Uniform Characterizations of Complexity Classes

*Heribert Vollmer**

Abstract

In the past few years, generalized operators (a. k. a. leaf languages) in the context of polynomial-time machines, and gates computing arbitrary groupoidal functions in the context of Boolean circuits have raised some interest. We survey results from both areas, point out connections between them, and present relations to a generalized quantifier concept known from finite model theory.

1 Introduction

There is an "amusing and instructive way of looking at [...] diverse complexity classes" [Pap94a, p. 504] that are of current focal interest in computational complexity theory. This way makes instrumental use of characterizations of classes in terms of conditions on computations trees of nondeterministic polynomial-time Turing machines.

As an example, let us look at the class NP. By definition, a language $A \in \mathrm{NP}$ is given by a nondeterministic polynomial-time machine (NPTM) M such that for all inputs x, we have that x belongs to A if and only if in the computation tree that M produces when working on x, we find at least one accepting path. A language $A \in \mathrm{US}$ is given by an NPTM M such that for all inputs x, we have that x belongs to A if and only if in the computation tree of M on x, there is exactly one accepting path. A language $A \in \mathrm{Mod}_p\mathrm{P}$ is given by an NPTM M such that for all inputs x, we have that x belongs to A if and only if in the computation tree of M on x, the number of accepting paths is not divisible by p. A language $A \in \mathrm{PP}$ is given by an NPTM M such that for all inputs x, we have that x belongs to A if and only if in the computation tree of M on x, we have more accepting than rejecting paths. (For background on these and other classes, the reader may consult [Joh90, Pap94a].)

If a computation path is accepting or rejecting is determined by its final halting configuration; this means that we can re-formulate the above examples by talking about conditions on the sequence of leaves in the computation tree of our machines. This has lead researchers to the framework of characterizing complexity classes by so called *leaf languages*, essentially nothing else than conditions on the sequence of leaves in computation trees. This concept was developed by Papadimitriou and Sipser around 1979 when teaching a complexity class at MIT [Pap94b]. It was later rediscovered and published independently in [BCS92, Ver93] and has since then been used actively in the study of complexity classes mostly in between NC^1 and PSPACE.

In the circuit world, Boolean circuits with gates for multiplication in certain algebraic structures, mostly monoids or groupoids (hence these gates are called *monoidal* or *groupoidal* gates), have been used to study the fine structure of NC^1 and LOGCFL [BT88, BIS90, LMSV99].

We define both frameworks here and survey a number of prominent applications.

*Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, D-97072 Würzburg, Germany. e-mail: vollmer@informatik.uni-wuerzburg.de

2 Leaf Languages

2.1 The Model

In the leaf language approach to the characterization of complexity classes, the acceptance of a word given as input to a nondeterministic machine depends only on the values printed at the leaves of the computation tree. To be more precise, let M be a nondeterministic Turing machine, halting on each path printing a symbol from an alphabet Σ, with some order on the nondeterministic choices. Then, $\mathrm{leafstring}^M(x)$ is the concatenation of the symbols printed at the leaves of the computation tree of M on input x (according to the order of M's paths induced by the order of M's choices).

Call a computation tree of a machine M *balanced*, if M branches at most binary, all of its computation paths have the same length, and moreover, if we identify every path with the string over $\{0,1\}$ describing the sequence of nondeterministic choices on this path, then there is some string z such that all paths y with $|y| = |z|$ and $y \preceq z$ (in lexicographic ordering) exist, but no path y with $y \succ z$ exists.

Given now a pair of languages $A, R \subseteq \Sigma^*$ such that $A \cap R = \emptyset$, this defines a complexity class $\mathrm{BLeaf}^{\mathrm{P}}(A, R)$ as follows: A language L belongs to $\mathrm{BLeaf}^{\mathrm{P}}(A, R)$ if there is a polynomial-time NTM M whose computation tree is always balanced, such that for all x, $x \in \mathrm{L} \implies \mathrm{leafstring}^M(x) \in A$ and $x \notin \mathrm{L} \implies \mathrm{leafstring}^M(x) \in R$.

In the case that $A = \overline{R}$ we also simply write $\mathrm{BLeaf}^{\mathrm{P}}(A)$ for $\mathrm{BLeaf}^{\mathrm{P}}(A, R)$. The classes which can be defined by a pair $(A, \overline{A})$ are *syntactic classes* in the terminology of Papadimitriou [Pap94a], while those which cannot are *semantic classes*.

This computation model was introduced, as already mentioned above, by Papadimitriou and Sipser around 1979, and published for the first time by Bovet, Crescenzi, and Silvestri, and independently by Vereshchagin [BCS92, Ver93] (see also the textbook [Pap94a, pp. 504f]).

