Extensions of an Idea of McNaughton

David A. Mix Barrington[1]
University of Massachusetts
June 21, 1993

1. Abstract

Two important measures of the computational complexity of a regular language are the type of finite automaton needed to recognize it and the type of logical expression needed to describe it. Important connections between these measures were studied by Büchi and McNaughton as early as 1960. In this survey we describe the logical formalism used, outline these early results, and describe modern extensions of this work. In particular, we show how the formalism is extended by the use of new quantifiers and atomic predicates to express many of the fundamental classes of boolean circuit complexity.

2. Introduction

A formal language may be thought of as either a *subset* of the set of all strings or as a *property* which some strings have and some do not. A natural question about the latter notion is how easy or hard it is to *express* a given property in the language of logic. By placing a metric of some sort on this expressibility, we create a complexity theory, which we can then compare to the conventional theory, which measures how easy or hard it is for various abstract computational devices to *decide* whether a given string has the property.

Let A be any finite alphabet and $A^\star$ the set of all finite strings on A. Our basic logical language will have variables which range over *positions* in an input string. Our atomic predicates will be equality and order on positions ("$x = y$" and "$x < y$") and, for each $a \in A$, a predicate "$\pi_a(x)$" denoting "the input letter in position x is a". We can combine these formulas by the boolean connectives "$\wedge, \vee, \neg$" and bind variables

[1]Former name David A. Barrington. Supported by NSF Computer and Computation Theory grant CCR-8714714. Mailing address: Dept. of Computer and Information Science, U. of Mass., Amherst MA 01003, U.S.A.

by the first-order quantifiers "$\exists, \forall$" by the usual rules of logic. A formula with no free variables (a *sentence*) is either true or false of any particular string, and thus any sentence expresses the characteristic property of some language. For example, consider the language $aA^\star$. We must say "the first character is a", or equivalently "there is a position which is before all other positions and which contains an a", i.e.,

$$\exists x \forall y (\pi_a(x) \wedge ((x = y) \vee (x < y))).$$

This formalism was introduced in the late 1950's by Büchi, who considered the system we have described with the addition of *monadic second-order* quantifiers. These bind variables of a new type which range over *sets* of positions in the input. A new atomic formula $x \in Y$, where x is a position variable and Y a set variable, is added to denote "x is a member of y". For example, we can use this new power to express the property of having an odd number of occurrences of the letter "a". We assert the existence of two sets — one the odd-numbered occurrences of a and one the even-numbered occurrences. The reader may easily verify that the following properties of sets Y and Z are first-order expressible, and that such a Y and Z exist iff our property holds:

- Y and Z are disjoint
- $\pi_a(x)$ iff $x \in Y$ or $x \in Z$
- between every two elements of Y is an element of Z
- between every two elements of Z is an element of Y
- there is an element of Y less than every element of Z
- there is an element of Y greater than every element of Z

Büchi showed [Bu60] that the languages expressible with these quantifiers are exactly the regular languages, and that any expressible property could be expressed by a single block of existential second-order quantifiers followed by a first-order formula. He also considered using the same logical formalism to express properties of *infinite* sequences of letters or "ω-languages". In this case the variables range over the natural numbers, and the expressible languages are an analogous class, the "ω-regular" languages. This work included an automata-theoretic proof of the decidability of a weak form of arithmetic [Bu62] — for a survey see [Mc89a].

McNaughton, at about the same time [Mc60], developed a similar formalism to describe the behavior of finite-state machines with output. Given Büchi's theorem,

the languages expressible in our original formalism (with only first-order quantifiers) must be a subclass of the regular languages. It appears to be a proper subclass, as there is certainly no obvious way to express the "odd number of a's" property with first-order quantifiers alone. About 1965 McNaughton posed this question to Schützenberger, who proved that this class of languages was equal to the *star-free* regular langauges, perhaps the most natural proper subclass of the regular languages [Mc89b]. In Section 3 we outline the proof of this important theorem, which first appeared in print in the 1971 book of McNaughton and Papert [MP71], which is entirely devoted to a large number of characterizations of this particular class of regular languages. We will describe the proof due to Ladner [La77], using the important technique of Ehrenfeucht-Fraïssé games.

A second and independent line of inquiry into expressibility as a complexity measure began with Fagin's Ph.D. thesis in 1973, part of which appeared as [Fa74]. Here he considered essentially the same formalism as did Büchi and McNaughton, but used a different, more general class of second-order quantifiers along with the ordinary first-order quantifiers. Instead of ranging over only monadic relations on the input positions, Fagin's quantifiers range over k-ary relations for arbitrary constants k. With these, it is easy to define non-regular languages, and Fagin was able to characterize the entire class of languages which can be defined.

This class turned out to be a familiar one to theoretical computer scientists, defined in terms of various kinds of polynomial-time Turing machines. (See, e.g., [HU79] for background on Turing machine complexity.) Fagin found that using only *existential* second-order quantifers (and ordinary first-order quantifiers), one can express a language iff it is in the famous class NP (the languages accepted by nondeterministic polynomial-time Turing machines). With only universal second-order quantifiers one gets $co-NP$, the languages whose complements are in NP. Finally, if both kinds of second-order quantifiers are allowed, one can express exactly those languages in the polynomial-time hierarchy of Meyer and Stockmeyer [St76].

In his simulations of Turing machines by formulas, Fagin introduced a two-place predicate on positions which we will call $BIT(i,j)$, meaning that the i'th bit of the binary expansion of j is one. The predicate $f = BIT$ is first-order definable, so in his system one can create it by saying $\exists f(f = BIT \wedge \ldots)$, quantifying over binary relations. When one adds just this predicate to the monadic second-order quantifiers used by Büchi, one suddenly express much more complex languages. Although it is not clear how to express all the languages in NP in this system, one can express some NP-complete languages (such as the set of three-colorable graphs), which are at least as difficult to recognize in the Turing machine model as any language in NP (again, see [Hu79] for background).

Inspired by this work, Immerman began to investigate other expressibility classes defined with respect to the system using this BIT predicate. The class P (languages recognized by polynomial-time Turing machines) corresponds exactly to those predicates expressible in first-order logic together with a special "least fixed point" operator which formalizes the power of defining new relations by induction [Im86] (this was also independently discovered by Vardi [Va82]). Immerman then found a similar characterization of those languages recognizable by Turing machines in logarithmic space [Im87]. Nondeterministic log-space machines recognize the class of properties expressible using first-order logic and a "transitive closure" operator (this is the class NL). A restricted form of the transitive closure operator yields the class L of languages recognized by deterministic log-space Turing machines. These characterizations helped Immerman to find his proof that nondeterministic space classes (including NL) are closed under complement [Im88] (a result also due independently to Szelepcsényi [Sz87]).

