Complexity and Structure in Formal Language Theory

Klaus-Jörn Lange
Fakultät für Informatik, Technische Universität München,
80290 München, Germany

Abstract: The following survey reviews some connections between formal languages and complexity theory. Families of formal languages are treated with complexity theoretical methods. In particular, the concept of unambiguity, common to both areas, is treated in detail. Some complexity theoretical aspects of operations on formal languages are indicated. This picture is completed by taking parallel models into account.

1 Introduction

The central aim of this paper is to illustrate the close relationship between the theory of formal languages and complexity theory. This selection of results is not meant to be a systematic survey, but represents a rather personal view, strongly biased by discussions and results of the complexity group at the Technische Universität München. It is an extended version of an invited talk at the *Structure in Complexity Theory 1993* Conference [44] .

The gist of this work is to see families of formal languages with complexity theoretical eyes. Here by family of formal languages we refer to a typical member of the world of pumping and iteration. More formally, we will treat language families with constructive closure properties and a decidable emptiness problem with tools used in complexity theory. Here we will not be interested in absolute lower and upper bounds, but rather in their behaviour with respect to structural complexity theory. To do so, the notion of a *canonical* storage type is introduced which relates families of formal languages and the complexity classes generated by them under appropriate reducibilities.

This paper is divided into two parts: in the first part, *Sequential Complexity*, we treat relations between families of formal languages and complexity classes defined by various types of sequential automata. This is done first for deterministic and nondeterministic families and then for the intermediate concepts unambiguity and symmetry. The second part, *Parallel Complexity*, relates these classes and families to parallel complexity theory. These investigations concentrate on the models of parallel random access machines and of Boolean circuits.

2 Preliminaries

The reader is assumed to be familiar with the basic facts of complexity theory as they are contained in [3] or [33]. So we will use without explanation the classes $DSPACE(\log n)$,

$NSPACE(\log n)$, P, NP, and $PSPACE$. In addition, let $DEXPOLYTIME$ denote $DTIME(2^{pol})$, which is meant to be an abbreviation of $\bigcup_{k\geq 1} DTIME(2^{n^k})$. $DTISP(f,g)$ and $NTISP(f,g)$ denote the class of languages recognized by deterministic resp. by nondeterministic Turing machines which are bounded in time by f and, simultaneously, in space by g.

For a family $\mathcal{A}$ of languages let $LOG(\mathcal{A})$ denote the class of sets reducible to some element of $\mathcal{A}$ by a many-one logspace reduction. For a single language L_0 we will write $LOG(L_0)$ instead of $LOG(\{L_0\})$.

Throughout this paper, we will refer to various families of formal languages. In particular, we assume the reader to know CFL, the family of context-free languages, LIN, the family of linear context-free languages, and $DCFL$, the family of deterministic context-free languages. Further material may be found in [6, 29, 33]. $DLIN$ will denote the family of *deterministic linear languages* which are recognized by deterministic one-turn push down automata [32].

By *1–NPDA* we denote the family of languages accepted by nondeterministic push down automata with a one-way input tape. As is well-known, this class coincides with CFL. The corresponding classes for nested stack automata, stack automata, nonerasing stack automata, checking stack automata, one-turn push down automata, and one counter automata are denoted by *1–NNstSA*, *1–NSA*, *1–NNeSA*, *1–NChSA*, $1\text{–}NPDA_{1-turn} = LIN$, and *1–NOCA* [1, 26, 68]. For deterministic automata this leads to the classes $1\text{–}DPDA = DCFL$, *1–DNstSA*, *1–DSA*, *1–DNeSA*, *1–DChSA*, $1\text{–}DPDA_{1-turn} = DLIN$, and *1–DOCA*.

The family $INDEX$ of indexed languages was characterized by nested stack automata in [1]. *Macro Grammars* were introduced in [21] and led to the families IO and OI of inside-out and outside-in macro languages. The later family coincides with $INDEX$.

Context-free Lindenmayer languages are defined by iterating homomorphisms and finite substitutions. For the definition of the families $EDOL$, $EDTOL$, EOL, and $ETOL$ we refer to [60].

3 Sequential Complexity

The most typical class of formal languages is CFL, the family of context-free languages. Its relations to complexity classes have been investigated by many researchers. In the following we will look at families of formal languages like CFL as complexity classes or as generators of complexity classes. We will first review the relation of determinism versus nondeterminism. In a second subsection, unambiguity and symmetry, intermediate concepts between determinism and nondeterminism, are shortly considered.

3.1 Determinism versus Nondeterminism

One of the central issues of the Theory of Formal Languages is the opposition of determinism versus nondeterminism. In contrast to the situation in Complexity Theory, the precise relations within formal language theory are well-known; e.g., it is easy to exhibit context-free languages which are not deterministic context-free and the same holds true for an abundance of other families of formal languages. On the other hand, nothing like that is known for the corresponding complexity classes $LOG(CFL)$ and $LOG(DCFL)$. In this subsection we will first look at complexity classes generated by families of formal languages. In most cases, the relation between determinism and nondeterminism

leads us to the well-known open questions of Complexity Theory, like P versus NP or $DSPACE(\log n)$ versus $NSPACE(\log n)$. In a second paragraph this transition from determinism to nondeterminism is characterized by operations on formal languages.

3.1.1 Complexities of Families of Formal Languages

In the following, we will consider complexity classes generated by families of formal languages via many-one logspace reducibilities. While these reducibilities are well-suited for tasks of sequential complexity, they are to powerful and to coarse to deal with small complexity classes below $DSPACE(\log n)$, as they appear in parallel complexity theory. Fortunately, most of the relations considered here, do not vary if we change to simpler notions of reducibility like AC^0 reductions or even projections.

Most of the more prominent families of formal languages are complete for complexity classes defined in terms of time or space. The following table summarizes some of these results by listing for some complexity classes $\mathcal{A}$ some families of formal languages $\mathcal{B}_i$ such that $\mathcal{A} = LOG(\mathcal{B}_i)$. It should be noted that in all these cases these families of formal languages are complete in a stronger sense, in that there always exists a single element in that family which is complete for the corresponding complexity class.

Complexity Class	**Complete Families**	**References**
NP	$INDEX = 1\text{–}NNstSA = OI$, $ETOL$, $1\text{–}NSA$, $1\text{–}NNeSA$, $1\text{–}NChSA$	[59, 71, 65]
P	$1\text{–}DNstSA$, $1\text{–}DSA$, $1\text{–}DNeSA$	[45]
$NSPACE(\log n)$	$LIN = 1\text{–}NPDA_{1-turn}$, $1\text{–}NOCA$, $EDTOL$	[67, 68, 37]
$DSPACE(\log n)$	$DLIN = 1\text{–}DPDA_{1-turn}$, $1\text{–}DOCA$, $1\text{–}DChSA$	[32, 34]

Table 1: Complete Families for Time and Space Classes

In contrast to these close relations between families of formal languages and time or space bounded complexity classes, $LOG(CFL)$, the class of all problems reducible to a context-free language, appears to be a class of its own, not definable as a time or space class. Nevertheless, there are many families of formal languages which are $LOG(CFL)$-complete.

Theorem 1 *The following families of formal languages are complete for $LOG(CFL)$:*

a) *IO, the family of inside-out macro languages [2],*

b) *EOL, the family of context-free Lindenmayer languages without tables [69],*

c) *(CF)EDTOL, the family of context-free controlled, deterministic, Lindenmayer systems with tables [41], and*

d) *GCSL, the family of growing context-sensitive languages [15].*

Thus, Complexity theory allows us to see some inclusions in a new light. For example, the proper inclusions $LIN \subset EDTOL$, $CFL \subset EOL$, $CFL \subset IO$, and $ETOL \subset OI$ turn into equalities when considering the complexity classes generated by these families

of formal languages. On the other hand, it is possible to relate the families *IO* and *OI*, which are incomparable as families of formal languages, within the framework of complexity theory by the inclusion $LOG(IO) \subseteq LOG(OI)$.

Greibach exhibited in [28] a *hardest context-free language* $L_{Greibach}$. Every context-free language is the inverse homomorphic image of $L_{Greibach}$ which is therefore $LOG(CFL)$-complete. Homomorphisms are computable in linear time and thus reductions by inverse homomorphisms preserve not only space bounds, but also running time of context-free languages; i.e.: if you could parse $L_{Greibach}$ in quadratic time, you could do that for every context-free language. This is no longer true for logspace reducibilities, which may distinguish $DSPACE(\log^2 n)$ from $DSPACE(\log^3 n)$, but not $DTIME(n^2)$ from $DTIME(n^3)$. On the other hand, Greibach's result made use of existence of Greibach Normal Form for context-free languages (or in terms of automata: the fact that each context-free language can be accepted by a push-down automaton without ϵ-moves). Thus, this approach applies neither to the various families of stack languages nor to *DCFL*, just to give some examples. With logspace reducibilities, however, it is enough to know that derivation lengths (resp. the runtimes of the corresponding automata) are polynomially bounded.

By the algorithms of Cocke, Kasami, and Younger [74] and of Lewis, Stearns, and Hartmanis [50] we have:

Theorem 2 **a)** $LOG(CFL) \subseteq P$
b) $LOG(CFL) \subseteq DSPACE(\log^2 n)$.

It is still open, whether a polynomial time bound and a polylogarithmic space bound can be achieved simultaneously. This is only known for *DCFL*, for which Cook exhibited in [14] an algorithm with these simultaneous bounds:

Theorem 3 $DCFL \subset DTISP(pol, \log^2 n) = SC^2$.

For arbitrary context-free languages only $CFL \subset NTISP(\text{pol},\log^2 n)$ is known (see [73]).

