UNAMBIGUOUS AUXILIARY PUSHDOWN AUTOMATA AND SEMI-UNBOUNDED FAN-IN CIRCUITS*

ROLF NIEDERMEIER AND PETER ROSSMANITH

Fakultät für Informatik
Technische Universität München
80290 München
Federal Republic of Germany

*This research was supported by the Deutsche Forschungsgemeinschaft, SFB 342, Teilprojekt A4 “KLARA”.

This paper is a revised and expanded version of a paper (Lange and Rossmanith, 1990) presented at the International Conference “Mathematical Foundations of Computer Science” held in Banská Bystrica, Czechoslovakia, August 27–31, 1990 and of a paper (Niedermeier and Rossmanith, 1992) presented at the International Conference “Latin American Theoretical INformatics” held in São Paulo, Brazil, April 6–10, 1992.

Proposed running head: Unambiguous AuxPDAs and Circuits

Address for correspondence:

Peter Rossmanith
Technische Universität München
Fakultät für Informatik
Arcisstr. 21
80333 München
Fed. Rep. of Germany

Abstract

Notions of unambiguity for uniform circuits and AuxPDAs are studied and related to each other. In particular, a coincidence for counting and unambiguous versions of AuxPDAs and semi-unbounded fan-in circuits is shown. Moreover, an improved simulation of LOGUCFL (the class of languages logspace many-one reducible to unambiguous context-free languages) by unambiguous circuits and AuxPDAs is developed. Next, an inductive counting technique on semi-unbounded fan-in circuits is presented and employed for several applications, especially an alternative proof for the closure under complementation of LOGCFL. A cost-free simulation of polynomially ambiguity bounded AuxPDAs by unambiguous ones is given. A first nontrivial upper bound for a circuit class defined by Lange and its closure under complementation are indicated. Finally, a normal form for AuxPDAs is investigated. Inter alia it is shown that for unambiguous AuxPDAs operating in polynomial time and logarithmic space a push-down height of $O(\log^2 n)$ suffices, thus paralleling results for deterministic and nondeterministic AuxPDAs. It is pointed out that without loss of generality the underlying machines of the most important AuxPDA classes work obliviously.

1. Introduction

The major aim of computational complexity theory is to decide which problems are efficiently solvable. To tackle this question, in general three cases are distinguished (Parberry, 1987):

(a) Efficiency in the sequential case means *polynomial time* computations,
(b) efficiency in the parallel case with unlimited parallelism leads to complexity classes within *polylogarithmic space* due to the parallel computation thesis (Goldschlager, 1982).
(c) efficiency in the parallel case with a limited amount of hardware, i.e., a polynomial number of processors, yields the class of problems commonly called NC.

In all three cases the nondeterminism vs. determinism problem plays a decisive role for the development of efficient algorithms.

In this paper we will deal with classes within the NC-hierarchy. NC is a fairly robust class. It was introduced by Pippenger (1979) and named by Cook (1979). NC can be characterized in terms of several parallel models, in particular PRAMs (Fortune and Wyllie, 1978; Goldschlager, 1978), alternating Turing machines (Chandra et al., 1981), uniform circuits (Borodin, 1977; Ruzzo, 1981), and polynomially time bounded auxiliary push-down automata (AuxPDAs) (Cook, 1971). For the question of determinism vs. nondeterminism within NC AuxPDAs are the most suitable model. An AuxPDA is a space bounded Turing machine with an additional unbounded push-down store. Since AuxPDAs are special Turing machines we immediately have deterministic and nondeterministic versions. For the moment, we only consider AuxPDAs that are simultaneously logarithmically space bounded and polynomially time bounded. These machines show strong relations to context free languages (CFLs) or, to be more precise, to their closure under log-space many-one reductions: Sudborough (1978) characterized LOGCFL by nondeterministic AuxPDAs and LOGDCFL by deterministic AuxPDAs. So the question of nondeterminism vs. determinism can be stated as LOGDCFL $\stackrel{?}{=}$ LOGCFL. One obvious approach to this question is to investigate a natural concept between these two: *unambiguity*. The concept of unambiguity took its origin in the theory of formal languages, where the demand for the existence of at most one derivation tree led to the consideration of unambiguous CFLs (UCFLs). In this way we naturally get LOGUCFL as a class between LOGDCFL and LOGCFL, whose relation to AuxPDA classes and the NC-hierarchy will be investigated in this paper. In the field of polynomial time the concept of unambiguity led to UP introduced by Valiant (1976), for space bounded computation we have e.g. UL (Àlvarez and Jenner, 1993; Buntrock et al., 1991). UP (resp. UL) consists of those languages accepted by polynomially time bounded (resp. logarithmically space bounded) Turing machines which have at most *one accepting computation path*. Machines restricted in this way are commonly called *unambiguous*.

Whereas UP and UL already obtained considerable attention, the task of this work will be to investigate unambiguity for AuxPDAs, i.e., within the NC-hierarchy.

