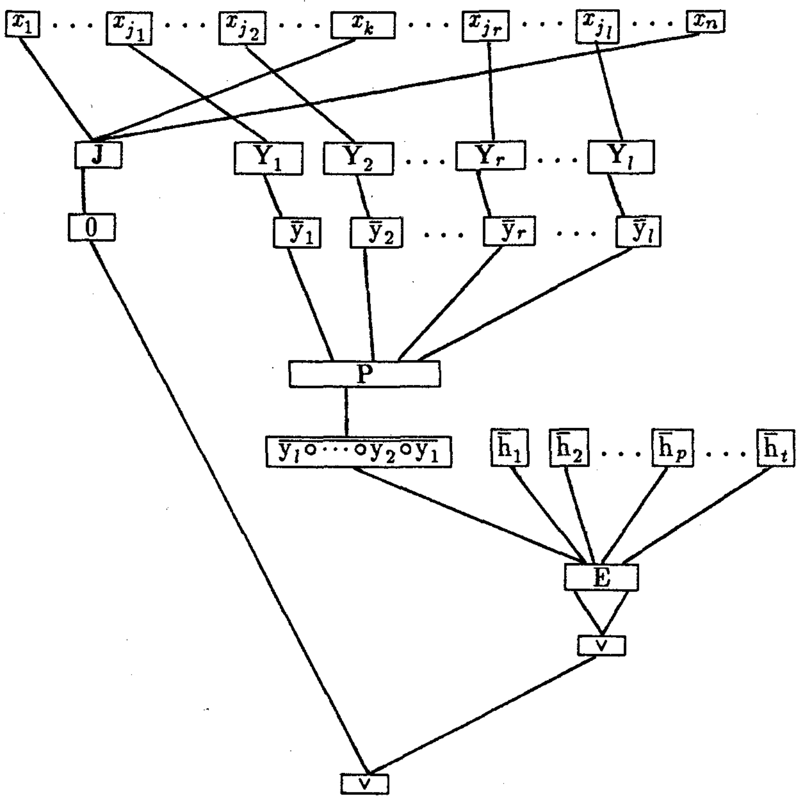
(i) By Corollary 2.1.24, for each $r \in \{1,2,\ldots,l\}$, there is a fan-in 2 Boolean circuit Z_r of depth 2 and size at most $w^2 + 3w + 1$ which will "determine the yield" y_r of the instruction ν_r, that is, which will, for any $u \in \{0,1\}^n$, output the Boolean representation of f_r if $u_{j_r} = 1$, and will output the Boolean representation of g_r if $u_{j_r} = 0$.

(ii) By Lemma 2.1.22, there is a fan-in 2 Boolean circuit Q of depth $\lceil \log_2 l \rceil(\lceil \log_2 w \rceil + 1)$ and size at most $(l-1)2w^3 + w^2$ which will "compute the product" $y_l \circ \cdots \circ y_2 \circ y_1$, that is, which will output the Boolean representation $\overline{y_l \circ \cdots \circ y_2 \circ y_1}$ of the product $y_l \circ \cdots \circ y_2 \circ y_1$.

(iii) By Corollary 2.1.18, there is a fan-in 2 Boolean circuit F of depth $\lceil \log_2 t \rceil + \lceil \log_2 w^2 \rceil + 3 = \lceil \log_2 t \rceil + \lceil 2(\log_2 w) \rceil + 3$ and size $7w^2t + w^2 - 1$ (where $t \leq w^w$), which "determines whether or not $y_l \circ \cdots \circ y_2 \circ y_1$ is in H," that is, whether or not $\overline{y_l \circ \cdots \circ y_2 \circ y_1} = \overline{h}_p$ for some h_p in H.

(iv) Let $I = \{1,2,\ldots,n\} \setminus \{j_1,j_2,\ldots,j_l\}$, and let $q \leq n$ denote c(I). If $q > 1$, then by Corollary 2.1.16, there is a fan-in 2 Boolean circuit K of size $2q$ and depth $\lceil \log_2 q \rceil + 1$ such that $K(u) = 0$ for all $u \in \{0,1\}^q$.