By augmenting the first-order system with BIT in a different way, Immerman also found expressibility classes corresponding to various parallel complexity classes [Im89]. So far all the classes described have used the same formula for inputs of any size. But one can define a *family* of formulas by having a block of first-order quantifiers which is iterated some number $f(n)$ of times at the beginning of the formula to be used on inputs of size n. Immerman showed that the languages definable in this way are exactly those recognizable by a certain type of parallel random access machine in time $O(f(n))$.

It was already known (e.g., [SV84]) that these parallel time classes were equal to certain *combinatorial* complexity classes defined in terms of boolean circuits. In Section 4 we give an overview of combinatorial complexity theory, and describe the important notion of uniformity. Circuit complexity classes involve infinite families of circuits to handle inputs of arbitrary length. A *uniformity condition* forces these circuits to be defined by some finite amount of information, such a description of a single Turing machine which can output a description of them. Immerman's expressibility classes with iterated quantifier blocks also correspond to circuit complexity classes with very restrictive uniformity conditions.

A special case of this correspondence is a characterization of the class of languages expressible with ordinary first-order logic and the BIT predicate — the most natural expressibility class once BIT is admitted. This class $FO + BIT$ is a subclass of the very limited combinatorial complexity class AC^0, which we will define below in terms of boolean circuits. Researchers in circuit complexity were at the same time searching for a natural uniform subclass of AC^0 (e.g., [Bu87]), and Immerman proposed $FO + BIT$ as a candidate.

Barrington, Immerman and Straubing [BIS88] then showed that a wide variety of possible definitions of "uniform AC^0", including $FO+BIT$, coincide. They extended the first-order framework to describe a variety of other circuit complexity classes in two ways. New atomic predicates in the formalism (such as BIT) correspond to relaxing the uniformity condition on circuit families, while new types of quantifiers correspond to new types of gates in the circuit. From this point of view, the classes of star-free, solvable, and arbitrary regular languages are naturally viewed as very uniform versions of the circuit complexity classes AC^0, ACC^0, and NC^1, further explaining the surprising connections between finite automata and circuit complexity [Ba89, BT88]. (See section 4 for definitions of these classes.) It is also possible to describe more languages (all those in uniform NC^1) by single formulas without the use of Immerman's iterated quantifier blocks.

In section 5 we describe how the addition of new atomic predicates allows non-uniform computation. In section 6 we describe some of the new quantifiers which can be added to the system. Finally, in section 7 we look at what this approach might offer in the future to the study of these complexity classes.

3. First-Order and Star-Free

The regular languages are the class of languages built up from the empty language and the one-letter languages by boolean operations, concatenation, and the Kleene star operation. It is natural to consider those languages which can be built up using only the first two of these, the *star-free* languages. Formally, given a fixed finite alphabet A, $\mathcal{S}$ is the least class of languages $L \subseteq A^\star$ such that:

- $\emptyset \in \mathcal{S}$, $\{a\} \in \mathcal{S}$ for all $a \in A$
- if $L, M \in \mathcal{S}$, then $L \cup M, L \cap M, \bar{L} \in \mathcal{S}$
- if $L, M \in \mathcal{S}$ and $a \in A$, then $LaM \in \mathcal{S}$

The last condition is slightly different from the usual definition in terms of ordinary concatenation of languages and is slightly more convenient from the standpoint of the algebraic theory of automata. It is not hard to show that the two conditions each give the same class of star-free languages. One shows by induction that if $L \in \mathcal{S}$ and $a \in A$, then the language $a^{-1}L = \{x : ax \in L\}$ is also in $\mathcal{S}$.

The fundamental construction of algebraic automata theory is that of the *syntactic monoid* of a regular language. Given a language $L \subseteq A^\star$, define an equivalence relation

$\equiv_L$ on strings in $A^\star$ by $x \equiv_L y$ iff for all $u, v \in A^\star$, $uxv \in L$ iff $uyv \in L$. The language L is clearly a union of equivalence classes of $\equiv_L$. The equivalence classes of this relation form a monoid under the operation induced by concatenation of strings, i.e., for $x, y \in A^\star, [x][y] = [xy]$ where $[x]$ is the equivalence class of the word x under $\equiv_L$. One can verify that this operation is well-defined and that the resulting monoid M_L is finite iff L is regular. The function taking x to $[x]$ is a homomorphism from the monoid $A^\star$ onto the monoid M_L (a function which preserves multiplication), called the *syntactic homomorphism* of L.

Schützenberger proved in 1965 [Sc65] that the star-free languages could be characterized by a particular property of their syntactic monoids. Any finite monoid has the property that there exists a positive integer q such that, for sufficiently large integers t, every element a satisfies the property $a^{t+q} = a^t$. A monoid is defined to be *aperiodic* if this q can be taken to be 1, or equivalently if no subset of it forms a nontrivial group under the monoid operation. (Incidently, a monoid is a group iff this t can be taken to be 0.) A language is star-free iff its syntactic monoid is aperiodic. The proof of this fact [Sc65, Me69] goes deeper into the algebraic theory than is possible for us in this brief survey. It allows us, however, to prove the equivalence of three properties of languages: being star-free, being first-order definable, and having an aperiodic syntactic monoid. The class of languages having these properties is the subject of [MP71], which consolidates earlier results on it and provides many more characterizations of it.

To begin, it is fairly easy to show that all star-free languages are first-order definable, by induction on the definition of the class $\mathcal{S}$. The property $w \in \emptyset$ is expressed by, say, $\exists x[\neg(x = x)]$, and $w \in \{a\}$ is expressed by $[\exists x(\pi_a(x))] \wedge [\forall x \forall y(x = y)]$. The boolean combination of properties is clearly expressed by the matching boolean combination of their formulas. It remains to show how to express $w \in MaN$, given formulas ϕ_M and ϕ_N expressing the characteristic properties of languages M and N.

To do this we define formulas $\phi[x,]$ and $\phi[,x]$ for every formula ϕ, where x is a new free variable, not occurring in ϕ itself. Given a word w and values for x and any other free variables in ϕ, $\phi[x,]$ will hold iff ϕ holds for the string obtained by taking all letters of w after position x. Similarly $\phi[,x]$ will hold iff ϕ holds for the string of letters of w before position x. Once we show how to construct these two formulas, we will be able to express the property $w \in MaN$ by $\exists x(\phi_M[,x] \wedge \pi_a(x) \wedge \phi_N[x,])$. The construction is by induction on the structure of the formula ϕ, and is left to the reader (or see, e.g., [STT88]).