Let $\mathcal{C}$ be a class of languages. The class $\mathrm{BLeaf}^{\mathrm{P}}(\mathcal{C})$ consists of the union over all $B \in \mathcal{C}$ of the classes $\mathrm{BLeaf}^{\mathrm{P}}(B)$. In a sequence of papers ([HLS⁺93, JMT94, CMTV98] and more) the complexity of the classes $\mathrm{BLeaf}^{\mathrm{P}}(\mathcal{C})$ was studied as a function of the complexity of $\mathcal{C}$. It was obtained that generally an exponential jump in complexity can be observed, e. g., $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{L}) = \mathrm{PSPACE}$, $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{P}) = \mathrm{EXPTIME}$, etc.

The examples given in Sect. 1 show that the classes NP, US and $\mathrm{Mod}_p\mathrm{P}$ can be defined by regular leaf languages: $\mathrm{NP} = \mathrm{BLeaf}^{\mathrm{P}}\big((0+1)^*10^*\big)$, $\mathrm{US} = \mathrm{BLeaf}^{\mathrm{P}}\big(0^*10^*\big)$, and $\mathrm{Mod}_p\mathrm{P} = \mathrm{BLeaf}^{\mathrm{P}}\big(\{\, w \mid |w|_1 \equiv 0 \pmod{p} \,\}\big)$. The characterization given for PP only gives a context-free leaf language: $\mathrm{PP} = \mathrm{BLeaf}^{\mathrm{P}}\big(\{\, w \mid |w|_1 > |w|_0 \,\}\big)$. Also for PSPACE, a context-free leaf languages can easily be found (recalling that PSPACE corresponds to polynomial time on alternating Turing machines [CKS81] we see that the language of all true Boolean expressions, involving the constants 0 and 1 and the connectives $\wedge$ and $\vee$, will do).

In [HLS⁺93] the question was raised if the latter two classes are also definable via a regular language. It was proved:

Theorem 2.1 [HLS⁺93]. *Let B be a regular language whose syntactic monoid is non-solvable. Then* $\mathrm{BLeaf}^{\mathrm{P}}(B) = \mathrm{PSPACE}$.

Proof sketch. The left to right inclusion is clear, since a PSPACE machine can traverse a whole computation tree. For the converse direction, recall the characterization of PSPACE by alternating machines. Hence we can think of languages in PSPACE as being defined by machines whose computation tree is a tree with leaves 0 and 1 and inner nodes associated with $\wedge$ or $\vee$. An input is accepted if and only if the tree evaluates to 1. Barrington [Bar89] noticed that the commutator

(g, h) of two elements taken from a non-solvable group G has the character of a logical conjunction: (g, h) is equal to the identity iff at least one of g, h is the identity. Here the identity corresponds to truth value 0 and every non-identity element corresponds to 1. Building on this he proved that an $\wedge$-$\vee$-expression ϕ can be transformed to a sequence of elements over G which multiplies out to the identity if and only if ϕ evaluates to 1. In our case, a computation tree of a polynomial-time machine yields an exponentially long sequence of group elements. This sequence can be produced as a leaf string of a polynomial-time machine. Now since the syntactic monoid of B has a non-solvable subgroup, the result follows. ❑

Let PH be the union of all classes of the polynomial-time hierarchy, i.e., $\mathrm{PH} = \mathrm{NP} \cup \mathrm{NP}^{\mathrm{NP}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}} \cup \cdots$. Let MOD-PH be the union of all classes of the oracle hierarchy constructed similarly using as building blocks not only NP but also all classes $\mathrm{Mod}_p\mathrm{P}$ for $p \in \mathbb{N}$.

Theorem 2.2 [HLS$^+$93]. *1.* $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{REG}) = \mathrm{PSPACE}$.

2. Let SOLVABLE *denote the class of all regular languages whose syntactic monoid is solvable. Then* $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{SOLVABLE}) = \text{MOD-PH}$.

3. Let APERIODIC *denote the class of all regular languages whose syntactic monoid is aperiodic. Then* $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{APERIODIC}) = \mathrm{PH}$.

Let us mention that many more complexity classes can be defined via regular leaf languages, for instance higher levels of the polynomial hierarchy (for example, Σ_2^{p} can be defined by $(0+1)^*11(010)^+11(0+1)^*$, intuitively: there is a pair of occurrences of the substring 11 such that in between we have only 010's; an aperiodic language) and all classes of the Boolean hierarchy over NP (for example, $\mathrm{NP} \wedge \mathrm{coNP}$ can be defined by the language of all words in which the substring 010 appears at least once, but the substring 0110 never appears). On the other hand, we see that if PP can be characterized by a regular leaf language, then either $\mathrm{PP} = \mathrm{PSPACE}$ or $\mathrm{PP} \subseteq \text{MOD-PH}$.

Certainly it also makes sense to consider the case of polynomial-time machines where the computation trees are not required to be balanced; the corresponding notation is $\mathrm{Leaf}^{\mathrm{P}}(A, R)$. Classes of this form are studied, e. g., in [Bor94, JMT94, HVW96].