For each $r \in \{1,2,\ldots,l\}$ let Y_r be the circuit Z_r excluding input and output gates, let P be the circuit Q excluding input and output gates, let E be the circuit F excluding input and output gates, and let J be the circuit K excluding input and output gates. Consider the Boolean circuit C shown below, keeping in mind that the j_r are not necessarily distinct, and that j_r is not necessarily less than j_{r+1}, even though the diagram has been constructed to correspond to this case.



The fan-in 2 Boolean circuit C has depth $\lceil \log_2 q \rceil + 2$, or depth $2 + \lceil \log_2 l \rceil(\lceil \log_2 w \rceil + 1) + \lceil \log_2 t \rceil + \lceil 2(\log_2 w) \rceil + 3$, whichever is larger. Without taking into account that some gates belong to more than one subcircuit, C has size at most $n + 2q + l(w^2 + 3w + 1) + (l - 1)2w^3 + w^2 + 7w^2t + w^2 - 1 + 1$. If $0 \leq q \leq l$, then C has depth $O(\log_2 l)$ where the constant depends only on w. Let $u \in \{0,1\}^n$, $u = u_1u_2 \cdots u_n$. Then, by definition of C, $C(u) = 1$ if and only if $0 \vee F(\overline{y_l \circ \cdots \circ y_2 \circ y_1}\ \bar{h}_1\bar{h}_2 \cdots \bar{h}_t) = 1$, if and only if $F(Q(\bar{y}_1\bar{y}_2 \cdots \bar{y}_l)\bar{h}_1\bar{h}_2 \cdots \bar{h}_t) = 1$, if and only if $F(Q(Z_1(u_{j_1})Z_2(u_{j_2}) \cdots Z_l(u_{j_l}))\bar{h}_1\bar{h}_2 \cdots \bar{h}_t) = 1$. By definition of the circuits F, Q, and Z_r, $C(u) = 1$ if and only if the composition of the yields of the instructions $\nu_1, \nu_2, \ldots, \nu_l$ on input u is in H, that is, if and only if $B(u) \in H$. But $B(u) \in H$ if and only if $u \in L$. This implies that $C(u) = 1$ if and only if $u \in L$. Therefore, C recognizes L. □

<u>Corollary</u> <u>3.4.8</u> ([3], [4]). *BWBP* $\subseteq$ *NC*^{1}.

<u>*Proof.*</u> $L \in$ BWBP implies that there is a polynomial length family $\mathfrak{B} = (B_n)_{n=0}^{\infty}$ of B-branching programs of width $w \geq 1$ which recognizes L. Then for each $n \in \mathbb{N}$, there is a subset A_n of M_w such that for all $u \in \{0,1\}^n$, $B_n(u) \in A_n$ if and only if $u \in L \cap \{0,1\}^n$. $\mathfrak{B}$ is polynomial length, so there are constants $c \geq 0$, $a \geq 1$, and $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $\lambda(n) \leq an^c$. From the proof of Theorem 3.4.7, for each $n \in \mathbb{N}$,