We now turn to the next step, that any first-order definable language has an aperiodic syntactic monoid, outlining a proof due to Ladner [La77]. We will define an equivalence relation $\equiv_k$ on words in $A^\star$ such that whenever $x \equiv_k y$, we will be

assured that x and y satisfy *exactly the same* set of sentences in prenex form with at most k quantifiers in our first-order formalism. Clearly, a language defined by such a formula must be a union of equivalence classes of this relation. We will be able to show that there are a large but finite number of equivalence classes of $\equiv_k$ for each k.

The key part of the argument is to show that these equivalence classes also form a finite monoid under the concatenation operation, and that this monoid is aperiodic. The result then follows by two fundamental facts of algebraic automata theory (see, e.g, [La79,Pi86]):

- If L is the inverse image of a subset of the finite monoid M under a monoid homomorphism $\phi : A^\star \to M$, then there is another monoid homomorphism $\psi : M \to M_L$, such that the composition $\psi \circ \phi : A^\star \to M_L$ is the syntactic homomorphism of L. Note that ψ must be onto M_L for this to happen.
- If there is a monoid homomorphism from M onto N and M is aperiodic, then N is aperiodic.

The relation $\equiv_k$ is defined in terms of *Ehrenfeucht-Fraïssé games*. These were originally defined by Ehrenfeucht [Eh61], who introduced the game methodology to explicate and extend a method due to Fraïssé [Fr54, Fr56] for showing two structures indistinguishable by first-order formulas. Given two words v and w in $A^\star$, we define a game between two players E and F, who will each make k moves before a winner is decided. In any finite deterministic zero-sum game of perfect information, each player has an optimal strategy (the game is *determined*) — this is easily shown by induction on the number of moves. We define $v \equiv_k w$ to be true iff F wins the game when both play optimally.

E moves first by naming either a position x_1 in v or a position y_1 in w. F responds by naming a position in the other word, to complete the first round. E then names x_2 or y_2, F names the other of these two positions, and so on until positions $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ have all been chosen. At this point, F wins the game iff

- for each i, the letter in position x_i of v and the letter in position y_i of w are the same, and
- for each i and j, $x_i < x_j$ iff $y_i < y_j$ and $x_i = x_j$ iff $y_i = y_j$.

It should be clear that $\equiv_k$ is an equivalence relation. Player F wins if $v = w$ by employing a simple *identity strategy*, always playing x_i equal to y_i or vice versa. The game is symmetric between v and w by definition. If F knows winning strategies

for the games defined by the pairs $\{u, v\}$ and $\{v, w\}$, it is easy to combine them to produce a strategy for the pair $\{u, w\}$. If E moves in word u, for example, F finds the correct move in word v by the $\{u, v\}$ strategy, imagines that move as E's move in the $\{v, w\}$ game, and makes the move in w provided by the $\{v, w\}$ strategy.

The important property of $\equiv_k$ is now a special case of the following result. If $x_1, \ldots, x_r$ are positions in a word v, and $\phi(z_1, \ldots, z_r)$ is a formula with free variables $z_1, \ldots, z_r$, then the tuple $\langle v, x_1, \ldots, x_r \rangle$ (called a "marked word" by Perrin and Pin [PP86]) can be said to satisfy or not to satisfy ϕ, with each x_i substituted for the corresponding z_i. We will omit the detailed definition of the semantics here.

Proposition 1: If the moves $x_1, \ldots, x_r, y_1, \ldots, y_r$ have already been made in the first r rounds of the $(k + r)$-round Ehrenfeucht-Fraïssé game on words v and w, and player F has a winning strategy for the remainder of the game, then the marked words $\langle v, x_1, \ldots, x_r \rangle$ and $\langle w, y_1, \ldots, y_r \rangle$ satisfy exactly the same formulas with r free variables and k quantifiers.

Proof: We prove this by induction on k, simultaneously for all r. If $k = 0$, a quantifier-free formula can only make assertions about the content and order of the positions designated by the free variables. It is clear from the definition of the game that F wins iff the two marked words satisfy the same formulas of this type. Assume then that $k > 0$ and that the theorem holds in the case of $k - 1$ quantifiers. We describe a strategy for player E which is guaranteed to win if the desired condition is not true.

If the two marked words satisfy different formulas, we can pick out a formula of the type $\exists x \phi(x)$ which one satisfies and the other does not. Here ϕ has $k-1$ quantifiers and r free variables in addition to x. Without loss of generality, assume that $\langle v, x_1, \ldots, x_r \rangle$ satisfies $\exists x \phi(x)$. Player E will choose a position x_{r+1} such that $\phi(x_{r+1})$ holds for the marked word $\langle v, x_1, \ldots, x_{r+1} \rangle$. Player F cannot choose a position y_{r+1} in w to make $\phi(y_{r+1})$ true for the marked word $\langle w, y_1, \ldots, y_{r+1} \rangle$, since no such y_{r+1} exists. So after this round, the two marked words will not satisfy the same formulas with $r + 1$ free variables and $k - 1$ quantifiers, since ϕ is one of these formulas. By the inductive hypothesis, then, player E wins the remaining game. ∎

Next we must show that the equivalence classes form a monoid. The monoid operation will be the operation on equivalence classes induced by concatenation, with the class of the empty word as identity. We need only show that the operation is well-defined:

Proposition 2: If $t \equiv_k u$ and $v \equiv_k w$ for words t, u, v, w in $A^\star$, then $tv \equiv_k uw$.

Proof: We must give a winning strategy for player F in the game for $\{tv, uw\}$, given strategies for the games for $\{t, u\}$ and $\{v, w\}$. Each of E's moves in the large

game may be interpreted as a move in one of the two smaller games. F will simply make the move in the large game corresponding to the move given by the winning strategy in the small game. For example, if E names a position x in the part of tv given by t, F uses the $\{t, u\}$ strategy to get a position y in u, and names the position in uw corresponding to position y. It is straighforward to verify that this is a winning strategy. ∎

The aperiodicity (as well as the finiteness) of this monoid follows from one more fact:

Proposition 3: Every element z of the monoid satisfies the equation $z^{t-1} \equiv_k z^t$ for every $t \geq 2^k$.