There is also a machine characterization of $LOG(CFL)$ in terms of *Auxiliary Push Down automata*, which were introduced by Cook in [13]. In an auxiliary push-down automaton the power of a push-down automaton is enhanced in two ways: first, the machine is equipped with a worktape bounded logarithmically[1] in the length of the input, and second, the machine has two-way access to its input. Observe, that the runtime of this enhanced automaton is no longer linear but may be exponential, i.e. $O(2^{pol})$. We will denote by *NAuxPDA-TIME*(*f*) (resp. *DAuxPDA-TIME*(*f*)) the class of all languages recognizable by nondeterministic (resp. deterministic) auxiliary push-down automata in time $O(f)$. Cook showed in [13]:

Theorem 4 $P = NAuxPDA\text{-}TIME\left(2^{pol}\right) = DAuxPDA\text{-}TIME\left(2^{pol}\right)$.

The relations to *CFL* and *DCFL* were determined by Sudborough, who showed in [70]:

Theorem 5 **a)** $LOG(CFL) = NAuxPDA\text{-}TIME(pol)$
b) $LOG(DCFL) = DAuxPDA\text{-}TIME(pol)$.

The relations between formal languages and complexity classes resembled in this section so far, are depicted in Figure 1.

[1] Cook worked with arbitrary space bounds and his results hold in a more general way than indicated here.

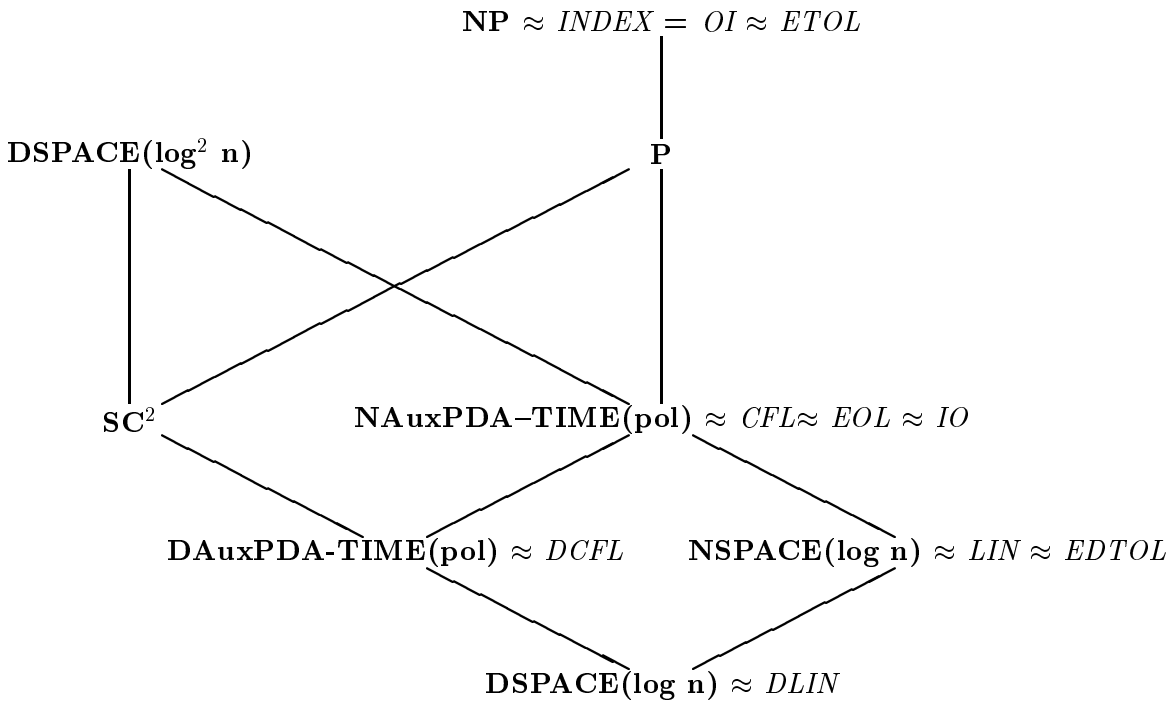

Figure 1: Complexity classes and families of formal languages

Theorems 4 and 5 are not restricted to the context-free languages, but hold for a large variety of families of formal languages. To express this we use the following notions:

Definition 6 **a)** *An automaton type $\mathcal{X}$ is called* $\in$-canonical *iff the word problem of the $\mathcal{X}$ automaton with one-way input is complete for the class of languages accepted by polynomially time bounded* auxiliary $\mathcal{X}$ automata*, i.e.: by $\mathcal{X}$ automata equipped by a logarithmically space bounded working tape and with a two-way input tape restricted by a polynomial time bound.*

b) *An automaton type $\mathcal{X}$ is called* $\emptyset$-canonical *iff the emptiness problem of the $\mathcal{X}$ automaton with one-way input is complete for the class of languages accepted by auxiliary $\mathcal{X}$ automata (without any time bound).*

Theorem 5 tells us that both nondeterministic and deterministic pushdown automata behave $\in$-canonically. The majority of all automata types considered in the theory of formal languages is $\in$-canonical. In particular, all types of automata occurring in Table 1 are $\in$-canonical. This is not restricted to determinism or nondeterminism, but also holds for parity and mod concepts [57]. On the other hand, unambiguous automata, which will be treated in the following subsection, don't seem to be canonical. Another counterexample are finite automata. But here we have the problem that the corresponding complexity classes are located below $DSPACE(\log n)$. Hence the usual logspace reducibilities are no longer appropriate. These low classes will be considered in the section on parallel complexity. Throughout of this section we will use logspace reductions and just mention that the listed results also hold for simpler types of re-

ducibilities like *DLOGTIME* mappings or projections, since the typical reductions used in this context are of the form $v \to v^{p(|v|)}$ for some polynomial p.

Another counterexample to $\in$-canonical relations are Turing machines or any other universal automata type, since then the polynomial time bounds cannot be achieved. But these don't lead to *typical* families of formal languages with pumping lemmata and a decidable emptiness problem. In fact, all known families of formal languages with a decidable emptiness problem and effective closure properties are contained in *NP*.

Theorem 4 tells us, that both nondeterministic and deterministic push down stores are $\emptyset$-canonical. Again, this is very common for many storage types. All storage types occurring in Table 1 except for the deterministic checking stacks are $\emptyset$-canonical. The background underlying $\emptyset$-canonical completeness results are relations between word problems of two-way automata and emptiness problems of one-way automata, which hold in a very general way ([20, 18, 48]). A related class of problems are *General* (or variable) membership problems, i.e., where the grammar or automaton is not fixed but regarded as part of the input [38]. They are typically computationally equivalent to the emptiness problem, if something corresponding to ϵ-productions or ϵ-moves is allowed in the language generating device given as part of the input. Otherwise, lower complexities, typically those of the fixed membership problem are met.

Considering Theorems 4 and 5, we see that push down stores are even *fully canonical*, i.e.: they are not only $\in$- and $\emptyset$-canonical with respect to both determinism and nondeterminism, but in addition, deterministic and nondeterministic auxiliary push down automata are of equal computational power. This pattern of relations is common for a variety of storage types and the automata associated with them. The following table gives some examples for these relations. The first column lists some storage types. The second one gives the complexity class for which the emptiness problem of the corresponding one-way automaton is complete. This is equivalent to the word problem of the corresponding two-way automaton. Here determinism and nondeterminism do not differ w.r.t. complexity. The third and fourth column list the complexity of the word problem in the one way case([34, 45, 70]).

Storage type	**Two-way class**	**Nondeterministic one-way class**	**Deterministic one-way class**
Nested Stack	*DEXPOLYTIME*	*NP*	*P*
Stack	*DEXPOLYTIME*	*NP*	*P*
Nonerasing Stack	*PSPACE*	*NP*	*P*
Push Down	*P*	$LOG(CFL)$	$LOG(DCFL)$

Table 2: Fully canonical Storage Types

It is possible to show, that this pattern is also fullfilled by various types of top-down automata characterizing several families of context-free Lindenmayer languages [60].

3.1.2 Complexity of Operations on Formal Languages

An important topic in the theory of formal languages are investigations of closure properties, that is of questions under which operations a given family of formal languages

is closed. This results in notions like *Trio*s or *AFL*s. A Trio (or cone) is an nonempty collection of sets containing at least one nonempty language, closed under homomorphisms, inverse homomorphisms, and intersections with regular sets. An **A**bstract **F**amily of **L**anguages is a Trio which is additionally closed under the regular operations, i.e.: union, concatenation, and Kleene's *-operation. *CFL*, *OI*, and *ETOL* are AFLs, *LIN* is a Trio, but not an AFL, and *DCFL* is not a Trio, since it is not closed under homomorphisms. There is a rich theory built upon these notions (see e.g. [6, 25]).

Most of the ties between complexity and formal languages are found when considering the complexities of decision problems, in particular of membership problems, of formal languages. This results in completeness of languages and of families of formal languages. In the following, we briefly deal with the complexity of operations on formal languages.

AFL Operations

Trio Operations: From the complexity theoretical view, the easiest operations are inverse homomorphisms and intersections with regular sets. Nearly every sequential machine with a finite control admits the application of these operations. So, at least with respect to sequential complexity, both operations are very easy. In contrast to that, the potential complexity of the remaining Trio operation, the homomorphism, is unbounded, since every recursively enumerable set is representable as the homomorphic image of a simple set. Here simple means containment in $DSPACE(\log n)$. In fact, every recursively enumerable set is the homomorphic image of some element in *Co-NLOGTIME*.