The leaf language way of defining acceptance of a machine has also been transfered into other contexts, e. g., logspace and logtime machines [JMT96], Boolean circuits [CMTV98], and polynomial-time machines computing functions [KSV98]. Instead of going into the details, we refer the reader to the cited papers and present some applications of polynomial-time leaf languages next.

2.2 Complexity Theoretic Applications

Besides just offering an "amusing" way to study complexity classes, as Papadimitriou wrote, leaf languages helped to obtain a number of interesting complexity theoretic statements. We survey some of these below. Of course we can only present a small (and subjective) selection.

2.2.1 Bottleneck Machines

The leaf language characterization of PSPACE leads to a surprising normal-form for this class. Cai and Furst [CF91] defined a class $\mathcal{K}$ to be *$\mathcal{K}'$-serializable*, if for every $A \in \mathcal{K}$ there is a $\mathcal{K}'$-algorithm M, a finite set Q, $F \subseteq Q$, and a polynomial p with the following property: Given input (x, i, e), where x is a word, i a natural number, and e an element from Q, M yields some $f \in Q$. Now let s_0 be a fixed element from Q, and define s_i to be the result of the computation of M on (x, i, s_{i-1})

for $1 \leq i \leq 2^{p(|x|)}$. An input x is accepted by such a computation if $s_{2^{p(|x|)}} \in F$. An M as just described is also called *bottleneck machine*. The term "bottleneck" refers to the restricted way of passing information onwards in the course of the computation.

Theorem 2.1 immediately yields that PSPACE is P-serializable. This can be seen as follows: Let $A \in$ PSPACE be accepted by NPTM M and leaf language B. Let N be a deterministic finite automaton accepting B. Let Q be the set of states of N, F be the set of final states of N, and s_0 be N's initial state. Then the following M' defines a bottleneck machine for A: On input (x, i, q), M' computes the leaf symbol a on the ith path of M. Then M' gives as a result the state that N reaches when reading a in state q. It is clear that as a bottleneck machine, M' accepts A. By a slightly more complicated argument, the following can be shown:

Corollary 2.3 [HLS+93]. PSPACE *is* AC^0*-serializable.*

The power of bottleneck machines was examined in detail in [Her97]. Hertrampf gave a connection between these machines and leaf languages defined via transformation monoids. The power of bottleneck machines as a function of the number of bits passed from one local computation to the next was determined. In this way, a full clarification of the power of bottleneck machines could be obtained.

2.2.2 Oracle Separations

The original motivation for the introduction of leaf languages in [BCS92, Ver93] was the wish to have a uniform oracle separation theorem. Most of the time when relativized complexity classes are separated, this is achieved by constructing a suitable oracle by *diagonalization,* usually a stage construction. Bovet, Crescenzi, Silvestri, and Vereshchagin wanted to identify the common part of all these constructions in a unifying theorem, such that for future separations, one could concentrate more on the combinatorial questions (which are often difficult enough). They showed that to separate two classes defined by leaf languages, it is sufficient to establish a certain non-reducibility between the defining languages. Let (A, R) and (A', R') be pairs of languages such that $A \cap R = A' \cap R' = \emptyset$. Say that (A, R) is *polylogarithmic-time bit-reducible* to (A', R'), in symbols: $(A, R) \leq_m^{plt} (A', R')$, if there are two functions $f\colon \Sigma^* \to \Sigma$ and $g\colon \mathbb{N} \to \mathbb{N}$, computable in polylogarithmic time, such that for all x the following holds: $x \in A \implies f(x,0)f(x,1)\cdots f(x,g(x)) \in A'$ and $x \in R \implies f(x,0)f(x,1)\cdots f(x,g(x)) \in R'$.

Theorem 2.4 [BCS92, Ver93]. $(A, R) \leq_m^{plt} (A', R')$ *if and only if for all oracles Y, the inclusion* $\mathrm{BLeaf}^{\mathrm{P}}(A, R)^Y \subseteq \mathrm{BLeaf}^{\mathrm{P}}(A', R')^Y$ *holds.*

Observe that a polylogarithmic-time bit-reduction cannot (simply because of its time bound) read all of its input. This often allows one to prove $A \not\leq_m^{plt} B$ by an adversary argument. Vereshchagin in [Ver93] used Theorem 2.4 to establish all relativizable inclusions between a number of prominent classes within PSPACE. His list contains besides the classes of the polynomial-time hierarchy also UP, FewP, RP, BPP, AM, MA, PP, IP, and others.