Proof: Let w be a representative of the equivalence class z. For $k = 1$, it is easy to see that $v \equiv_1 w$ iff v and w contain the same set of letters, so that $w^{t-1} \equiv_k w^t$ for every $t \geq 2$. We will describe F's winning strategy by induction on k. Consider E's first move in the game for $\{w^{t-1}, w^t\}$ for t as given. E names a position in some copy of w — let r be the number of full copies of w before the one picked. That is, we view either w^t or w^{t+1} as $w^r w w^s$, with $r + s + 1$ equal either to t or to $t + 1$. F will move in the identical position in some copy of w in the other word. We let r' be the number of copies before the one picked, so that this word looks like $w^{r'} w w^{s'}$. F can win the remainder of the game if $w^s \equiv_{k-1} w^{s'}$ and $w^r \equiv_{k-1} w^{r'}$, by playing separately in the three subgames as in the proof of the last proposition. If $r + s + 1 = t - 1$ one of r or s (say r), is at least $2^{k-1} - 1$. So $w^r \equiv_{k-1} w^{r+1}$ by the inductive hypothesis, and we can set $r' = r + 1$ and $s' = s$. If $r + s + 1 = t$ then one of r or s (again, say r) is at least 2^{k-1} and we can set $r' = r - 1$ and $s' = s$. ∎

The final step in the three-way equivalence is to show that all regular languages with aperiodic syntactic monoids are star-free, as first proved by Schützenberger [Sc65]. His argument (also in [Pi86]) is by induction on the number of elements of the syntactic monoid, and would take us too far into the realm of algebraic automata theory. Another proof of this result by Meyer [Me69] exploits the structure theorem for finite monoids due to Krohn and Rhodes [KRT68, Ei76]. By this theorem, all aperiodic monoids can be obtained from a single primitive component using the wreath product operation, taking of submonoids, and taking of images under monoid homomorphisms. This allows a much simpler proof by induction on the number of copies of this component which are used. For a third proof, using an intermediate characterization of the star-free languages as those recognized by "loop-free nerve nets", see chapters 7 and 8 of [MP71].

4. Combinatorial Complexity

One important way to formalize the intuitive notion of the difficulty of a problem is to look at combinatorial structures, made up of simple computing elements, which solve it. We expect that more difficult problems would require a greater number of simple elements and would require them to be combined in more complicated ways. This is the point of view of combinatorial complexity. We define particular basic elements, rules for combining them, and combinatorial cost measures on the structures required to recognize a given formal language. We often also place a *uniformity condition* on the structures — a restriction on the computational power which may be used in defining how the elements combine. The class of languages which may be recognized by a structure meeting particular cost constraints and a particular uniformity condition is a *combinatorial complexity class.*

For example, the basic element of the boolean circuit model is a *gate*, which inputs some number of boolean values (which might be inputs to the problem or outputs from other gates) and outputs a boolean value (to other gates or as the output of the problem). To describe a structure of such gates (a circuit) we must give the function computed by each gate (typically AND, OR, or NOT) and tell which gates output to which others. Our cost measures are *size* (number of gates), *depth* (the length of the longest path of gates between an input and an output) and *fan-in* (the largest number of inputs to a single gate).

A fixed circuit has a fixed number of inputs and thus can only process input strings of a fixed length. To recognize a language we need a *circuit family* — a sequence of circuits with one for each possible input length. (We can use the alphabet $\{0,1\}$ or use a binary encoding of the letters in an arbitrary finite alphabet.) Uniformity conditions on circuit families typically restrict the computational power needed to describe each circuit, or the power needed to answer certain questions about it. For example, a circuit family is defined to be *P-uniform* if there is a polynomial-time Turing machine which can produce a description of the n'th circuit given input n in unary. A circuit family is *log-time uniform* if there is a deterministic Turing machine running in time $O(\log n)$ which can answer questions of the form "what is the i'th bit of the description of the n'th circuit"? (A log-time Turing machine is defined to have a read-only, random-access input tape, i.e., it can put a number i on a special work tape and then obtain the i'th bit of the input in one step.)

When one takes a particular combinatorial restriction (say, that the number of gates in the circuit must be bounded by a polynomial in the input size) and varies the uniformity condition, an interesting phenomenon emerges. There appears to be a range of "natural" uniformity conditions which all give the same complexity class. For

example, uniform polynomial-size circuit families recognize exactly those languages in the class P, whether the uniformity condition is "P-uniform" or "log-time uniform". If we make the condition less restrictive (for example, if we remove it entirely and allow arbitrary polynomial-size families) then new languages become recognizable which are not "feasibly computable" in the usual sense. For example, any unary language, even $\{1^n$: the n'th Turing machine halts on blank input $\}$, is recognized by a non-uniform family of small circuits. Similarly, if one makes the uniformity condition too restrictive (as, for example, by demanding that questions about the circuits be answerable by a finite automaton), one gets a smaller complexity class which does not capture the informal notion of what can be computed by circuits of that type.

One of the prime motivations for studying circuit complexity is its connection with parallel computation. Uniform polynomial-size families can recognize P, which is, roughly, the class of problems which are feasibly solvable by sequential algorithms. Some problems in P can be solved much more quickly by parallel algorithms, while others appear to be "inherently sequential". The complexity class NC [Pi79, Co85] is the subclass of P consisting of those languages recognizable by uniform polynomial-size circuit families with depth bounded by a polynomial in $\log n$ for inputs of size n. Again a variety of uniformity conditions, including log-time uniformity, give the same complexity class here (though P-uniform NC is probably different and may be of independent interest — see [Al89]). The class NC is held by many to be a good formalization of the notion of "effectively computable in parallel", in part because it can be defined in a wide variety of models of parallel computation. Although the utility of NC for the study of practical parallel algorithms is hotly debated [Sa89], it offers a good formal setting in which to study the difference between parallelizable and inherently sequential problems. NC is believed to be strictly contained in P, and a number of *P-complete* problems have been found, which are outside of NC if $P \neq NC$.

No techniques are currently known which could prove a language in P to be outside of NC. This has led researchers in parallel complexity to study the internal structure of NC, which has proved to be very rich (see, for example, [Co85]). The class NC^i for each i consists of those languages recognized by circuit families of polynomial size, fan-in two, and depth $O(\log^i n)$. The class AC^i for each i consists of those languages recognized by circuit families of polynomial size, unbounded fan-in, and depth $O(\log^i n)$. In each case, to define a specific class we must also provide a uniformity condition, so that we speak of, for example, non-uniform AC^2 or log-time uniform NC^1. It is easy to see that $NC^i \subseteq AC^i \subseteq NC^{i+1}$ for each i, and that the union of the NC^i and the union of the AC^i are each the entire class NC.