Here we see a fundamental difference of families of formal languages and of complexity classes concerning the impact of erasing. While formal languages usually are rather insensitive with respect to erasing homomorphisms, complexity classes generate all recursively enumerable sets in connection with erasings. Hence within complexity theory only nonerasing homomorphism can be considered. Still these can be of a high complexity (see e.g. [9]): each set in *NTIME(n)* is the homomorphic image of the intersection of three context-free sets, as it was shown in [8]. This implies the *NP*-completeness of $H_{nonerasing}(LOG(CFL))$, since $LOG(CFL)$ is closed under intersection. It is easy to see, that this holds even for deterministic logspace:

Proposition 7 $NP = LOG(H_{nonerasing}(DSPACE(\log n)))$.

On the other hand, *NP* is closed under nonerasing homomorphisms:

Proposition 8 $H_{nonerasing}(NP) = NP$.

These two equations tell us, that the operation of taking the nonerasing homomorphic image is *NP*-complete. For a more formal treatment see [31].

Boolean Operations: Since the complexity classes investigated in this paper are defined by automata with a two-way input tape, they are closed both under union and under intersection. Hence there is no increase in complexity when applying the operation of union or intersection to complexity classes like $DSPACE(\log n)$.

The situation is similar with respect to the operation of taking complements. Classes based on determinism or on nondeterministic space are closed under complementation. But also for classes and concepts which possibly are not closed under complementation

like nondeterministic time, symmetry, or unambiguity there is no jump of complexity. This may be illustrated by the fact, that the closure of P under nonerasing homomorphisms and complement is only the polynomial hierarchy and not $PSPACE$.

Regular Operations: In the same way as nonerasing homomorphisms may be regarded as being NP-complete, Kleene's *-operation is $NSPACE(\log n)$-complete, since we have:

Theorem 9 **a)** $NSPACE(\log n) = LOG(DSPACE(\log n)^*)$[2] *and*
b) $NSPACE(\log n)^* \subseteq NSPACE(\log n)$.

(See [22, 51]. Again these results hold for complexity classes below $DSPACE(\log n)$, e.g. $(AC^0)^*$ contains $NSPACE(\log n)$-complete sets.)

All complexity classes used here are closed under concatenation, which seems to be as easy as inverse homomorphisms or intersection with regular sets. There is one example due to Inga Niepel which characterizes the complexity of concatenation. Hartmanis and Mahaney considered in [30] the $LOG(\cdot)$-closure of all single letter alphabet languages in $NSPACE(\log n)$. They showed this class to coincide with the class of all languages recognizable by logspace automata which in the beginning work deterministically and then enters a second nondeterministic phase that is *blind*, that is in which the input can no longer be read except for its length. Niepel renamed this class ENL (ending nondeterminism) and confronted it with BNL (beginning nondeterminism) where the logspace machine starts with a blind nondeterministic phase and then enters a second, deterministic one in which the machine has full access to its input. Niepel was able to show:

Theorem 10 ([53]) **a)** $ENL \subseteq BNL$,
b) $BNL = LOG(ENL \cdot ENL)$, *and*
c) $BNL \cdot BNL \subseteq BNL$.

Thus the complexities of ENL and BNL are related by the operation of concatenation.

Iterated Insertions

Another nondeterministic complexity class representable in this way is $LOG(CFL)$. The related operation on formal languages will be Ⓑ, the operation of iterated binary insertion [31]. The underlying idea of this approach is the simulation of the grammar with rules $S \to aSbSc \mid d$, which is a generator of the context-free languages (see [6]). For languages L_1, L_2, and L_3 we define the operation of *Binary Insertion* by

$$(L_1, L_2) \to L_3 := \{uvwxy \mid v \in L_1, x \in L_2, uwy \in L_3\}.$$

We will now iterate this operation to get the desired operator Ⓑ. There are two possibilities to do so:

Outside-In One possibility is to insert atomic words into composed words, i.e. to define: $L^{OI-0} := \{\epsilon\}$, $L^{OI-i+1} := L^{OI-i} \cup ((L, L) \to L^{OI-i})$, and $L^{OI-Ⓑ} := \bigcup_{i \geq 0} L^{OI-i}$.

[2]Here $\mathcal{A}^*$ denotes $\{L^* \mid L \in \mathcal{A}\}$.

Inside-Out The other possibility is to insert composed words into atomic words, i.e. to define: $L^{IO-0} := \{\epsilon\}$, $L^{IO-i+1} := L^{IO-i} \cup ((L^{IO-i}, L^{IO-i}) \to L)$, and $L^{IO-Ⓑ} := \bigcup_{i \geq 0} L^{IO-i}$.

Observe, that these two possibilities coincide for associative operations like concatenation: the iterated concatenation both in the outside-in and the inside-out way results in the *-opration. In the case of binary insertion, these two approaches do not seem to be equivalent, since *OI*-Ⓑis *NP*-complete and *IO*-Ⓑis $LOG(CFL)$-complete; i.e. we have:

Theorem 11 **a)** $NP = LOG\Big(DSPACE(\log n)^{OI-Ⓑ}\Big) = LOG\Big(NP^{OI-Ⓑ}\Big)$,
b) $NAuxPDA\text{-}TIME(pol) = LOG\Big(DSPACE(\log n)^{IO-Ⓑ}\Big)$, *and*
c) $NAuxPDA\text{-}TIME(pol) = LOG\Big(NAuxPDA\text{-}TIME(pol)^{IO-Ⓑ}\Big)$.

(Compare this with the *NP*-completeness of *OI* and the $LOG(CFL)$-completeness of *IO*.) Hence we will in the following only consider the operation *IO*-Ⓑ, which will be called Ⓑ, for short. These relation also hold on the formal language side: there is a finite set F, such that $F^{Ⓑ}$ is not only a generator of *CFL*, but also a hardest context-free language. In addition, *CFL*is closed under the Ⓑ operation. Further details will be contained in [31].

Remark 1 *It is also possible to do the same for monadic insertions, i.e.: for the operation* $L_1 \to L_2 := \{uvw \mid v \in L_1, uw \in L_2\}$. *Iterating this we get two operations* OI-Ⓜ *and* IO-Ⓜ. *Again* OI-Ⓜ *is NP-complete, while* IO-Ⓜ *now is* $NSPACE(\log n)$*-complete. Since* OI-Ⓜ *essentially coincides with* OI-Ⓑ *we will deal in the following only with* IO-Ⓜ, *which we will name* Ⓜ, *for short. The* Ⓜ*operation is closely connected with the linear context-free languages.*

Relativization

A remarkable fact of these results is that the relations described in this paragraph do relativize. We have [42]:

Theorem 12 $NP^A = LOG\Big(H\Big(DSPACE(\log n)^A\Big)\Big)$ *for every oracle set A.*

When relativizing $NSPACE(\log n)$ or $NAuxPDA\text{-}TIME(pol)$ we have to be careful which kind of relativization to choose. There are two basic types: In the LL-relativization of Ladner and Lynch in [40] the oracle machine is allowed to perform nondeterministic steps during the generation of a query string. In the RST-relativization of Ruzzo, Simon, and Tompa in [63], the oracle machine is only allowed to perform nondeterministic steps, while the query tape is empty. None of the relations Co-$NSPACE(\log n) = NSPACE(\log n)$, $NSPACE(\log n) \subseteq DSPACE(\log^2 n)$, or $NSPACE(\log n) \subseteq P$ does LL-relativize, but all do when using the RST-relativization. As usual, $NSPACE(\log n)^A$ denotes the LL-relativization, and $NSPACE(\log n)^{\langle A \rangle}$ the RST-version. We then have [42]:

Theorem 13 $NSPACE(\log n)^{\langle A \rangle} = LOG\Big(\Big(DSPACE(\log n)^A\Big)^*\Big)$ *for every oracle set A.*

In the same way we have [31]:

Theorem 14 $NSPACE(\log n)^{\langle A \rangle} = LOG\Big(\big(DSPACE(\log n)^A\big)^{\textcircled{M}}\Big)$ *for every oracle set* A.

Essentially, the same holds true for the operation Ⓑ. Again, we have to use the RST-relativization, which in this case pertains also to the use of the push-down store. (So, here RST and LL no longer coincide in the deterministic case.) We then have [31]

Theorem 15 $NAuxPDA\text{-}TIME(pol)^{\langle A \rangle} = LOG\Big(\big(DSPACE(\log n)^A\big)^{\textcircled{B}}\Big)$
for every oracle set A.

3.2 Between Determinism and Nondeterminism

The consideration of the relationship between determinism and nondeterminism led to the investigation of intermediate concepts like symmetry or unambiguity. A nondeterministic automaton is said to be *unambiguous*, if for any input there is at most one accepting computation. Obviously, every deterministic automaton is unambiguous. While in a usual definition of a language accepted by some automaton the reflexive and transitive closure of some one-step transition relation is used, a *symmetric* computation is based on the symmetric, reflexive, and transitive closure of that relation. Since the configuration graph of a deterministic automaton is a tree, the language accepted by a deterministic automaton coincides with its symmetric language, i.e.: the set of all words accepted by symmetric computations. Thus both concepts are located between determinism and nondeterminism.

3.2.1 Unambiguity

The concept of *Unambiguity* is well known from the theory of automata and formal languages. An automaton is unambiguous if there exists at most one accepting computation for every input. Correspondingly, a grammar is unambiguous if there exist no two different derivations producing the same word. Thus, if there exists an accepting computation or a derivation for some input, it is unique. On the formal language side, this leads to the families *UCFL* of unambiguous context-free languages and *ULIN* of unambiguous linear context-free languages, which coincide with the families of languages accepted by unambiguous push-down and one-turn push-down automata. On the complexity theoretical side, we get classes like *UP*, $USPACE(\log n)$, and $UAuxPDA\text{-}TIME(pol)$.