Recently, another approach towards unambiguity in the NC-hierarchy was made by Lange (1993). He defined unambiguous circuits in order to provide the up to then lacking characterization of CREW-PRAMs in terms of circuits. And indeed, he characterized CREW-PRAMs by an unambiguous version of AC-circuits, thus ending the isolation of this important PRAM class. Note that in contrast to CRCW-PRAMs (Stockmeyer and Vishkin, 1984) and CROW-PRAMs (Dymond and Ruzzo, 1986) up to then no characterization of CREW-PRAMs by another computational model was known. A first hint for the "unambiguous behavior" of CREW-PRAMs was given earlier by Rytter (1987), who showed that LOGUCFL is contained in the class of languages recognized by CREW-PRAMs in logarithmic time using polynomially many processors. The central idea of Lange's definition of unambiguous circuits is the introduction of *vulnerable* gates, i.e., gates which may receive at most one input with value 1 (in the case of OR-gates) resp. 0 (in the case of AND-gates). However, this definition seems to differ considerably from the conventional notion of unambiguity for automata given above. Thus in this work we introduce so-called weakly unambiguous circuits, which, by definition, have at most one accepting subtree. And indeed it will turn out that those circuits capture the conventional notion of unambiguity for AuxPDAs. On the other hand, we introduce the notion of strong unambiguity for automata, which corresponds to Lange's definition of unambiguous circuits. In this way one of the implicit consequences of this work will be the demand for two notions of unambiguity — one for the world of NC, i.e., strong unambiguity, and one for the sequential world, i.e. (weak) unambiguity.

In the following we provide a survey on the internal structure of the paper including concise statements of the main results. In the next section all the basic notions and definitions needed are supplied. Afterwards, in Section 3, we introduce strong and weak unambiguity and some generalizations for circuits as well as for AuxPDAs.

In Section 4 we develop new, improved simulations between semi-unbounded fan-in circuits and AuxPDAs. The major aim of this section is to ameliorate Venkateswaran's equality $\mathrm{SAC}^k = \mathrm{NAPDA}^k$ (Venkateswaran, 1991) first of all in order to get simulations where *one* accepting computation is simulated by *one* accepting subtree. This is necessary to provide simulations between the corresponding unambiguous models. Venkateswaran's simulation does not have this property. By way of contrast, it is well known that most reductions between NP-complete problems preserve the number of solutions. For example, this holds for the satisfiability problem or Hamiltonian paths. Unfortunately, this is not true at all in the case of SAC^1 and NAPDA^1. Venkateswaran's simulation of AuxPDAs by circuits incorporates a number of accepting subtrees which exceeds the original number of accepting paths tremendously. Our improved simulation will yield the equality of the counting versions of the above classes for $k = 1$, i.e., $\#\mathrm{SAC}^1 = \#\mathrm{APDA}^1$ and provides characterizations of weakly and strongly unambiguous AuxPDAs by their semi-unbounded fan-in counterparts in the field of circuits. Note that to get an analogous result to $\mathrm{SAC}^1 = \mathrm{NAPDA}^1$ in the unambiguous world is particularly useful, since such a simulation result allows to study properties of unambiguous AuxPDAs in the context of a static, combinatorial model. A further result of Section 4 will be the inclusion of LOGUCFL in a strongly unambiguous circuit class. On the one hand this improves a result of Lange (1993), where only the inclusion of LOGDCFL in this class is shown. On the other hand, it also ameliorates Rytter's inclusion $\mathrm{LOGUCFL} \subseteq \mathrm{CREW}^1$ (Rytter, 1987), since the above circuit class is clearly included in CREW^1.

In Section 5 a new inductive counting technique for semi-unbounded fan-in circuits is presented, which due to the characterization results of Section 4 enables the counting of accepting paths of an AuxPDA. For a lot of problems the ability to guess nondeterministically is a crucial prerequisite for solving them efficiently. However, nondeterministic computations involve up to exponentially many accepting computations. There may be problems which are really in need of such an abundance of nondeterminism, but there also may be problems where a smaller amount of nondeterminism, say a polynomially bounded number of computation paths between arbitrary configurations, suffices. Simulations of nondeterminism by determinism are known only by machines which need drastically more resources. This is true for space bounded, time bounded, and AuxPDA classes. The best known results in these settings are

(a) $\mathrm{NP} \subseteq \mathrm{DEXP}$,
(b) $\mathrm{NL} \subseteq \mathrm{DSPACE}(\log^2 n)$,
(c) $\mathrm{NAPDA}^1 \subseteq \mathrm{DAPDA}^2$.