A large body of recent work has centered on the internal structure of NC^1, the

smallest class in this hierarchy not known to be different from P. The languages in non-uniform NC^0 are easily characterized as those for which membership depends on a specific constant number of inputs for each input size. AC^0 is a more interesting class, but we still know how to prove natural languages to be outside of it. The simple language $\{x \in \{0,1\}^\star : x$ has an odd number of ones $\}$ is not in non-uniform AC^0, as shown by Furst, Saxe, and Sipser [FSS84], and independently Ajtai [Aj83]. Curiously, although most interesting languages in non-uniform AC^0 are known to be in log-time uniform AC^0, we do not know how to prove a language outside the latter without proving it outside the former.

We get further interesting subclasses of NC^1 by looking at circuit families with polynomial size, constant depth, and unbounded fan-in and allowing the gate functions to vary. If we take the ordinary AND and OR functions we get the class AC^0. If we gates which calculate the exclusive OR function we get a class which we may call $AC^0[2]$ (this class is strictly larger than AC^0, by the result quoted above). Razborov [Ra87] has shown that $AC^0[2]$ does not contain the language $\{x \in \{0,1\}^\star : x$ has more ones than zeroes $\}$. Smolensky [Sm87] has considered the classes $AC^0[q]$, defined similarly to $AC^0[2]$ but with gates which test whether the number of inputs which are one is divisible by some number q. He showed that the classes $AC^0[p]$ for p prime are independent, in the sense that none of them is contained in another. The union of the classes $AC^0[q]$ for all q (prime and composite) is called ACC^0 [MT89] (also called ACC, e.g., in [BT88, BIS88]). Again, all the techniques for proving languages outside of classes work just as well for the non-uniform classes.

If we include gates which calculate threshold functions (test whether the number of inputs which are one exceeds some number defined for the gate) we define a new class, which has been given the name TC^0 [CSV84, PS88]. This is a subclass of NC^1 which contains most of the languages known to be in NC^1, with two principal exceptions. These are the word problem for a non-solvable group [Ba89] and the boolean sentence value problem [Bu87], each of which is known to be outside of TC^0 unless $TC^0 = NC^1$. NC^1 itself can also be characterized as a constant-depth polynomial-size unbounded fan-in circuit class, where the gates perform multiplication in a fixed non-solvable group [Ba89].

It is a major open question whether TC^0 is equal to NC^1 in different uniformity settings. One reason to believe that they are not equal is that no one has any idea how to do either of the main NC^1-complete problems (evaluating boolean expressions or multiplying sequences of elements of a non-solvable group) in TC^0. A more technical reason is that $TC^0 = NC^1$ would imply that for some fixed k, all of TC^0 can be recognized by threshold circuits of depth k and polynomial size. This is known not to happen in the case of monotone threshold circuits [Ya89].

At the TC^0 level the uniformity condition appears to affect significantly the natural problems that can be solved within the class. While a variety of problems, such as integer multiplication, can be solved in log-time uniform TC^0 (the method of [CSV84] can be made log-time uniform following [BIS88]), there are several which are only known to be in P-uniform TC^0. These include integer division and related problems [BCH86], and various operations in large finite fields [Re87]. There seems to be a sort of "orthogonality" in these classes, where new computing power can be added either by allowing more powerful gates or by weakening the uniformity condition.

5. New Atomic Predicates

The atomic predicates in our first-order system fall into two categories. The predicates $\pi_a(x)$ depend on the input word, while the equality, order, and BIT predicates do not. We call this latter class *numerical* predicates, since for each n they define a k-ary predicate (for some finite k) on the set of numbers $\{1, \ldots, n\}$. We will now consider augmenting the first-order system by adding more predicates, as we have already done once with BIT.

What languages are definable if any numerical predicate is allowed (this question is posed in chapter 11 of [MP71] — see the end of this section)? The answer is very simple in terms of the circuit complexity classes defined above. This result has been proved in the literature several times, apparently first by Immerman in the conference version of [Im87] (but see also [GL84, BCST88, Mo88]). We give a proof here to show the explicit relationship between first-order formulas and constant-depth circuits.

Theorem 4: A language is definable by a first order formula using arbitrary numerical predicates iff it is in non-uniform AC^0.

Proof: To prove that we can evaluate a formula on a particular input using a circuit of the required type, we use induction on the number of quantifiers. If there are no quantifiers, the formula is a boolean combination of input predicates (which become input gates) and numerical predicates (which can be evaluated and become constants). Suppose that the formula is $\exists x \phi(x)$. By induction, for each value a of x there is a circuit of constant depth, polynomial size, and unbounded fan-in evaluating $\phi(a)$. Our desired circuit is simply the OR of these n circuits. If the formula is $\forall x \phi(x)$, we take the AND of n circuits on a similar fashion. Increasing the number of quantifiers by one increases the depth by one and multiplies the size by n, so for a single formula the resulting circuits will be of constant depth and size polynomial in n.

If $\langle C_n : n > 0 \rangle$ is a circuit family satisfying the given restrictions, we can define

a series of numerical predicates as follows. We index the gates of C_n by k-tuples of values in $\{1, \ldots, n\}$, for some k such that no circuit has more than n^k gates. Then we define:

- $AND(x_1, \ldots, x_k)$ to represent "gate number $\langle x_1, \ldots, x_k \rangle$ is an AND gate",
- $OR(x_1, \ldots, x_k)$ similarly,
- $CHILD(x_1, \ldots, x_k, y_1, \ldots, y_k)$ to represent "gate number $\langle x_1, \ldots, x_k \rangle$ is a child of gate number $\langle y_1, \ldots, y_k \rangle$",
- $INPUT(x_1, \ldots, x_k, z)$ to represent "gate number $\langle x_1, \ldots, x_k \rangle$ is an input gate for input variable number z", and finally
- *NEGATED-INPUT*$(x_1, \ldots, x_k, z)$ for "gate number $\langle x_1, \ldots, x_k \rangle$ is an input gate for the negation of input variable number z".