This feature has to be distinguished from the concept of *Uniqueness*. While an unambiguous automaton allows for at most one accepting computation, a unique machine simply rejects all words having more than one accepting computations. Thus unambiguity means using the unique existence of accepting computations as a restriction, whereas uniqueness is using this as a tool. Hence unambiguity is a concept located between determinism and nondeterminism. On the other hand uniqueness seem to be more powerful than nondeterminism. The class *US* of unique polynomial time introduced in [7] contains *Co-NP* and is conjectured to be different from it. Unique space classes coincide with their nondeterministic counterparts, because of the complement closure of nondeterministic space classes.

While uniqueness is a constructive concept and classes defined by uniqueness possess complete sets, unambiguity is a nonconstructive concept. Complexity classes defined by unambiguity don't seem to be recursively representable or to possess complete sets.

Neither they seem to be $\in$-canonical: it is not known, whether the families *UCFL* and *ULIN* are complete for *UAuxPDA-TIME*(*pol*) and *USPACE*($\log n$). The corresponding construction in [70] preserves determinism, but not unambiguity. We only have $LOG(UCFL) \subseteq$ *UAuxPDA-TIME*(*pol*) and $LOG(ULIN) \subseteq$ *USPACE*($\log n$).

There are two possibilities to define unambiguity. Apart from the weak version of looking only at accepting computations or whole derivations, we could be more restrictive and require that for no pairs of configurations there are two or more different computations leading from the first configuration to the second one. For grammars this would mean that for no pair of sentential forms there are two or more different derivations deriving one out of the other. For time classes these concepts do not differ, since we have enough space to keep track of the whole history of a computation. In this way no configuration has more than one predecessor. For space bounded classes and machines augmented with a space bounded working tape this method does not work and we get additional classes corresponding to strong unambiguity: *StUAuxPDA-TIME*(*pol*) and *StUSPACE*($\log n$). (In fact there are many more variants, see e.g. [12].)

These two concepts coincide on the formal language side, since we can get rid of both the unproductive and the unreachable nonterminals of a grammar. With this it is possible to show the inclusion of formal languages even in the corresponding strong unambiguous classes (see [47, 12]):

Proposition 16 $LOG(UCFL) \subseteq$ *StUAuxPDA-TIME*(*pol*) $\subseteq$ *UAuxPDA-TIME*(*pol*)

and

Proposition 17 $LOG(ULIN) \subseteq$ *StUSPACE*($\log n$) $\subseteq$ *USPACE*($\log n$).

Allthough we know $DCFL \subset UCFL \subset CFL$ and $DLIN \subset ULIN \subset LIN$, the precise relationship of unambiguous complexity classes to deterministic and nondeterministic ones is open. There are some indications that unambiguity is a proper restriction of nondeterminism. An example for this are parity and mod classes; A language $L \subseteq X^*$ is in $\otimes$*SPACE*($\log n$) (*Parity Logspace*) iff there is a nondeterministic logarithmically space bounded Turing machine, which has an odd number of accepting computations for each element of L and an even number number on each input from $X^* \backslash L$. Counting modulo an arbitrary positive integer instead of counting modulo two leads from parity classes to mod classes [11]. By defintion of unambiguity we have:

Proposition 18 **a)** *USPACE*($\log n$) $\subseteq \otimes$*SPACE*($\log n$) *and*
b) *UAuxPDA-TIME*(*pol*) $\subseteq \otimes$*AuxPDA-TIME*(*pol*).

These inclusions are not known to be true for *NSPACE*($\log n$) and *NAuxPDA-TIME*(*pol*). The parity concept can also be defined for formal languages and leads to the canonical equations:

Theorem 19 ([57]) **a)** $LOG(\otimes CFL) = \otimes$*AuxPDA–TIME*(*pol*) *and*
b) $LOG(\otimes LIN) = \otimes$*SPACE*($\log n$).

A further hint that unambiguity might be less powerful than nondeterminism and be rather close to determinism is given in [12] by:

Theorem 20 *StUSPACE*($\log n$) $\subseteq$ *DAuxPDA-TIME*(*pol*).

As a consequence, $StUSPACE(\log n)$ and hence $ULIN$ is a subset of $DTISP(pol, \log^k n) = SC^2$. The corresponding relationship for nondeterminism: $NSPACE(\log n) \subseteq SC^2$ is not known to be true and generally conjectured to be wrong.

The idea underlying the inclusion of $StUSPACE(\log n)$ in $DAuxPDA\text{-}TIME(pol)$ is to prune the reachability subtree of the reachability graph of a strongly unambiguous machine. The proof can be extended to some slightly larger classes, but not (yet) to the class $USPACE(\log n)$ (see [12]). With the same idea it can be shown that $StUSPACE(\log n)$ is closed under complement and even coincides with its alternating version.

One consequence of this inclusion is, that we now can relate the families $ULIN$ and $DCFL$. As families of formal languages they are incomparable: the Dyck languages are not linear context-free and the *Mirror Language* $\{w\overleftarrow{w} \mid w \in \{a,b\}^*\}$ is not deterministic context-free. But if we look at them from the complexity theoretical side, we now have

$$LOG(ULIN) \subseteq DAuxPDA\text{-}TIME(pol) = LOG(DCFL).$$

But it seems appropriate to give a caveat: In the same way CFL and $EDTOL$ are incomparable as families of formal languages. If we now look at the generated complexity classes, we get

$$LOG(EDTOL) = NSPACE(\log n) \subseteq LOG(CFL).$$

But things turn over, if we look at the AFLs generated by CFL and by $EDTOL$, which in this case means to allow for inverse homomorphisms. Observe that inverse homomorphisms are very harmless from the complexity theoretical point of view in that very low classes like NC^1 are closed under inverse homomorphisms. But while AFL(CFL)=CFL stays in $NAuxPDA\text{-}TIME(pol)$, the AFL generated by $COPY$ contains NP-complete languages (see [22]).

3.2.2 Symmetry

The concept of a symmetry was introduced in [49]. In a *symmetric computation* one is allowed to go computational steps both backward and forward. As unambiguity, symmetry is intermediate in computational power between determinism and nondeterminism. But in contrast to unambiguity, symmetry is of a more constructable nature, since it is defined in terms of locally checkable properties. Thus, symmetric classes possess complete problems.

Structural Relations: At the expense of space it is possible to make arbitrary nondeterministic computations symmetric by keeping track of the whole history of a computation. This is the reason for the equivalence of symmetric time and nondeterministic time. Much more interesting is the case of symmetric space classes. They resemble certain similarities with unambiguous space classes concerning their structural behaviour. As $StUSPACE(\log n)$, also $SSPACE(\log n)$ is contained in both SC^2 and $\otimes SPACE(\log n)$:

Theorem 21 **a)** $SSPACE(\log n) \subseteq SC^2$ *[55], and*
b) $SSPACE(\log n) \subseteq \otimes SPACE(\log n)$[*39, 11*].

But it should be mentioned that the proofs on the symmetric side are much more involved than those on the unambiguous one. But there are also certain differences.

On the one hand, it was observed in [49] that all problems in $NSPACE(\log n)$ can be recognized by symmetric machines, which are simultaneously bounded by polynomial time and $O(\log^2 n)$ space:

Theorem 22 $NSPACE(\log n) \subseteq STISP(pol, \log^2 n)$.

Here $STISP$(f,g) denotes the class of all problemes recognizable by symmetric Turing machines bounded in time by f and simultaneously in space by g. An unambiguous version of this containment is not known to be true. On the other hand, it was recently shown in [56] that $SSPACE(\log n)$ is contained in $DSPACE(\log^{1.5} n)$. Again, it is an open problem whether the corresponding inclusions hold for the classes $USPACE(\log n)$ or $StUSPACE(\log n)$.

Complete Operations The known complete sets for $SSPACE(\log n)$ are undirected variants of $NSPACE(\log n)$-complete problems. But there is up to now no family of formal languages which could play for $SSPACE(\log n)$ the role, LIN and $DLIN$ played for $NSPACE(\log n)$ and $DSPACE(\log n)$. This seems to be connected with the problem of defining the concept of symmetry for automata with a one-way access to the input.

In the following we will describe an operation SYM-* on formal languages which is $SSPACE(\log n)$-complete. To define SYM-* which will be a symmetric version of the *-operation we need the following notation: If $w = a_1 a_2 ... a_n$ is a word of length n, we select subwords of w for $0 \leq i \leq j \leq n$ by setting ${}_iw_j := a_{i+1} ... a_j$. Then w is an element of L^* for some language L iff there exists an integer k and indices $0 =: i_0 < i_1 < i_2 ... < i_k := n$ such that for all $1 \leq \mu \leq k$ ${}_{i_{\mu-1}}w_{i_\mu}$ is in L. We extend now the ${}_iw_j$-notation to "negative" subwords by letting ${}_iw_j$ be the reversal of ${}_jw_i$, i.e. $a_i a_{i-1} ... a_{j+1}$, if $j < i$. We then define w to be an element of $L^{\text{SYM-}^*}$ iff there exist an integer k and indices $i_0 := 0, i_1, i_2, ..., i_k := n$ such that for all $1 \leq \mu \leq k$ ${}_{i_{\mu-1}}w_{i_\mu}$ is an element of L. Observe, that now the sequence of the i_μ no longer has to be increasing. As for * and $NSPACE(\log n)$, there exist a simple set L_0 such that $L_0^{\text{SYM-}^*}$ is $SSPACE(\log n)$-complete. (In fact we can take the same set L_0, for which L_0^* is $NSPACE(\log n)$-complete; compare to Theorem 9.)