We want to mention one further application of Theorem 2.4. It turned out that in the case of leaf languages A where membership of a word in A depends only on the frequency with which letters appear in A (in other words: two inputs with the same Parikh vector are either both in A or both not in A; these languages have been called *cardinal languages*), the above criterion is especially useful. Building heavily on previous work in [Her95], Hertrampf, Vollmer, and Wagner [HVW95] obtained: A function $f\colon \mathbb{N}^k \to \mathbb{N}$ is a relativizable closure property of #P (i. e., relative to all oracles, if $h_1, \ldots, h_k \in \#\mathrm{P}$ then also $f(h_1, \ldots, h_k) \in \#\mathrm{P}$), if and only if f is a positive linear combinations of multinomial coefficients.

The importance of relativization results certainly is somewhat questionable. Nevertheless we think it is widespread conviction that they may at least help to direct complexity theoretic research [For94]. We believe that this is particularly true for the leaf language model, since it provides a unified panorama of complexity classes.

2.2.3 Circuit Lower Bounds

Leaf languages given by Boolean circuits have been considered in [CMTV98, Vol98]. In the following, we will particularly make reference to the following chain of circuit classes:

$$\mathrm{AC}^0 \subset \mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{SAC}^1 = \mathrm{LOGCFL} \subseteq \mathrm{P}.$$

A language is in a circuit classes from this chain, if it is accepted by a family of polynomial size circuits; moreover we require for AC^0: constant depth, unbounded fan-in, $\neg, \vee, \wedge$ gates; for ACC^0: constant depth, unbounded fan-in, $\neg, \vee, \wedge$ and mod_q gates for q taken from a finite set of natural numbers; for TC^0: constant depth, unbounded fan-in, majority gates; for NC^1: logarithmic depth, bounded fan-in, $\neg, \vee, \wedge$ gates; for SAC^1: logarithmic depth, semi-unbounded fan-in, $\neg, \vee, \wedge$ gates, "semi-unbounded" means that one gate type (either $\vee$ or $\wedge$) is allowed to be of unbounded fan-in, the other one must be of bounded fan-in. Without going into details, we mention that below, all classes are assumed to be logtime-uniform without further mention. For background on circuit complexity, we refer the reader to [Str94, Vol99].

It is easy to see from Theorem 2.1 that $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{NC}^1) = \mathrm{PSPACE}$. Additionally one can prove, e.g., that $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{AC}^0) = \mathrm{PH}$, and that $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{TC}^0)$ is equal to CH, the union of all classes from the counting hierarchy [Wag86], i.e., $\mathrm{CH} = \mathrm{PP} \cup \mathrm{PP}^{\mathrm{PP}} \cup \mathrm{PP}^{\mathrm{PP}^{\mathrm{PP}}} \cup \cdots$.

Building on leaf language characterizations, the class TC^0 was separated from PP:

Theorem 2.5 [CMTV98, All99]. $\mathrm{TC}^0 \neq \mathrm{PP}$.

Proof. First, suppose that $\mathrm{TC}^0 = \mathrm{CH}$. Then we have $\mathrm{TC}^0 = \mathrm{CH} = \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{TC}^0) = \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{CH}) \supseteq \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{P}) = \mathrm{EXPTIME}$, thus $\mathrm{P} \supseteq \mathrm{EXPTIME}$, which is a contradiction; hence $\mathrm{TC}^0 \neq \mathrm{CH}$.

Suppose now that $\mathrm{TC}^0 = \mathrm{PP}$, then $\mathrm{PP}^{\mathrm{PP}} \subseteq \mathrm{PP}^{\mathrm{TC}^0} \subseteq \mathrm{PP} \subseteq \mathrm{TC}^0$; continuing this inductively yields $\mathrm{CH} \subseteq \mathrm{TC}^0$, contradicting the above. ❑

In the non-uniform case no similar lower bound for TC^0 is known.

3 Circuits with Groupoidal Gates

3.1 The Model

Let us now turn to Boolean circuits with gates given by a groupoidal function. Let G be a *groupoid* (i.e., a set, which we also denote by G most of the time, together with a binary operation $\circ\colon G \times G \to G$; below a groupoid is always finite), which has an identity element. Let $F \subseteq G$. The word problem of G with respect to F, denoted by $L(G, F)$, is the set of all sequences of elements from G for which there is a way of parenthesizing them in a legal way such that, when evaluating the resulting expression in G, we obtain an element in F. Fix a $k \in \mathbb{N}$, and a mapping $\phi\colon \{0,1\}^k \to G$. Define a family of functions $f = (f_n)_{n \in \mathbb{N}}$, $f_n\colon \{0,1\}^{k \cdot n^\ell} \to \{0,1\}$, as follows: $f(x_{1,1},\ldots,x_{1,k},x_{2,1},\ldots,x_{2,k},\ldots,x_{n^\ell,1},\ldots,x_{n^\ell,k}) = 1$ if and only if $\phi(x_{1,1},\ldots,x_{1,k}) \circ \phi(x_{2,1},\ldots,x_{2,k}) \circ \cdots \circ \phi(x_{n^\ell,1},\ldots,x_{n^\ell,k}) \in L(G,F)$.