Given these numerical predicates, and a constant bound on the depth of the circuit, we can proceed to define predicates for "gate number $\langle x_1, \ldots, x_k \rangle$ is at depth d" using first-order quantifiers and induction on d. It is then easy to define "gate number $\langle x_1, \ldots, x_k \rangle$ is at depth d and outputs one" by induction on d. For $d = 0$ this is

$$\exists z[(INPUT(x_1, \ldots, x_k, z) \wedge \pi_1(z)) \vee (\textit{NEGATED-INPUT}(x_1, \ldots, x_k, z) \wedge \pi_0(z))]$$

and for larger d we need to say essentially "this is a depth-d OR gate with a child outputting one or this is a depth-d AND gate with no child outputting zero", which is clearly expressible within our system given the inductive hypothesis. ∎

Does a similar result hold in the presence of uniformity? If so, how strong a uniformity condition can be imposed? These questions are answered by Barrington, Immerman, and Straubing, in a result which unifies the various definitions of "uniform AC^0":

Theorem 5:[BIS88] A language is definable using first-order formulas with BIT iff it is in log-time uniform AC^0.

Proof Outline: The structure of the proof is just like that of the previous theorem. To simulate a formula by a log-time uniform circuit family, we must show that the circuits defined above can be numbered in such a way that a log-time Turing machine can find the i'th bit of the n'th circuit given i and n. There are several possible coding schemes, and the details will depend on which is chosen (note that

in this survey we have mostly ignored exactly how a circuit is to be denoted by a string).

To express a log-time uniform AC^0 language by a formula, we can proceed as above once we know that the predicates AND, OR, $CHILD$, $INPUT$, and $NEGATED\text{-}INPUT$ used above are first-order expressible. The uniformity condition tells us that they are log-time computable, and in fact all log-time predicates are expressible, by a key lemma proved in [BIS88]. This lemma depends in turn on a sublemma interesting in its own right, that within the first-order system it is possible to add up $O(\log n)$ bits with a single formula. ∎

Gurevich and Lewis [GL84] observed that the equivalence between AC^0 and first-order formulas holds at virtually any uniformity level, as long as there is enough computational power available to carry out the proof of equivalence. They showed that the power of log-space Turing machines is enough, but we can refine this somewhat using their proof and the previous result:

Corollary 6: Let $\mathcal{C}$ be any class of functions containing the log-time computable functions and closed under composition. Then a language is in $\mathcal{C}$-uniform AC^0 iff it can be expressed by a first-order formula using numerical predicates in $\mathcal{C}$. ∎

Another natural question, raised in [MP71], is which whether new numerical predicates allow one to define any new *regular* languages. There are non-star-free regular languages, such as the set of strings of even length, which are definable using numerical predicates which test divisibility by a constant. In fact (and as conjectured in [MP71]) these are all the new regular languages one obtains even with arbitrary numerical predicates. This is proved in [BCST88], where it is shown that the definibility of any other regular language in this way would cause a violation of the Furst-Saxe-Sipser theorem [FSS84].

6. New Quantifiers

We have seen how the action of ordinary existential and universal quantifiers corresponds to the function of AND and OR gates in a circuit. Given a formula with a free variable, we get a sequence of n boolean values upon which an AND or OR is performed to get a boolean answer. We now consider how this notion can be generalized to get new kinds of quantifiers in our logical system.

The AND and OR operations each make $\{0,1\}$ into a monoid, and there is only one other operation on bits which does this — the exclusive-OR function, or addition mod 2. We can easily define a quantifier "$\exists^{2,1}$" such that for a formula $\phi(x)$ with one

free variable x, "$\exists^{2,1}x\phi(x)$" means "$\phi(x)$ is true for an odd number of x in $\{1,\ldots,n\}$". This certainly increases our class of expressible languages, as $\exists^{2,1}x\pi_1(x)$ is the parity function. But we can just as easily define "$\exists^{m,a}x\phi(x)$" to mean "$\phi(x)$ is true for a number of x equal to a mod m". Here we are adding up the boolean truth values of $\phi(x)$ viewed as elements of a larger monoid, the integers mod m under addition. We convert the result of the addition to a boolean value by testing it for equality with a.

In the same way, we can work with a monoid which is not even fixed with respect to n, such as the integers mod n. We can define "$Mx\phi(x)$" to be true iff $\phi(x)$ is true for a majority of x in $\{0,1\}$, or define "$T_y x\phi(x)$", a formula with free variable y, to be true iff the number of x for which $\phi(x)$ is true is at least y. Both these quantifiers operate by adding the truth values as integers and applying a boolean test to the result.

The modular counting quantifiers were investigated, in the absence of the BIT predicate, by Straubing, Thérien, and Thomas [STT88]. First, since a modular counting quantifier can be simulated by a monadic second-order quantifier, adding these new quantifiers to the ordinary ones still allows us to express only regular languages. But Straubing, Thérien, and Thomas were able to characterize the subclass of the regular languages that can be expressed, as exactly those regular languages which have *solvable* syntactic monoids. These are the monoids for which any subset which forms a group forms a solvable group. In the Krohn-Rhodes structure theory mentioned in section 3, the solvable monoids are exactly those which can be built up from components which are aperiodic monoids or cyclic groups. This is a very natural generalization of the relationship between first-order expressibility (with ordinary quantifiers) and aperiodic monoids, since the latter are just those which can be built up from the components corresponding here to the former.

In the presence of the BIT predicate, the power of the new quantifiers can be determined exactly as in the previous section. The proof of the equivalence of log-time uniform AC^0 and the ordinary first-order expressible languages only assumes that the available gate types include AND and OR. The same argument shows that with the modular counting quantifiers and BIT we get exactly log-time uniform ACC^0 [BIS88]. Similarly, when we add other numerical predicates we still get ACC^0 but with a weakened uniformity condition, so that with arbitrary numerical predicates we get non-uniform ACC^0 [BCST88, Mo88]. This characterization allows one to prove languages to be in log-time uniform ACC^0 by exhibiting formulas rather than first exhibiting circuit families and then giving an explicit proof of uniformity. Such a technique is used by Barrington and Corbett in the case of the languages of binary encodings of semilinear sets of integer vectors [BC90].

In the same way, the languages expressible with majority or threshold quantifiers,

using the BIT predicate, are exactly those in log-time uniform TC^0. In the absence of the BIT predicate, with the ordinary majority quantifier, it is not known whether all of log-time uniform TC^0 can be expressed. However, as shown in [BIS88], one can simulate BIT, and thus express this entire class, with a particular "majority of pairs" quantifier. If $\phi(x,y)$ is a formula with two free variables, then $M^2\langle x,y\rangle\phi(x,y)$ is defined to be true iff $\phi(x,y)$ is true for a majority of the n^2 pairs $\langle x,y\rangle$ with $x,y \in \{1,\ldots,n\}$. Once again, adding numerical predicates of a certain complexity gives the version of TC^0 with the corresponding uniformity condition, following the argument of [GL84]. Also, one can show languages to be in log-time uniform TC^0 by exhibiting formulas, as done by Barrington and Corbett for certain context-free languages [BC89].