Theorem 23 ([31]) **a)** $SSPACE(\log n) = LOG\Big(DSPACE(\log n)^{\text{SYM-}^*}\Big)$ *and*
b) $SSPACE(\log n)^{\text{SYM-}^*} \subseteq SSPACE(\log n)$.

An interesting question is now, whether the SYM-*-operation can be characterized as the iteration of a more simple operation, just as the *-operation is iterated concatenation, Ⓜ is iterated monadic insertion, and Ⓑ is iterated binary insertion. This question seems to be closely related to the problem of properly defining symmetry for one-way devices.

Relativization: We finally remark, that as in subsection 3.1.2 it is possible to relativize Theorem 23 when using an appropriate relativization of symmetric pace [31].

Some of the connections between unambiguous and symmetric complexity classes and families of formal languages are indicated in Figure 2.

4 Parallel Complexity

There are several different models used in parallel complexity theory, the most basics being parallel random access machines, Boolean circuits, alternating automata, and in

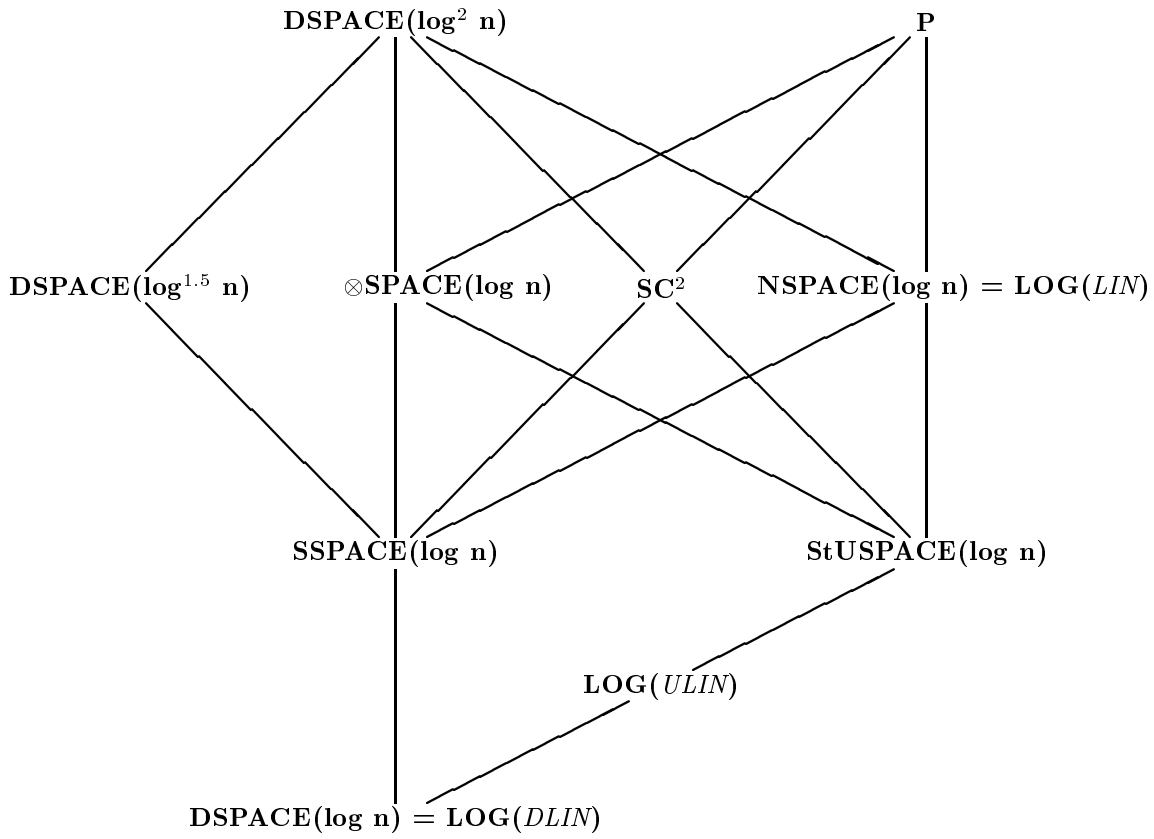

Figure 2: Unambiguous and symmetrical aspects of formal languages

some sense auxiliary push-down automata. In the following, we will mainly deal with PRAMs and circuits.

All these devices lead to the same class NC of languages, which are *efficiently parallelizable*. Dealing with PRAMs, this leads to the class of problems, solvable simultaneously with small (i.e.: polylogarithmic) time and moderate (i.e.: polynomial) number of processors. For circuits it would mean small depth and moderate size, for alternating machines small depth and logarithmic space, and for auxiliary push-down automata time $O(2^{polylog})$ and logarithmic size of the auxiliary working tape ([62] contains a nice unification of several approaches). Parallel complexity theory now contrasts membership in NC as meaning *efficiently parallelizable* with P-completeness meaning *inherently sequential*. On the one hand, this leads to a beautiful theory, on the other hand, it is not clear how to relate these results to existing parallel machines.

The following explanations are centered around CFL and its subfamilies. We start with a treatment of the most important model of parallel complexity theory, the PRAM. $LOG(DCFL)$ coincides with a parallel complexity class. Then we review some results on complexity classes repreented by boolean circuits. In this context $LOG(CFL)$ will be exhibited as a parallel complexity class, as well. In a third subsection on unambiguity we present some of the surprisingly close relationships between concepts of automata theory like nondeterminism, unambiguity, and determinism and concepts of accessing a shared memory like concurrent write, exclusive write, and owner write.

4.1 Parallel Random Access Machines

The concept of a *PRAM* goes back to [23, 27]. A PRAM is a set of Random Access Machines, called *processors*, communicating via a *global memory*. The processors work in a synchronous way, i.e.: each step takes one time unit regardless whether it performs a local or a global (i.e.: remote) operation. The number of processors is variable, depending in the size of the input. Unless stated otherwise, we have a polynomial bound on the number of processors. An essential restriction we put on this machine, is the assumption, that the length off all register cells, being either global or local, is bounded logarithmically in the length of the input. Thus all address and data stored in a memory cell is polynomially bounded.

There are several types of PRAMs classified according to their ability to access simultaneously a cell of global memory when reading or writing. We will consider three versions concerning the write access: a machine with *Concurrent Write* access allows the simultaneous write access of several processors to the same memory cell at one moment. There are several conventions how to solve this conflict, i.e.: how to determine what will be the new value of the referenced cell. In our context, all these methods are equivalent. A machine with *Exclusive Write* access forbids simultaneous writes and requires that in each step at most one processor may change the content of a global memory cell. A machine with *Owner Write* access is even more restricted by assigning to each cell of global memory a processor, called the *Write-Owner*, which is the only one to have write access to this memory cell (see [19]). Correspondingly, we get three ways to manage read access to the global memory: *Concurrent Read, Exclusive Read,* and *Owner Read*. In this way we get nine versions of PRAMs, denoted as *XRYW*-PRAMs with $X, Y \in \{O, E, C\}$, where *XR* specifies the type of read access and *YW* that of the write access and where the access types are designated by their initials. For historical reasons, the commonly used models are CRCW-, CREW-, and EREW-PRAMs. By *XRYW-TIME(f(n))* we denote the class of all languages recognizable in time f by *XRYW*-PRAMs with a polynomial number of processors. By definition we know that *XRYW-TIME(f)* $\subseteq$ *X'RY'W-TIME(f)* for X,X',Y,Y' $\in \{O, E, C\}$ if X $\leq$ X' and Y $\leq$ Y' where we set $O \leq E \leq C$.

In the most powerful of these models, the *CRCW*-PRAM, the global memory looks like a shared memory, since each processor can access each cell of global memory. In the most restricted model, the *OROW*-PRAM, however, the global memory is deteriorated to a set of one-directional channels between pairs of processors. Thus an *OROW*-PRAM is something like a completely connected synchronous network. Although this machine seems to be much more restricted, the relations to other parallel classes listed in the next paragraphs indicate that it is a model as "parallel" as a *CRCW*-PRAM.

Relations to Context-free Languages The context-free languages are well-known to be contained in *NC*. In fact, Ruzzo showed in [61]:

Theorem 24 *CFL* $\subseteq$ *CRCW-TIME*$(O(\log n))$.

This algorithm makes essential use of the concurrent write feature and it is open whether context-free languages could be recognized in logarithmic time by CREW or CROW-PRAMs. The largest subfamily of *CFL* known to be recognizable in logarithmic time without concurrent write, are the unambiguous context-free languages since Rytter showed in [64]:

Theorem 25 *UCFL* $\subseteq$ *CREW-TIME*$(O(\log n))$.

Again it is open, whether this could be done without exclusive write access. The largest subfamily of *UCFL* known to be recognizable in logarithmic time with owner write access, are the deterministic context-free languages, as Dymond and Ruzzo showed in [19]:

Theorem 26 $DCFL \subseteq CROW\text{-}TIME(O(\log n))$.