Definition 3.1. A family of Boolean functions $f = (f_n)_{n\in\mathbb{N}}$ is a *groupoidal function*, if it can be obtained from a groupoid as just described.

Thus, a groupoidal functions is nothing else than the characteristic function of the word problem of some groupoid, encoded somehow over the binary alphabet. In our notation below we will sometimes not distinguish between a groupoidal function f and the set whose characteristic function is f.

For a set $\mathcal{F}$ of groupoidal functions, let $\mathrm{AC}^0[\mathcal{F}]$ denote the class of all languages which can be accepted by circuit families of polynomial size and constant depth with $\neg$ gates, unbounded fan-in $\wedge$, $\vee$ gates, and gates for functions from a finite subset of $\mathcal{F}$. The notation $\mathrm{AC}^0[A]$ is actually used somewhat more general in circuit complexity theory: it is the class of all sets that are AC^0 *reducible* to A, i.e., can be recognized by AC^0 circuits with additional gates for the characteristic function of A, where A is not necessarily a groupoid word problem but can be an arbitrary set.

3.2 Complexity Theoretic Applications

3.2.1 The Fine Structure of NC^1

Barrington and Thérien considered the special case of associative groupoids (i.e., monoids) and called the resulting functions *monoidal* functions. By the well-known relation between regular languages and finite monoids, it is clear that every $\mathrm{AC}^0[\mathcal{F}]$ for monoidal $\mathcal{F}$ coincides with a class $\mathrm{AC}^0[\mathcal{C}]$ for a class $\mathcal{C} \subseteq \mathrm{REG}$.

Theorem 3.2 [Bar89, BT88].
1. *Let B be a regular language whose syntactic monoid is non-solvable. Then* $\mathrm{AC}^0[B] = \mathrm{AC}^0[\mathrm{REG}] = \mathrm{NC}^1$.
2. $\mathrm{AC}^0[\mathrm{SOLVABLE}] = \mathrm{ACC}^0$.
3. $\mathrm{AC}^0[\mathrm{APERIODIC}] = \mathrm{AC}^0$.

Hence the structure of the class of regular languages forms a skeleton for the circuit class NC^1; regular languages can be found in NC^1 up through the complete degree.

The power of the class TC^0 is an interesting topic in this context. Above we mentioned that $\mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1$. From the definition of TC^0 it is clear that $\mathrm{TC}^0 = \mathrm{AC}^0[\mathrm{MAJ}]$, where $\mathrm{MAJ} = \{\, w \mid |w|_1 > |w|_0 \,\}$. Hence $\mathrm{TC}^0 = \mathrm{NC}^1$ if and only if some non-solvable regular language is in TC^0 if and only if majority is complete for NC^1. If $\mathrm{TC}^0 \neq \mathrm{NC}^1$ then all regular languages in TC^0 are already in ACC^0, and if moreover TC^0 can be obtained as $\mathrm{AC}^0[\mathcal{C}]$ for some $\mathcal{C} \subseteq \mathrm{REG}$, then $\mathrm{TC}^0 = \mathrm{ACC}^0$.

3.2.2 The class LOGCFL

In an analogous way as the regular languages relate to finite monoids, the context-free languages relate to groupoids: Every word problem of a groupoid is context-free, and conversely every context-free language reduces via a homomorphism to such a word problem. Similar to the above we thus see that every $\mathrm{AC}^0[\mathcal{F}]$ for groupoidal $\mathcal{F}$ coincides with a class $\mathrm{AC}^0[\mathcal{C}]$ for a class $\mathcal{C} \subseteq \mathrm{CFL}$.

Theorem 3.3 [BLM93]. $\mathrm{AC}^0[\mathrm{CFL}] = \mathrm{SAC}^1$.

Subclasses of SAC^1 have been studied in [LMSV99].

4 Relating Polynomial Time to Constant Depth

In Sect. 2.2.3 we already mentioned that, e.g., $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{AC}^0) = \mathrm{PH}$ and $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{NC}^1) = \mathrm{PSPACE}$. Recalling Theorems 2.2 and 3.2, we see that the related circuit and polynomial time classes are given by the same groupoidal functions. Below we present a general theorem along these lines.