there is a fan-in 2 Boolean circuit C_n of size at most $n + 2q_n + \lambda(n)(w^2 + 3w + 1) + (\lambda(n) - 1)2w^3 + w^2 + 7w^2|A_n| + w^2$, and depth the larger of $5 + \lceil\log_2\lambda(n)\rceil(\lceil\log_2 w\rceil + 1) + \lceil\log_2|A_n|\rceil + \lceil 2(\log_2 w)\rceil$ and $\lceil\log_2 q_n\rceil + 2$, which recognizes $L \cap \{0,1\}^n$, where q_n is the cardinality of the set of integers i, $1 \leq i \leq n$ which do not appear in the instructions for the w-BBP B_n. For all $n \in \mathbb{N}$, $q_n \leq n$, and $c(A_n) \leq c(M_w) = w^w$. Since B_n has size less than or equal to an^c for all $n \geq n_0$, for such n, the fan-in 2 Boolean circuit C_n has depth at most the larger of $\lceil\log_2 n\rceil + 2$ and $5 + \lceil\log_2 an^c\rceil(\lceil\log_2 w\rceil + 1) + \lceil w(\log_2 w)\rceil + \lceil 2(\log_2 w)\rceil$, and C_n has size at most $3n + (an^c)(w^2 + 3w + 1) + ((an^c) - 1)2w^3 + 2w^2 + 7w^{w+2}$. Since w is constant, by Proposition 2.1.28, $\delta(n) = O(\log_2 n)$. Also by Proposition 2.1.28, since w is constant, $\sigma(n) = O(n^c)$. Thus, the family $\mathcal{C} = (C_n)_{n=0}^{\infty}$ of fan-in 2 Boolean circuits has polynomial size and depth $\log_2 n$, and is such that for all $n \in \mathbb{N}$, $C_n(u) = 1$ if and only if $u \in L \cap \{0,1\}^n$. Therefore, L is in NC^1. The language L was chosen arbitrarily, so $BWBP \subseteq NC^1$. □

Theorem 3.4.9 *PLP = BWBP = 5-PBP =* NC^1.

Proof. Corollaries 3.3.4, 3.4.6, and 3.4.8. □

Conclusion

Theorem 3.4.9 tells us that the language classes defined by the four "different" non-uniform models are the same. However, when it was shown that CWBP is the same class as BWBP, one definition of the language recognized by a branching program was used, while another definition was used in order to show that CWBP is the same class as PLP. Consider the following. Let L be a language in NC^1. Then for all $n \in \mathbb{N}$, $L \cap \{0,1\}^n$ can be 5-cycle recognized by a 5-PBP B_n, by Theorem 3.4.5. This means, firstly, that for all $n \in \mathbb{N}$, $L \cap \{0,1\}^n$ is strongly recognized by B_n and, secondly, that for all $n \in \mathbb{N}$, there is an element k of $\{1,2,3,4,5\}$ such that $B_n(u)(1) = k$ if and only if $u \in L \cap \{0,1\}^n$. Thus for all $n \in \mathbb{N}$, $L \cap \{0,1\}^n$ is both weakly and strongly recognized by B_n (Definition 3.3.1), and $L \cap \{0,1\}^n$ is recognized by B_n as in Definition 3.2.4. Since NC^1 is BWBP, we see that BWBP remains the same regardless of the choice of definition for the language recognized by a branching program. We note that, although we presented the proof of Theorem 3.4.8 without taking uniformity into account, Barrington has shown that the result holds in the uniform case as well ([3], [4]).

We now provide a sampling of some continuing work in this area. Denis Therien ([25]) has introduced the concept of a variety of congruences to add to the well known correspondence between classes of languages and varieties of finite monoids ([13]). Combining this work, and the work that we have presented in this thesis, Barrington and Therien ([5]) were able to characterize AC^0 as the class of languages which are recognizable by a polynomial length family of programs for a NUDFA over an aperiodic monoid, and they give a similar characterization for another subclass of NC^1. Howard Straubing, Barrington, and Therien ([6]) have looked at programs for NUDFAs over groups. Straubing, Barrington, Therien, and Kevin Compton ([7]) have characterized regular languages in NC^1.

We hope that we have provided an interesting and helpful introduction to this fascinating and fruitful area of research.