The first of these three inclusions implies that $NSPACE(\log n)$ and $NAuxPDA\text{-}TIME(pol)$ are subclasses of $CRCW\text{-}TIME(O(\log n))$. The second inclusion implies that $LOG(UCFL)$ is contained $CREW\text{-}TIME(O(\log n))$, but says nothing concerning $StUAuxPDA\text{-}TIME(pol)$ or $UAuxPDA\text{-}TIME(pol)$. Using the circuit model, it is possible to include $StUAuxPDA\text{-}TIME(pol)$ and $StUSPACE(\log n)$ in $CREW\text{-}TIME(O(\log n))$, as will be shown below. Also, we will give some evidence, that $UAuxPDA\text{-}TIME(pol)$ and $USPACE(\log n)$ are probably not contained in $CREW\text{-}TIME(O(\log n))$. The third inclusion implies that $DAuxPDA\text{-}TIME(pol)$ is a subclass of $CROW\text{-}TIME(O(\log n))$. Surprisingly, also the converse holds, as Dymond and Ruzzo showed in [19]:

Theorem 27 $DAuxPDA\text{-}TIME(pol) = CROW\text{-}TIME(O(\log n))$.

It should be remarked that this result implies the inclusion of *DCFL* in SC^2, i.e.: Theorem 3. All these algorithms make use of the concurrent read feature. Even for the linear subfamilies *LIN* and *ULIN* there is up to now no logarithmically time bounded algorithm known without using concurrent read. Only for the deterministic linear context-free languages we know something better by Rossmanith (see [58]):

Theorem 28 $DSPACE(\log n) \subseteq OROW\text{-}TIME(O(\log n))$.

Since *DLIN* is contained in $DSPACE(\log n)$ [32], this yields an PRAM recognizing deterministic linear languages in logarithmic time without the concurrent read feature.

Corollary 29 $DLIN \subseteq OROW\text{-}TIME(O(\log n))$

4.2 Circuits

A very important parallel model within structural complexity theory are boolean circuits. Formally, a circuit is a directed acyclic graph C. The nodes, called *gates*, are labelled by boolean functions such that the number of arguments of a function f corresponds to the number of ingoing arcs of each node v labelled by f. The nodes of in-degree 0, which have to be labelled by boolean constants, are called *Inputs*. Nodes of outdegree 0 are called *Outputs*. By attaching boolean values to each node according to the boolean functions passed at each node, going from the inputs into the direction of the outputs a circuit C computes a boolean function with domain $\{0,1\}^n$ and range $\{0,1\}^m$, if C has n inputs and m outputs. If $m = 1$, C might be regarded as a language acceptor by setting $\mathrm{L}(C) := \{w \in \{0,1\}^n \mid C \text{ on input w evaluates to } 1\}$.

Throughout of this paper, we will work with disjunctions and conjunctions as labels, only. Negation is avoided by the usual assumption, that the inputs are given pairwise, i.e.: with each input x_i also an input $\overline{x_i}$ is given, labelled with the negation of the label of x_i.

Since a fixed circuit has always some finite domain $\{0,1\}^n$, it is necessary to consider circuit families $\mathrm{C} = (C_n)_{n \geq 1}$, where C_n is a circuit with n inputs. To capture complexity

classes, it is usual to assume *Uniformity* restrictions which make the structure of each C_n easily computable for given n and thus relate the elements of a circuit family with each other (see [62]).

Complexity measures of a circuit are the *size*, which is the number of nodes of a circuit, and the *depth*, which is the height, i.e.: the length of the longest path from any input to any output. In the following we will be interested in circuit families of polynomial size, i.e.: the size of C_n is bounded by some polynomial $p(n)$, and of polylogarithmic depth, i.e.: the depth of C_n is bounded by $c \log^k n$, for some c and k independently of n.

Circuit families are classified according the fan-in, that is the in-degree of the nodes of each circuit. A circuit family C is of *bounded fan-in* if there is a constant c, such that the indegree of every node in every C_n is not larger than c. C is of *semi-unbounded* fan-in if all nodes, labelled by a conjunction, have an indegree bounded by some fixed c. If the indegrees of neither the conjunctions nor the disjunctions are uniformly bounded, C is said to be of *unbounded* fan-in. Let AC^k be the class of all languages recognized by uniform circuit families of polynomial size, depth $O(\log^k n)$, and unbounded fan-in. The corresponding classes for circuit families of semi-unbounded and bounded fan-in are denoted by SAC^k and NC^k.

By definition, we have

Proposition 30 *$NC^k \subseteq SAC^k \subseteq AC^k$ for each positive integer k.*

Since a node of large indegree t labelled by a disjunction (resp. conjunction) may be replaced by a tree of disjunctions (resp. conjunctions) of height $\log t$, we have

Proposition 31 *$AC^k \subseteq NC^{k+1}$ for each $k \geq 1$.*

4.2.1 Structural Relations

Stockmeyer and Vishkin proved in [66]:

Theorem 32 *CRCW-TIME$\left(O\left(\log^k n\right)\right) = AC^k$ for each positive integer k.*

This result implies a "log-length" normal form for CRCW-PRAMs: each CRCW-PRAM A with polynomially many processors working in time O(f) can be simulated by a CRCW-PRAM B with polynomially processors working in time O(f) such that in B all values stored in global or local memory cells are polynomially bounded and thus can be stored in $O(\log n)$ bits. This kind of normal form is not known to exist for exclusive write or owner write PRAMs. Hence we had to add this restriction as a property of PRAMs as they were described in the previous subsection.

By [58] we have

Theorem 33 *$NC^k \subseteq$ OROW-TIME$\left(O\left(\log^k n\right)\right)$ for each $k \geq 1$.*

Since OROW-TIME$\left(O\left(\log^k n\right)\right)$ is a subset of DAuxPDA-TIME$\left(2^{O\left(\log^k n\right)}\right)$ (for this inclusion we need the restriction that PRAMs have a logarithmically bounded word length), this result implies the following inclusion of Ruzzo derived in [62]:

Theorem 34 *$NC^k \subseteq$ DAuxPDA-TIME$\left(2^{O\left(\log^k n\right)}\right)$ for $k \geq 1$.*

For $k = 1$ it is possible to show ([10, 72]):

Proposition 35 $NC^1 \subseteq DSPACE(\log n) \subseteq SAC^1$.

None of these inclusions is known, but all are conjectured, to be proper.

A rather surprising relation, motivating the notion of semi-unbounded fan-in, is the equation

Theorem 36 $SAC^k = NAuxPDA\text{-}TIME\left(2^{O\left(\log^k n\right)}\right)$ *for every* $k \geq 1$

by Venkatesvaran in [72]. In particular, this gives a new complexity theoretical characterization of the context-free languages:

Corollary 37 $LOG(CFL) = SAC^1$

Thus, in view of Theorem 24 circuits give a very informative characterizations of the relations between CRCW-PRAMs and NAuxPDAs in terms of the concepts of unbounded and semi-unbounded fan-in.

4.2.2 Low circuit classes

In this part the very close connections between low circuit classes and subfamilies of regular languages are shortly indicated.

The lowest circuit classes and sequential complexity classes are related by the inclusions

$$AC^0 \subset NC^1 \subseteq DSPACE(\log n) \subseteq NSPACE(\log n) \subseteq AC^1.$$

When dealing with complexity classes below $DSPACE(\log n)$ it is necessary to use reducibility notions which are finer than the usual logspace or polynomial time reductions. This leads to many-one reducibilities based on functions computable by AC^0 or NC^1 circuits or even to DLOGTIME reductions and projections [36]. We don't go into the details here, but only mention that the following relations of this subsection hold for DLOGTIME reductions. Let $DLOGTIME(\mathcal{A})$ be the class of all problems many-one reducible to some element of $\mathcal{A}$ by a DLOGTIME computable function.

The precise relation between NC^1 and $DSPACE(\log n)$ is unknown. The general conjecture assumes these classes to be different. There is a characterization of this relation in terms of formal languages. Let $LIN_{LL(1)}$ and $LIN_{LR(1)}$ denote the set of all languages generated by linear context-free grammars subject to a LL(1) respectively LR(1) condition. We then have $1\text{–}DPDA_{1-turn} = LIN_{LR(1)}$ [32]. This is the reason why $LIN_{LR(1)}$ and not the proper subset $LIN_{LL(1)}$ should be called $DLIN$, the family of *deterministic linear context-free languages*. The relation between NC^1 and $DSPACE(\log n)$ is now characterized by:

Theorem 38 **a)** $DLOGTIME\left(LIN_{LL(1)}\right) = NC^1$ *[35] and*
b) $DLOGTIME\left(LIN_{LR(1)}\right) = DSPACE(\log n)$ *[32].*

Thus we see that deterministic one-turn push down automata are $\in$-canonical.

On the other hand AC^0 is known to be a proper subset of NC^1 [24]. Again there is a characterization of this relation in terms of formal languages:

Theorem 39 **a)** $DLOGTIME(REG) = NC^1$ *[4] and*
b) $DLOGTIME(REG_{*-free}) = AC^0$ *[5].*

Here REG_{*-free} denotes the family of *star-free regular sets*. It coincides with the family of those regular sets, in which the transformation monoid of the minimal automaton doesn't contain any nontrivial group. If solvable groups are allowed, this corresponds to AC^0 circuits enriched by *mod* gates [5]. If the transformation monoid contains a nonsolvable group, the accepted language is NC^1-complete [4].

It is remarkable, that *EDOL* is of an even lower complexity:

Theorem 40 ([16, 17]) *$DLOGTIME(EDOL)$ is a proper subset of AC^0.*

4.3 Unambiguity

We will now deal with unambiguity in parallel models. As suggested by Rytter's inclusion $UCFL \subseteq CREW\text{-}TIME(O(\log n))$, the concept of exclusive access to a global memory turns out to be very closely related to unambiguity. The way to show this is by introducing unambiguity for circuits.