Let $g_1, \ldots, g_k$ be groupoidal functions. Let $\mathrm{Circ}[g_k, \ldots, g_1]$ be the class of all languages A for which there exists a family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ of unbounded fan-in boolean circuits of polynomial size and constant depth, where we require that the circuits are leveled, the depth is exactly k, and on level i we have only gates of type g_i for $1 \le i \le k$. Level k is the level of the output gate. On the input level (level 0) we have input variables as well as their negations. $\mathrm{qCirc}[g_k, \ldots, g_1]$ is defined analogously but now we allow quasi-polynomial size, i.e. the size of C_n must be bounded by some $2^{\log^c n}$ for $c \in \mathbb{N}$. Circuits as just described are called *stratified circuits* in the literature. When we want to place a restriction on the fan-in of gates on certain levels, we denote the maximal fan-in in parentheses after the gate type, for example $\mathrm{Circ}[g_k, \ldots, g_1(f)]$ would refer to the class $\mathrm{Circ}[g_k, \ldots, g_1]$ where our circuits additionally fulfill the property that the g_1 gates on level 1 have fan-in bounded by f.

Let us consider some examples. We have $\mathrm{AC}^0_k = \mathrm{Circ}[\underbrace{\vee, \wedge, \ldots}_{k \text{ times}}]$ (AC^0 circuits of depth exactly k), and $\mathrm{TC}^0_k = \mathrm{Circ}[\underbrace{\mathrm{MAJ}, \mathrm{MAJ}, \ldots}_{k \text{ times}}]$ (TC^0 circuits of depth exactly k). Let S_5 denote the set of all sequences of permutations on 5 elements which multiply out to the identity permutation, suitably encoded over the binary alphabet. Since S_5 is a regular language with a non-solvable syntactic monoid, it is complete for NC^1 under AC^0 reductions (Theorem 3.2). Thus we have $\mathrm{NC}^1 = \bigcup_k \mathrm{Circ}[S_5, \underbrace{\vee, \wedge, \ldots}_{k \text{ times}}]$ (where we also use the notation "S_5" to refer to the corresponding monoidal function). If in the above examples we only require quasi-polynomial size we get the classes qAC^0, qTC^0, and qNC. Observe that in the quasi-polynomial context there is only one analogue of the possibly distinct classes L (logarithmic space), NC^1, and NC: this is just the class qNC which equals polylogarithmic space on Turing machines. For more background on quasi-polynomial size circuit complexity, we want to refer the reader to Barrington's excellent survey [Bar92].

Let us generalize the leaf language notion as follows [BS97, Vol97]: Let $\mathcal{C}$ be a complexity class and A be a set of binary strings. Then $A \cdot \mathcal{C}$ consists of all languages L for which there exist some $B \in \mathcal{C}$ and a function $f \in \mathrm{FP}$ such that for all x, $x \in L \iff (c_B(x,1)c_B(x,2)\cdots c_B(x,f(x))) \in A$. It can be seen that $A \cdot \mathrm{P} = \mathrm{BLeaf}^{\mathrm{P}}(A)$. Below, we will use the shorthand $(A)\mathcal{C} = A \cdot \mathcal{C}$ and $(A, A_1, \ldots, A_k)\mathcal{C} = A \cdot (A_1, \ldots, A_k)\mathcal{C}$.

Recall that we identify context-free sets with their characteristic groupoidal functions.

Theorem 4.1 [Vol98]. *For groupoidal functions $g_1, \ldots, g_k$, we have* $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{Circ}[g_1, \ldots, g_k]) = \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{qCirc}[g_1, \ldots, g_k]) = (g_1, \ldots, g_k)\mathrm{P}$.

This theorem yields as consequences that $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{AC}^0) = \mathrm{PH}$, $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{ACC}^0) = \mathrm{MOD\text{-}PH}$, $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{TC}^0) = \mathrm{CH}$, $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{NC}^1) = \mathrm{PSPACE}$, and so on.

Circuit lower bounds have been used in the past few years to obtain separating oracles for polynomial-time classes (see [FSS84, Yao85] and many more). As mentioned in Sect. 2.2.2, one of the main motivations for the leaf language concept was the wish to obtain oracle separations in a uniform way. Hence the just given theorem hints in the direction that both approaches might conveniently be combined.

Central for oracle separations of leaf language defined classes is the non polylogarithmic-time reducibility of the considered leaf languages. This relates as follows to the circuit world. Let $\mathcal{C}$ be a circuit class. We say that $A \leq_m^{\mathcal{C}} B$ if A is many-one reducible to B where the reduction function f can be computed by $\mathcal{C}$ circuits. This means that every bit of $f(x)$ can be computed by a $\mathcal{C}$ circuit, and in the case of polynomial size circuits, the length of $f(x)$ is polynomial in the length of x; and in the case of quasi-polynomial size circuits, the length of $f(x)$ is quasi-polynomial in the length of x.