Can we use quantifiers to encode other kinds of multiplication? For example, what about the operations of the finite monoids which are not solvable — those which contain non-abelian simple groups? Here we encounter the problem that our values to be multiplied are bits, and so if each of the two values is assigned a value we cannot in general use those two values to generate any element of the monoid. We get around this by essentially introducing a new variable type to the formal system, that of vectors of truth values. For example, fix any group G and an encoding of the elements of G (alternatively, an encoding of a subset of the elements that generates G) as vectors in $\{0,1\}^k$ for some k. Now a vector of k formulas, $\langle\phi_1,\ldots,\phi_k\rangle$, has a truth value which can be interpreted as an element of G. If the formulas have a common free variable x, then for each $i \in \{1,\ldots,n\}$ the vector $\langle\phi_1(i),\ldots,\phi_k(i)\rangle$ can be interpreted as some $g_i \in G$. We may define a set of new "group quantifiers", $Q^{G,g}$ for each $g \in G$, so that $Q^{G,g}x\langle\phi_1(x),\ldots,\phi_k(x)\rangle$ is true iff the product $g_1g_2\ldots g_n$, computed in G, is equal to g.

Barrington, Immerman, and Straubing [BIS88] give an exact characterization of the languages which can be expressed with these new group quantifiers (along with the ordinary first-order ones). First, consider the situation in the absence of BIT. Since the group quantifiers can be simulated using Büchi's monadic second-order quantifiers, only regular languages can be defined. If quantifiers for all finite simple groups are available, any regular language can be expressed. If only some groups are available, only those languages whose syntactic monoids can be constructed from those groups (in the Krohn-Rhodes sense [KRT68, Ei76]) can be expressed. Just as with the modular counting quantifiers, the expressibility theory of the regular languages corresponds exactly with the complexity theory given by the algebraic study of automata.

With BIT, our earlier result tells us that group quantifiers are going to model constant-depth, poly-size, log-time uniform circuits which have "group gates" — gates which interpret their inputs as a sequence of elements of the given group and multiply

them. A slight reformulation [BIS88] of the result of [Ba89] shows that such circuits can recognize exactly those languages in log-time uniform NC^1. This shows that the relative power of gate types depends crucially on the uniformity setting. With log-time uniformity, group gates are at least as powerful as threshold gates. In the more uniform setting given by formulas without BIT, we know that threshold gates give new languages. Similarly, with BIT one non-abelian simple group can simulate another, while this is provably not the case without BIT.

The idea of quantifiers for general operations can be extended even further. In [BIS88] it is shown that quantifiers can be defined for any function on bit vectors which is derived from an associative operation with an identity. Some recent work by Reif [Re87] and by Boyar, Frandsen, and Sturtevant [BFS88] has considered variants of the circuit model where the wires of the circuit contain values in some finite field which depends on the input size. It should be possible to adapt the logical formalism used here, by the addition of new variable types, to model these circuits. This should clarify the effect of the uniformity setting on the simulation arguments given in these papers. Finally, the iterated quantifier blocks of Immerman [Im89] can be thought of as a new type of quantifier, whose effect also depends on the input size.

7. Conclusion

So far, the primary importance of this logical formalism has been the new view of complexity which it provides. It has contributed to new results on nondeterministic space [Im88] and on the connection between automata theory and circuit complexity [BT88], but these results have been proved in a manner largely independent of the logical formalism. There are proof techniques, such as the Ehrenfeucht-Fraïssé games, which are intrinsically logical. The next step in the development of this theory will be to apply the power of these techniques to prove essentially new results. A first step could be to provide intrinsically logical proofs of known results which are simply stated in the logical setting. For example, consider the fact that the parity predicate cannot be expressed with a first-order formula. Work with circuits tells us that even arbitrary numerical predicates will not help us do so [FSS84]. Without the BIT predicate, it is easy to show it impossible by an Ehrenfeucht-Fraïssé game argument. What about with BIT alone? Of course it is still impossible, but it is unsatisfying not to have a logical proof of this logical fact. The right proof might extend to other types of quantifiers, which we have seen would given us entirely new results (potentially separating NC^1 from higher classes). We need to see whether such proofs are possible, as we continue to develop the expressive power of the formalism.

8. Acknowledgements

I would like to thank Bob Ingalls and Mukkai Krishnamoorthy for organizing the very stimulating meeting in Troy, in honor of Bob McNaughton, which prompted the writing of this survey. I would also like to thank Kevin Compton, Jay Corbett, Ron Fagin, Neil Immerman, Bob McNaughton, Sushant Patnaik, Howard Straubing, and the anonymous referees for various helpful discussions and comments.

9. References

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

[Al86] E. Allender, "P-uniform circuit complexity", *J. ACM*, to appear. Also Technical Report DCS-TR-198 (Aug. 1986), Dept. of Comp. Sci., Rutgers University.

[Ba89] D. A. Barrington, "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1", *J. Comp. Syst. Sci.* **38:1** (Feb. 1989), 150-164.

[BCST88] D. A. M. Barrington, K. Compton, H. Straubing, and D. Thérien, "Regular languages in NC^1", Technical report BCCS-88-02 (Oct. 1988), Boston College. Revised version *J. Comp. Syst. Sci.*, to appear.

[BC89] D. A. M. Barrington and J. Corbett, "On the relative complexity of some languages in NC^1," *Inf. Proc. Letters* **32** (1989), 251-256.

[BC90] D. A. M. Barrington and J. Corbett, "A note on some languages in uniform ACC^0," *Theoretical Computer Science*, to appear.

[BIS88] D. A. M. Barrington, N. Immerman, and H. Straubing, "On uniformity within NC^1," *Structure in Complexity Theory: Third Annual Conference* (Washington: IEEE Computer Society Press, 1988), 47-59. Revised version *J. Comp. Syst. Sci.*, to appear.

[BT88] D. A. M. Barrington and D. Thérien, "Finite monoids and the fine structure of NC^1", *J. ACM* **35:4** (Oct. 1988), 941-952.

[BCH86] P. W. Beame, S. A. Cook, and H. J. Hoover, "Log-depth circuits for division and related problems", *SIAM J. Comp.* **15** (1986), 994-1003.