The notion of an unambiguous circuit is introduced in [46]. It might be explained by the intuitive notion of a *vulnerable gate*: In contrast to a normal *robust* OR-gate, a vulnerable OR-gate or vulnerable disjunction is a partially defined OR-function which works correctly on inputs containing at most one bit equal to 1. On inputs containing more than 1, the output is *undefined* and the gate is assumed to be destroyed. We assume that the value *undefined* as input of a vulnerable gate implies the output of that gate to be *undefined*, too. On the other hand, *undefined* as an input to a robust gate behaves like a value "0.5".

Unambiguous circuits consist in two types of gates:

- Small robust AND and OR-gates of bounded fan-in and
- Large vulnerable AND and OR-gates of unbounded fan-in.

According to the two types of weak and strong unambiguity for sequential machines, there are two types of unambiguous circuits:

In order to characterize CREW-PRAMs by circuits, the following notion has been introduced in [46]: A *circuit* is called *unambiguous*, iff for all combinations of the input bits no vulnerable gate receives an input containing more than one 1.

In [47] an notion corresponding to the usual version of unambiguity has been investigated: A *circuit* is called *weakly unambiguous*, iff for all combinations of the input bits the result of no destroyed vulnerable gate affects the output bit of the circuit.

That is, in a weakly unambiguous circuit we allow some vulnerable gate to be destroyed, but require that the output of every destroyed gate on its way to the output of the whole circuit is later either robustly conjoined with a 0 or robustly disjoined with a 1.

In correspondence to AC^k, we get the unambiguous classes UAC^k and $WUAC^k$. It was shown in [46] that

Theorem 41 $CREW\text{-}TIME\Big(O\left(\log^k n\right)\Big) = UAC^k$ *for each* $k \geq 1$.

When looking for unambiguous circuit classes corresponding to SAC^1, two possibilities may be considered:

- A more powerful one using large vulnerable disjunctions, small robust disjunctions, and small robust conjunctions or

- A more restricted one without the use of any robust disjunction.

In [43], the original version of [46], the first class was named USAC, and the second one URAC. But the properties of these two classes strongly suggest to choose the second possibility to be the unambiguous version of SAC circuits.[3] Thus, in the following $USAC^k$ and $WUSAC^k$ denote language classes corresponding to unambiguous circuits using vulnerable disjunctions and small robust conjunctions.

That these are the correct choice as unambiguous versions of SAC^k is illustrated by the two equations from [47]:

Theorem 42 **a)** $USAC^1 = StUAuxPDA\text{-}TIME(pol)$ *and*
b) $WUSAC^1 = UAuxPDA\text{-}TIME(pol)$.

It is remarkable how the structural properties of unambiguity (and the exclusive write feature) resemble the nondeterministic case (and the concurrent write feature).

In [46] the inclusion $NC^k \subseteq USAC^k$ was shown for all k$\geq$ 1. For the case k = 1, the more refined inclusion structure is:

$$NC^1 \subseteq DSPACE(\log n) \subseteq OROW\text{-}TIME(O(\log n))$$

$$\subseteq LOG(DCFL) \subseteq LOG(UCFL) \subseteq USAC^1.$$

These relations are summarized in Figure 3.

We mention in passing, that there is also a circuit characterization of exclusive read (see [54]).

Discussion and Open Questions

One aim of this paper was to illustrate the very many and very deep connections between complexity classes and families of formal languages. One striking example for this is the close correspondence of the following pairs: determinism versus owner access, unambiguity versus exclusive access, and nondeterminism versus concurrent access. For historical reasons, exclusive write and read are very commonly used models. The connection to unambiguity explains the many nonconstructive aspects of this concept, e.g. the apparent nonexistence of complete problems. Even worse, for polynomial size, i.e.: for logarithmically space bounded Turing machines or auxiliary push-down automata, there is no single problem known, being a member of some unambiguous class, which is suspect of being not an element of the corresponding deterministic class. Relations like that are only known for families; e.g.: $UCFL$, which is known to be in $UAuxPDA\text{-}TIME(pol)$, is conjectured not to be in $DAuxPDA\text{-}TIME(pol)$. Thus the use of PRAMs of exclusive access type to solve a single problem should be regarded as a bit lazy. It might be probable, that all existing $XREW$-PRAM algorithms solving specific problems could be converted into $XROW$-algorithms. It would be very interesting to single out a specific problem resisting this approach, since this would be a candidate of a single problem known to be unambiguously solvable, but apparently not deterministically in polynomial size.

We close with a list of open questions:

[3] Some new results concerning the classes built according to the first possibility may be found in [52].

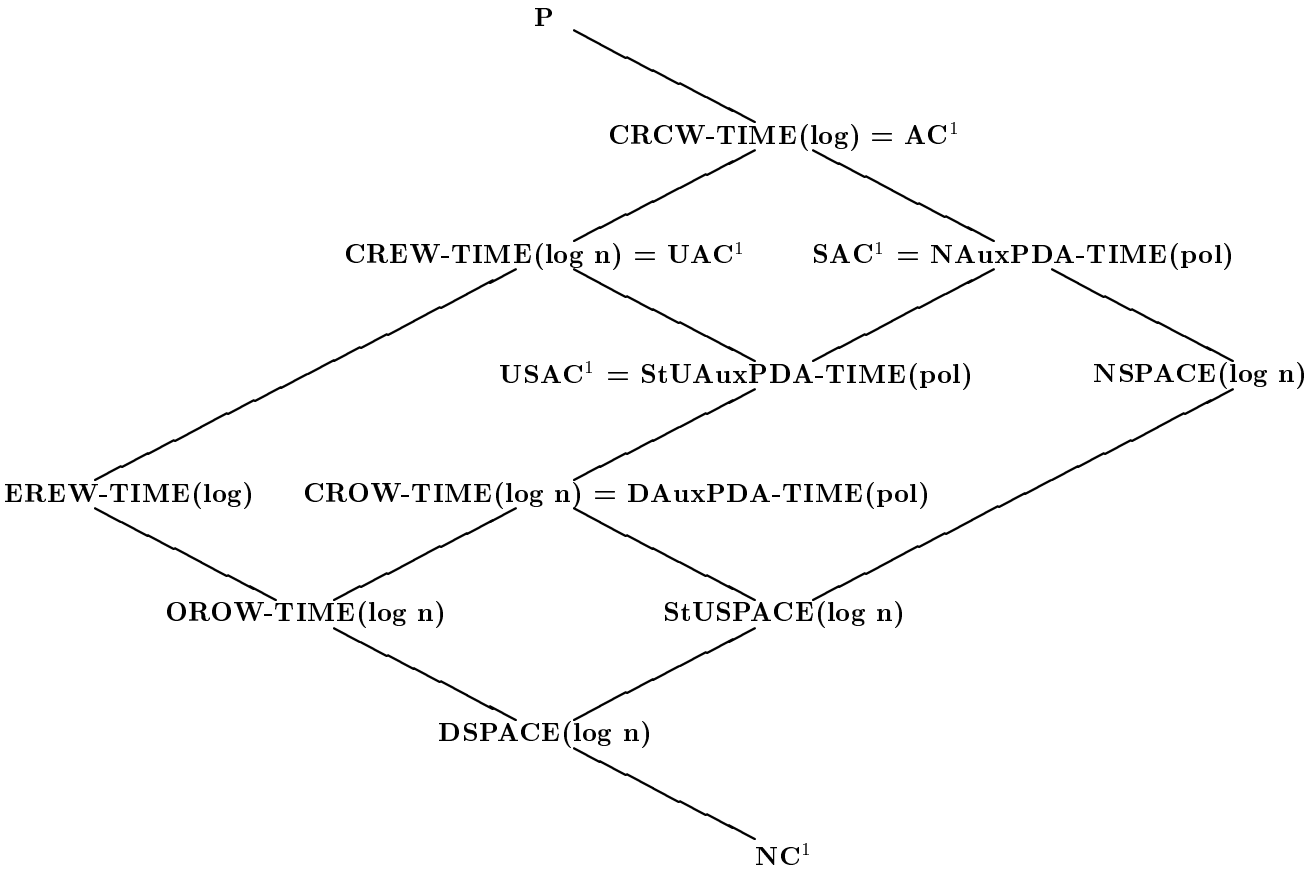

Figure 3: The *NC*-Structure between NC^1 and AC^1

1. Are there families of formal languages which are complete for $SSPACE(\log n)$? In particular, is there a reasonable definition of a symmetric (one-turn) push down automaton which would be $\in$-canonical?

2. Under which conditions storage types are $\in$-canonical or fully canonical? (See [20, 18] for $\emptyset$-canonical storage types.)

3. Are there families of formal languages complete for AC^k, k$\geq$ 1, or for NC^k, k$\geq$ 2?

4. Is every family of formal languages which is effectively closed under the AFL operations and which has a decidable emptiness problem contained in NP?

5. Sequential and parallel models provide a unifying framework to structure families of formal languages. In particular, this is true for boolean circuits. It would be interesting to find a circuit feature representing determinism as a subcase of nondeterminism. In particular this would mean to characterize $DSPACE(\log n)$, $LOG(DCFL)$, or P by circuits of logarithmic depth.

6. Are there versions of PRAMs or circuits which correspond to symmetric complexity classes?

References

[1] A. Aho. Nested stack automata. *J. Assoc. Comp. Mach.*, 16:383–406, 1969.

[2] P. Asveld. Time and space complexity of inside-out macro languages. *Internat. J. Comput. Math.*, 10:3–14, 1981.

[3] J. Balcácar, J. Diáz, and J. Gabárro. *Structural Complexity Theory I.* Springer, 1988.

[4] D.A. Barrington. Bounded-width polynomial-size branching programs can recognize exactly those languages in NC^1. *J. Comp. System Sci.*, 38:150–164, 1989.