Lemma 4.2 [Vol98]. *Let A, B be any languages such that $A \leq_m^{plt} B$. Then $A \leq_m^{\mathrm{qCirc}[\vee,\wedge(\log^{O(1)} n)]} B$ and $A \leq_m^{\mathrm{qCirc}[\mathrm{mod}_k,\wedge(\log^{O(1)} n)]} B$ for any $k > 1$.*

Theorem 4.1 and Lemma 4.2, together with Theorem 2.4, establish a systematic way to obtain oracle separations from circuit lower bounds. Let us just give two examples (other applications can be found in [Vol98]):

Theorem 4.3. $\mathrm{NC}^1 \subseteq \mathrm{qTC}^0$ *("quasipolynomial-size* TC^0*") if and only if for all oracles Y, the class* PSPACE^Y *collapses to the counting hierarchy relative to Y.*

Proof sketch. ($\Rightarrow$): If $\mathrm{NC}^1 \subseteq \mathrm{qTC}^0$, then $\mathrm{PSPACE}^Y = \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{NC}^1)^Y \subseteq \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{qTC}^0)^Y = \mathrm{CH}^Y$ for all oracles Y (by Theorem 2.4).

($\Leftarrow$): If for all Y, $\mathrm{PSPACE}^Y = \mathrm{CH}^Y$, then again by Theorem 2.4 every language $A \in \mathrm{NC}^1$ $\leq_m^{plt}$-reduces to some language $B \in \mathrm{qTC}^0$, hence (by Lemma 4.2) $A \leq_m^{\mathrm{qCirc}[\vee,\wedge(\log^{O(1)} n)]} B$; however, qTC^0 is closed under this reducibility. ❑

Theorem 4.4. *There is an oracle Y such that* $\mathrm{PP}^{\oplus \mathrm{P}^Y} \neq \mathrm{PSPACE}^Y$ *if the language S_5 cannot be computed by* $\mathrm{qCirc}[\mathrm{MAJ}, \mathrm{par}, \wedge(\log^{O(1)} n)]$ *circuits.*

Proof sketch. Suppose that for all oracles Y, $\mathrm{PP}^{\oplus \mathrm{P}^Y} = \mathrm{PSPACE}^Y$. Then, by Theorem 2.4, S_5 *plt*-reduces to the "majority-of-parity" leaf language. Hence we conclude by Lemma 4.2, statement 2, that $S_5 \in \mathrm{qCirc}[\mathrm{MAJ}, \mathrm{par}, \wedge(\log^{O(1)} n)]$. ❑

5 Lindström Quantifiers

In descriptive complexity theory, i.e., the study of the complexity of problems by means of the syntactic complexity of logical languages capable to define them, a notion of generalized quantifier is known, which relates to the above introduced notions in several interesting ways. For a systematic introduction to descriptive complexity we refer the reader to [Str94, Imm98].

Let $f = (f_n)_{n \in \mathbb{N}}$, $f_n \colon \{0,1\}^{k \cdot n^\ell} \to \{0,1\}$, be groupoidal, as defined in Sect. 3.1. Such an f defines a first-order quantifier Q_f as follows: Q_f binds ℓ variables and operates on k formulas. More precisely, given $\phi_1(\overline{x}), \ldots, \phi_k(\overline{x})$, each with ℓ free variables $\overline{x}$, the string

$$Q_f x_1, \ldots, x_\ell \, [\phi_1(\overline{x}), \ldots, \phi_k(\overline{x})]$$

is a sentence which evaluates to true, if f yields 1 when applied to the sequence of the n^ℓ vectors of truth values $(\phi_1(\overline{a}), \ldots, \phi_k(\overline{a}))$ for all $\overline{a} \in \{1, \ldots, n\}^\ell$ (for an input of length n), ordered lexicographically.

This definition goes back to Lindström [Lin66], where general algebraic structures (with possibly more than one operation) were used instead of groupoids. Therefore these quantifiers are known as Lindström quantifiers. For a groupoidal function f, we say that Q_f is the *Lindström quantifier defined by f*.

Circuits with groupoidal gates relate to first-order logic with generalized quantifiers, as shown by Barrington, Immerman, and Straubing:

Theorem 5.1 [BIS90]. *The class $\mathrm{AC}^0[\mathcal{F}]$ for a set $\mathcal{F}$ of groupoidal functions is equal to $\mathrm{FO}[\mathcal{F}]$, the class of all languages definable by first-order sentences with Lindström quantifiers defined by functions from $\mathcal{F}$.*

It should be remarked that in the above theorem we always assume ordered structures, i. e., in our language we have a predicate representing a linear order in the input positions. For some classes $\mathcal{F}$ we furthermore need a built-in bit-predicate. An examination of the fine structure of NC^1 and SAC^1 from a logical point of view, where these peculiarities are discussed in detail, is given in [BIS90, LMSV99].

Also the leaf language concept has a corresponding characterization. For this, *second-order Lindström quantifiers* were defined by Burtschick and Vollmer and it was shown:

Theorem 5.2 [BV98]. *Let A be a leaf language which has a neutral letter. Then the class $\mathrm{BLeaf}^{\mathrm{P}}(A)$ coincides with $Q^1_A\mathrm{FO}$, the class of all languages definable by a sentences starting with the second-order Lindström quantifier defined by A, followed by a first-order formula.*

This gives, e. g., the following model-theoretic characterization of PSPACE: A language is in PSPACE if and only if it can be defined by a any second-order Lindström quantifier, defined by a regular language with non-solvable syntactic monoid, followed by an arbitrary first-order formula.