[BFS88] J. Boyar, G. Frandsen, and C. Sturtivant. "An algebraic model for bounding circuit threshold depth," Technical report 88-005 (April 1988), University of Chicago.

[Bu60] J. R. Büchi, "Weak second-order arithmetic and finite automata", *Z. Math. Logik Grundlagen Math.* **6** (1960), 66-92.

[Bu62] J. R. Büchi, "On a decision method in restricted second-order arithmetic," *Proc. 1960 Int. Congress on Logic, Methodology, and Philosophy of Science* (Palo Alto: Stanford Univ. Press, 1962), 1-11.

[Bu87] S. R. Buss, "The Boolean formula value problem is in ALOGTIME," *19th ACM Symp. on Theory of Computing* (1987), 123-131.

[CSV84] A. K. Chandra, L. J. Stockmeyer, and U. Vishkin, "Constant depth reducibility," *SIAM J. Comp.* **13:2** (1984), 423-439.

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

[Eh61] A. Ehrenfeucht, "An application of games to the completeness problem for formalized theories," *Fund. Math* **49:2** (1961), 129-141.

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

[Fa74] R. Fagin, "Generalized first-order spectra and polynonial-time recognizable sets," *SIAM-AMS Proceedings* **7** (1974), 43-73.

[Fr54] R. Fraïssé, "Sur quelques classifications des systèmes de relations," Thesis (1953), Université de Paris, also *Alger-Mathématiques* **1:1** (1954), 35-182.

[Fr56] R. Fraïssé, "Application des γ-operateurs au calcul logique du premier echelon," *Z. Math. Logik Grundlagen Math.* **2** (1956), 76-92.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits, and the polynomial-time hierarchy", *Math. Syst. Theory* **17** (1984), 13-27.

[GL84] Y. Gurevich and H. R. Lewis, "A logic for constant depth circuits," *Information and Control* **61** (1984), 65-74.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* (Reading, Mass.: Addison-Wesley, 1979).

[Im86] N. Immerman, "Relational queries computable in polynomial time," *Information and Control,* **68** (1986), 86-104.

[Im87] N. Immerman, "Languages that capture complexity classes," *SIAM J. Comput.* **16:4** (1987), 760-778.

[Im88] N. Immerman, "Nondeterministic space is closed under complementation," *SIAM J. Comp.* **17:5** (Oct. 1988), 935-938.

[Im89] N. Immerman, "Expressibility and parallel complexity," *SIAM J. Comput.* **18** (1989) 625-638.

[KRT68] K. B. Krohn, J. Rhodes, and B. Tilson, in M. A. Arbib, ed., *The Algebraic Theory of Machines, Languages, and Semigroups* (New York: Academic Press, 1968).

[La77] R. E. Ladner, "Application of model-theoretic games to discrete linear orders and finite automata", *Information and Control* **33** (1977), 281-303.

[La79] G. Lallement, *Semigroups and Combinatorial Applications* (New York: J. Wiley & Sons, 1979).

[MT89] P. McKenzie and D. Theérien, "Automata theory meets circuit complexity," *Proc. 16th ICALP, Springer Lecture Notes in Computer Science* **372** (1989), 589-602.

[Mc60] R. McNaughton, "Symbolic logic and automata," Technical Note 60-244 (July 1960), Wright Air Development Division, Wright-Patterson AFB, Ohio.

[Mc89a] R. McNaughton, "Büchi's sequential calculus," Technical Report 89-8, Dept. Comp. Sci., Rensselaer Polytechnic Institute.

[Mc89b] R. McNaughton, personal communication (1989).

[MP71] R. McNaughton and S. Papert, *Counter-Free Automata* (Cambridge, Mass.: MIT Press, 1971).

[Me69] A. R. Meyer, "A note on star-free events," *J. ACM* **16** (1969), 220-225.

[Mo88] B. Molzan, "Expressibility and nonuniform complexity classes," preprint (1988), Akademie der Wissenschaften der DDR.

[PS88] I. Parberry and G. Schnitger, "Parallel computation with threshold functions", *J. Comp. Syst. Sci.* **36:3** (1988), 278-302.

[PP86] D. Perrin and J. E. Pin, "First-order logic and star-free sets," *J. Comp. Syst. Sci.* **32** (1986), 393-406.

[Pi86] J. E. Pin, *Varieties of Formal Languages* (New York: Plenum Press, 1986).

[Pi79] N. Pippenger, "On simultaneous resource bounds (preliminary version)," *Proc. 20th IEEE Symp. on Foundations of Computer Science* (1979), 307-311.

[Ra87] A. A. Razborov, "Lower bounds for the size of circuits of bounded depth with basis $\{\&, \oplus\}$", *Mathematicheskie Zametki* **41:4** (April 1987), 598-607 (in Russian). English translation Math. Notes Acad. Sci. USSR **41:4** (Sept. 1987), 333-338.

[Re87] J. H. Reif, "On threshold circuits and polynomial computation", *Second Structure in Complexity Theory Conference* (1987), 118-123.

[Sa89] J. L. C. Sanz, editor, *Opportunities and Constraints of Parallel Computing* (New York: Springer Verlag, 1989).

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

[Sm87] R. Smolensky, "Algebraic methods in the theory of lower bounds for Boolean circuit complexity", *19th ACM STOC Symp.* (1987), 77-82.

[St76] L. J. Stockmeyer, "The polynomial time hierarchy," *Theoretical Computer Science* **3:1** (1976), 1-22.

[SV84] L. Stockmeyer and U. Vishkin, "Simulation of parallel random access machines by circuits," *SIAM J. Comput.* **13:2** (1984), 409-422.

[STT88] H. Straubing, D. Thérien, and W. Thomas, "Regular languages defined with generalized quantifiers", *Proc. 15th ICALP* (1988), 561-575.

[Sz88] R. Szelepcsényi, "The method of forcing for nondeterministic automata," *Bull. Eur. Ass. Theoretical Comp. Sci* **33** (Oct. 1987), 96-99.

[Th82] W. Thomas, "Classifying regular events in symbolic logic", *J. Comp. Sys. Sci.* **25** (1982), 360-376.

[Va82] M. Vardi, "Complexity of relational query languages," *14th ACM STOC Symp.* (1982), 137-146.

[We87] I. Wegener, *The Complexity of Boolean Functions* (New York: J. Wiley & Sons; Stuttgart: B. G. Teubner, 1987).

[Ya89] A. C. Yao, "Circuits and Local Computation," *Proc. 21st ACM STOC Symp.* (1989), 186-196.