[5] D.A. Barrington, N. Immerman, and H. Straubing. On uniformity conditions within NC^1. In *Proc. 3rd Structure in Complexity Theory*, pages 47–59. IEEE, 1988.

[6] J. Berstel. *Transductions and Context-Free Languages.* Teubner Verlag, Stuttgart, 1979.

[7] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Inform. and Control*, 55:80–88, 1982.

[8] R. Book and S. Greibach. Quasi-realtime languages. *Math. System Theory*, 4:97–111, 1970.

[9] R. Book, S. Greibach, and B. Wegbreit. Time- and tape-bounded turing acceptors and AFLs. *J. Comp. System Sci.*, 4:606–621, 1970.

[10] A. Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, 6(4):733–744, 1977.

[11] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace-MOD-classes. In *Proc. 7th Symposium Theoretical Aspects of Computer Science*, number 480 in LNCS, pages 360–371. Springer, 1991.

[12] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Proc. of the 8th Conference on Fundamentals of Computation Theory*, number 529 in LNCS, pages 168–179, 1991.

[13] S. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. Assoc. Comp. Mach.*, 18:4–18, 1971.

[14] S. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proc. of the 11th Annual ACM Symp. on Theory of Computing*, pages 338–345, 1979.

[15] E. Dahlhaus and M. K. Warmuth. Membership for growing context-sensitive grammars is polynomial. *J. Comp. System Sci.*, 33:456–472, 1986.

[16] C. Damm, M. Holzer, and K.-J. Lange. The parallel complexity of iterated morphisms and the arithmetic of small numbers. In *Proc. 17th Symposium on Mathematical Foundations of Computer Science*, number 629 in LNCS, pages 227–235. Springer, 1992.

[17] C. Damm, M. Holzer, K.-J. Lange, and P. Rossmanith. The very low complexity of deterministic 0L languages: D0L is in AC^0. In *Proc. Developments in Language Theory*, 1993. to appear.

[18] J. Dassow and K.-J. Lange. Complexity of automata with abstract storages. In *Proc. of the 8th Conference on Fundamentals of Computation Theory*, number 529 in LNCS, pages 200–209. Springer, 1991.

[19] P. Dymond and W. Ruzzo. Parallel RAMs with owned global memory and deterministic context-free language recoginition. In *Proc. of 13th International Colloquium on Automata, Languages and Programming*, number 226 in LNCS, pages 95–104. Springer, 1986.

[20] J. Engelfriet. Iterated pushdown automata and complexity classes. In *Proc. on the 15th Annual ACM Symp. on Theory of Computing*, pages 365–373, 1983.

[21] M.J. Fischer. Grammars with macro-like production. Ph.d. thesis, Harvard Univ., 1968.

[22] P. Flajolet and J. Steyaert. Complexity of classes of languages and operators. Rap. de Recherche 92, IRIA Laboria, Nov. 1974.

[23] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing*, pages 114–118, 1978.

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

[25] S. Ginsburg. *Formal Languages.* North-Holland, Amsterdam, 1975.

[26] S. Ginsburg, S. Greibach, and M. Harrison. One-way stack automata. *J. Assoc. Comp. Mach.*, 14:389–418, 1967.

[27] L. M. Goldschlager. A unified approach to models of synchronous parallel computation. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing*, pages 89–94, 1978.

[28] S. Greibach. The hardest context-free language. *SIAM J. Comp.*, 2:304–310, 1973.

[29] M.A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading Mass., 1978.

[30] J. Hartmanis and S. Mahaney. Languages simultaneously complete for one-way and two-way log-tape automata. *SIAM J. Comp.*, 10:383–390, 1981.

[31] M. Holzer and K.-J. Lange. On the complexity of operations on formal languages. In preparation.

[32] M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proc. of the 9th FCT*, number 710 in LNCS, pages 299–308. Springer Verlag, 1993.

[33] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison-Wesley, Reading Mass., 1979.

[34] O. Ibarra. Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata. *J. Comp. System Sci.*, 5:88–117, 1971.

[35] O.H. Ibarra, T. Jiang, and B. Ravikumar. Some subclasses of context-free languages in NC^1. *Information Processing Letters*, 29:112–117, 1988.

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

[37] N. Jones and S. Skyum. Recognition of deterministic ETOL languages in logarithmic space. *Inform. and Control*, 35:177–181, 1977.

[38] N. Jones and S. Skyum. Complexity of some problems concerning L systems. *Math. Systems Theory*, 13:29–43, 1979.

[39] M. Karchmer and A. Wigderson. On span programs. In *Proc. of the 8th IEEE Structure in Complexity Theory Conference*, pages 102–111, 1993.

[40] R. Ladner and N. Lynch. Relativization of questions about log space computability. *Math. Systems Theory*, 10:19–32, 1976.

[41] K.-J. Lange. Context-free controlled ETOL systems. In *Proc. of 9th International Colloquium on Automata, Languages and Programming*, number 154 in LNCS, pages 723–733. Springer, 1983.

[42] K.-J. Lange. Decompositions of nondeterministic reductions. *Theoret. Comput. Sci.*, 58:175–181, 1988.

[43] K.-J. Lange. Unambiguity of circuits. In *Proc. of the 5th IEEE Structure in Complexity Conference*, pages 130–137, 1990.

[44] K.-J. Lange. Complexity and structure in formal language theory. In *Proc. of the 8th IEEE Structure in Complexity Conference*, pages 224–238, 1993.

[45] K.-J. Lange. A note on the P-completeness of deterministic one-way stack languages. In preperation, 1993.

[46] K.-J. Lange. Unambiguity of circuits. *Theoret. Comput. Sci.*, 107:77–94, 1993.

[47] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in LNCS, pages 399–406. Springer, 1990.

[48] K.-J. Lange and M. Schudy. The complexity of the emptiness problem for EOL systems. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer Systems*, pages 167–175, Berlin, 1992. Springer.

[49] P. Lewis and C.H. Papadimitriou. Symmetric space-bounded computation. *Theoret. Comput. Sci.*, 19:161–187, 1982.

[50] P. Lewis, R. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pages 191–209, 1965.

[51] B. Monien. About the deterministic simulation of nondeterministic ($\log n$)-tape bounded turing machines. In *2-te GI Fachtagung Automatentheorie und Formale Sprachen*, number 33 in LNCS, pages 118–126. Springer, 1975.

[52] R. Niedermeier and P. Rossmanith. Unambiguous simulations of auxiliary pushdown automata and circuits. *Inform. and Control*, 1993. (to appear).

[53] I. Niepel. Logarithmisch platzbeschränkte Kompleitätsklassen - Charakterisierung und offene Fragen. Diplomarbeit, Universität Hamburg, 1987. (in German).

[54] I. Niepel and P. Rossmanith. Uniform circuits and exclusive read PRAMs. In *Proc. of the 11th FST&TCS*, number 560 in LNCS, pages 307–318. Springer Verlag, 1990.

[55] N. Nisan. $RL \subseteq SC$. In *Proc. of the 24th Annual ACM Symposium on Theory of Computing*, pages 619–623, 1992.

[56] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proc. of 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 24–29, 1992.

[57] K. Reinhardt. Counting and empty alternating pushdown automata. In *Proc. 7th International Meeting of Young Computer Scientists*, pages 198–207, Smolenice Castle, Tschechoslowakei, 1992.

[58] P. Rossmanith. The owner concept for PRAMs. In *Proc. of the 8th STACS*, number 480 in LNCS, pages 172–183. Springer, 1991.

[59] W. C. Rounds. Complexity of recognition in intermediate-level languages. In *Proc. of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 145–158, 1973.

[60] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.

[61] W. Ruzzo. Tree-size bounded alternation. *J. Comp. System Sci.*, 21:218–235, 1980.

[62] W. Ruzzo. On uniform circuit complexity. *J. Comp. System Sci.*, 22:365–338, 1981.

[63] W. Ruzzo, J. Simon, and M. Tompa. Space – bounded hierarchies and probabilistic computations. *J. Comp. System Sci.*, 28:216–230, 1984.

[64] W. Rytter. Parallel time O(log n) recognition of unambiguous context-free languages. *Inform. and Control*, 73:75–86, 1987.

[65] E. Shamir and C. Beeri. Checking stacks and context-free programmed grammars accept p-complete languages. In *Proc. of 2nd ICALP*, number 14 in LNCS, pages 277–283. Springer, 1974.

[66] L. Stockmeyer and C. Vishkin. Simulation of random access machines by circuits. *SIAM J. Comp.*, 13:409–422, 1984.

[67] I. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *J. Assoc. Comp. Mach.*, 22:499–500, 1975.

[68] I. Sudborough. On tape-bounded complexity classes and multi-head finite automata. *J. Comp. System Sci.*, 10:62–76, 1975.

[69] I. Sudborough. The complexity of the membership problem for some extensions of context-free languages. *Internat. J. Comput. Math. SECT A*, 6:191–215, 1977.

[70] I. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assoc. Comp. Mach.*, 25:405–414, 1978.

[71] J. van Leeuwen. The membership question for ETOL languages is polynomially complete. *Information Processing Letters*, 3:138–143, 1975.

[72] H. Venkateswaran. Properties that characterize LOGCFL. In *Proc. of the 19th Annual ACM Symp. on Theory of Computing*, pages 141–150, 1987.

[73] T. Verbeek. Time-space trade-offs for general recursion. In *Proc. of 22th Annual IEEE Symposium on Foundations of Computer Science*, pages 228–234, 1981.

[74] D. Younger. Recognition and parsing of context-free languages in time n^3. *Inform. and Control*, 10:189–208, 1967.