Veith [Vei96] showed that in Theorem 5.2, the second-order quantifier can be replaced by a first-order quantifier defined by the succinct version of A.

In [BV98] an actually stronger statement than Theorem 5.2 was proved: If A is a leaf language defined by first-order sentence ϕ, then $\mathrm{BLeaf}^{\mathrm{P}}(A)$ is the class that is defined by the sentence that we obtain from ϕ by replacing the first-order quantifiers by second-order quantifiers, more specifically:

Theorem 5.3 [BV98]. *Let A be a language with a neutral letter. Then $\mathrm{BLeaf}^{\mathrm{P}}(Q^0_A\Sigma^0_k) = Q^1_A\Sigma^1_k$.*

Acknowledgment. For helpful discussions during the preparation of this survey, I am grateful to Sven Kosub, Würzburg.

References

[All99] E. Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 1999. To appear. A preliminary version appeared in *Proceedings 2nd Computing and Combinatorics Conference*, vol. 1090 of *Lecture Notes in Computer Science*, pages 127–135. Springer Verlag, 1990.

[Bar89] D. A. Mix Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[Bar92] D. A. Mix Barrington. Quasipolynomial size circuit classes. In *Proceedings 7th Structure in Complexity Theory*, pages 86–93. IEEE Computer Society Press, 1992.

[BCS92] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[BIS90] D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[BLM93] F. Bédard, F. Lemieux, and P. McKenzie. Extensions to Barrington's M-program model. *Theoretical Computer Science*, 107:31–61, 1993.

[Bor94] B. Borchert. On the acceptance power of regular languages. In *Proceedings 11th Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 449–460. Springer Verlag, 1994.

[BS97] B. Borchert and R. Silvestri. The general notion of a dot operator. In *Proceedings 12th Conference on Computational Complexity*, pages 26–44. IEEE Computer Society Press, 1997.

[BT88] D. A. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the Association of Computing Machinery*, 35:941–952, 1988.

[BV98] H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *International Journal of Foundations of Computer Science*, 9:277–294, 1998.

[CF91] J.-Y. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2:67–76, 1991.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28:114–133, 1981.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[For94] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the EATCS*, 52:229–244, 1994.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[Her95] U. Hertrampf. Classes of bounded counting type and their inclusion relations. In *Proceedings 12th Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 60–70. Springer Verlag, 1995.

[Her97] U. Hertrampf. Acceptance by transformation monoids (with an application to local self reductions). In *Proceedings 12th Conference on Computational Complexity*, pages 213–224. IEEE Computer Society Press, 1997.

[HLS+93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[HVW95] U. Hertrampf, H. Vollmer, and K. W. Wagner. On the power of number-theoretic operations with respect to counting. In *Proceedings 10th Structure in Complexity Theory*, pages 299–314, 1995.

[HVW96] U. Hertrampf, H. Vollmer, and K. W. Wagner. On balanced vs. unbalanced computation trees. *Mathematical Systems Theory*, 29:411–421, 1996.

[Imm98] N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer Verlag, New York, 1998.

[JMT94] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. In *Proceedings 9th Structure in Complexity Theory*, pages 242–254, 1994.

[JMT96] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Information & Computation*, 129:21–33, 1996.

[Joh90] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 67–161. Elsevier, 1990.

[KSV98] S. Kosub, H. Schmitz, and H. Vollmer. Uniformly defining complexity classes of functions. In *Proceedings 15th Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 607–617. Springer Verlag, 1998.

[Lin66] P. Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

[LMSV99] C. Lautemann, P. McKenzie, T. Schwentick, and H. Vollmer. The descriptive complexity approach to LOGCFL. In *Proceedings 16th Symposium on Theoretical Aspects of Computer Science*. Springer Verlag, 1999. To appear.

[Pap94a] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[Pap94b] C. H. Papadimitriou. Personal communication, 1994.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.

[Vei96] H. Veith. Succinct representation, leaf languages, and projection reductions. In *Proceedings 10th Structure in Complexity Theory*, pages 118–126. IEEE Computer Society Press, 1996.

[Ver93] N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.

[Vol97] H. Vollmer. A generalized quantifier concept in computational complexity theory. Technical report, Institut für Informatik, Universität Würzburg, 1997.

[Vol98] H. Vollmer. Relating polynomial time to constant depth. *Theoretical Computer Science*, 207:159–170, 1998.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

[Wag86] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[Yao85] A. C. C. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings 26th Foundations of Computer Science*, pages 1–10. IEEE Computer Society Press, 1985.