References

[1] M. Ajtai: Σ_1^1 Formulae on Finite Structures (*Annals of Pure and Applied Logic* 1983, vol. 24, p. 1-48)

[2] D. A. Barrington: Width Three Permutation Branching Programs (MIT Technical Memorandum TM-293, 1985)

[3] D. A. Barrington: Bounded Width Branching Programs (Ph. D. Thesis, Massachusetts Institute of Technology 1986)

[4] D. A. Barrington: Bounded Width, Polynomial Size Branching Programs Recognize Exactly those Languages in NC^1 (*Proceedings,* 18*th ACM Symposium on the Theory of Computing* 1986, p. 1-5)

[5] D. A. Barrington and D. Therien: Finite Monoids and the Fine Structure of NC^1 (*Proceedings,* 19*th ACM Symposium on the Theory of Computing* 1987, p. 101-109)

[6] D. Barrington, H. Straubing, and D. Therien: Non-Uniform Automata Over Groups (University of Massachusetts, Amherst 1988, COINS Technical Report 88-77)

[7] D. Barrington, K. Compton, H. Straubing, and D. Therien: Regular Languages in NC^1 (Boston College Technical Report BCCS-88-02, 1988)

[8] A. Borodin: On Relating Time and Space to Size and Depth (*Siam Journal on Computing* 1977, vol. 6, p. 733-744)

[9] A. Borodin, M. Fischer, D. Kirkpatrick, N. Lynch, and M. Tompa: A Time Space Tradeoff for Sorting on Non-Oblivious Machines (*IEEE Symposium on the Foundations of Computer Science* 1979, p. 319-327)

[10] A. Borodin, D. Dolev, F. Fich, and W. Paul: Bounds for Width Two Branching Programs (*Siam Journal on Computing* 1986, vol. 15, p. 549-560)

[11] S. Cook: A Taxonomy of Problems with Fast Parallel Algorithms (*Information and Control* 1985, vol. 64, p. 2-22)

[12] M. Davis and E. Weyuker: *Computability, Complexity, and Languages* (Academic Press, Inc., New York 1983)

[13] S. Eilenberg: *Automata, Languages, and Machines* (Academic Press, Inc., New York, vol. A 1974, vol. B 1976)

[14] J. B. Fraleigh: *A First Course in Abstract Algebra* (3rd ed., Addison-Wesley, Inc. Philippines 1982)

[15] M. Furst, J. Saxe, and M. Sipser: Parity, Circuits, and the Polynomial Time Hierarchy (*Proceedings,* 22*nd IEEE Symposium on the Foundations of Computer Science* 1981, p. 260-270)

[16] J. Håstad: Lower Bounds in Computational Complexity Theory (*Notices of the American Mathematical Society* 1988, vol. 35, no. 5, p. 677-683)

[17] J. Hopcroft and J. Ullman: *Introduction to Automata Theory, Languages, and Computability* (Addison-Wesley, Inc., Mass. 1979)

[18] J. M. Howie: *An Introduction to Semigroup Theory* (Academic Press, Inc. London, Ltd. 1976)

[19] G. Lallement: *Semigroups and Combinatorial Applications* (John Wiley and Sons, Inc., New York 1979)

[20] N. Pippenger: On Simultaneous Resource Bounds (*Proceedings, 20th IEEE Symposium on the Theory of Computing* 1979, p. 307-311)

[21] W. Ruzzo: On Uniform Circuit Complexity (*Journal of Computer and Systems Sciences* 1981, vol. 22, p. 365-383)

[22] J. Savage: Computational Work and Time on Finite Machines (*Association for Computing Machinery Journal* 1972, vol. 19, p. 660-674)

[23] J. Savage: *The Complexity of Computing* (John Wiley and Sons, Inc., New York 1976)

[24] C. Schnorr: The Network Complexity and the Turing Machine Complexity of Finite Functions (*Acta Informatica* 1976, vol. 7, p. 95-107)

[25] D. Therien: Classification of Finite Monoids: The Language Approach (*Theoretical Computer Science* 1981, vol. 14, p. 195-208)

[26] I. Wegener: *The Complexity of Boolean Functions* (John Wiley and Sons Ltd, and B. G. Teubner, Stuttgart 1987)