Until now it is still not clear whether the simulation of nondeterministic AuxPDAs can be done efficiently by deterministic ones when the ambiguity is limited. Notwithstanding, we will show that an unambiguous AuxPDA can simulate a nondeterministic one much better than a deterministic one if the degree of nondeterminism is not too high. It is proved that polynomial ambiguity bounded AuxPDAs operating in polynomial time and logarithmic space can be simulated by unambiguous ones within the *same* time and space bounds. A similar result for space bounded computations was obtained by Buntrock et al. (1993). However, here unambiguous computations beat deterministic ones only for sub-polynomial ambiguity*, and a simulation without space and time penalty is only possible for constant ambiguity. Our simulation can deal with polynomial ambiguity with neither

*Buntrock et al. (1993) regarded the special case of polynomial ambiguity as the most interesting one. Nevertheless, in (Buntrock et al., 1991) it was shown that in this case even a simulation by deterministic AuxPDAs (instead of unambiguous AuxPDAs) within the same time and space bounds exists.

time nor space penalty. In addition, there are exceptions where our simulation technique still yields better results: For a certain class of semi-unbounded fan-in circuits defined in (Lange, 1993) which even involve exponential ambiguity (i.e., an exponential number of accepting subtrees which is the highest number possible even for unambiguous circuits) nevertheless a "cost-free" simulation by unambiguous AuxPDAs is possible. This answers an open question posed by Lange. In addition, it is demonstrated that this circuit class is closed under complementation. Finally, it should be noted that a corollary to the basic inductive counting lemma of this section provides a new proof for the closure under complementation of SAC^k (and, as a consequence, of LOGCFL), a result due to Borodin et al. (1989).

In Section 6 some new normal form theorems for AuxPDAs are proved and several old ones are improved. In particular, we show that deterministic, nondeterministic, strongly and weakly unambiguous AuxPDAs work without loss of generality obliviously. That is, the movements of all working-heads do not depend on the input except its length. Prior to this, we obtain the restriction of push-down heights especially for unambiguous AuxPDAs. These effects all are enabled by the characterizations of AuxPDAs by circuits and complete results given in (Dymond and Ruzzo, 1986) and (Ruzzo, 1980).

In the end, in Section 7, we will briefly recapitulate the main techniques and results of this work. Moreover, we discuss perspectives for future work and remaining open questions.

2. Preliminaries

We assume familiarity with basic facts and definitions of structural complexity theory as to be found in (Balcázar et al., 1990), (Hopcroft and Ullman, 1979), or (Wagner and Wechsung, 1986). In order to keep the paper readable for the nonspecialist reader, here we provide the central notions used in this work.

Without loss of generality, we will only consider languages over alphabet $\{0,1\}$ whose symbols also will be interpreted as Boolean values true and false.

PRAMs (parallel random access machines), introduced by Fortune and Wyllie (1978) and Goldschlager (1978), only play a minor role in this paper. So we only define the PRAM-complexity classes and refer to the literature, e.g., (Karp and Ramachandran, 1990; Parberry, 1987), for details. PRAMs are classified accordingly to the settlement (concurrent (C), exclusive (E), owner (O)) of read and write conflicts on global memory. With $\mathrm{XRYW}^k, k \geq 1, X, Y \in \{C, E, O\}$ we denote the classes of languages which are recognizable in time $O(\log^k n)$ by XRYW-PRAMs using polynomially many processors.

(Boolean) Circuits are one of the two fundamental computational models of this paper. A circuit for inputs of size n is an acyclic directed graph whose nodes (called gates) are labeled with Boolean operators. Nodes of indegree zero are labeled from the set $\{0, 1, x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$, where $x_1, \ldots, x_n$ are the input nodes of the circuit

and $\overline{x_i}$ ($1 \leq i \leq n$) denotes the negated value of x_i. All the other nodes are labeled as either AND- or OR-gates. Note that we do not include negation gates in circuits. In general, due to De Morgan's laws this means no restriction, because we can 'push' negations to the input gates. However, it is a restriction for so-called semi-unbounded fan-in circuits. Since we are only interested in circuits accepting languages, our circuits have exactly one node with outdegree zero, that is the output gate. A circuit *accepts* a string $w \in \{0,1\}^n$ iff its output gate evaluates to 1 on input w. Observe that in the context of circuits one often speaks of fan-in (resp. fan-out) instead of indegree (resp. outdegree). The *size* of a circuit is the number of gates it contains. The *depth* is the length of the longest directed path from some input gate to the output gate.

A *circuit family* $\{C_n \mid n \in \mathbb{N}\}$ is an infinite set of circuits, where C_n is a circuit for inputs of size n. An important requirement for circuits is that of *uniformity*. A circuit family $\{C_n \mid n \in \mathbb{N}\}$ is called logspace-uniform iff there exists a deterministic Turing machine which computes a function $n \rightarrow \langle C_n \rangle$ (where $\langle C_n \rangle$ is an encoding of circuit C_n) in space $\log n$ (see (Ruzzo, 1981) for details).

Next, we define three basic circuit complexity classes. We distinguish between the fan-in allowed for AND- and OR-gates. That is, unbounded fan-in means that all gates may have unbounded (non-constant) indegree, bounded fan-in means that all gates have to have constant indegree (w.l.o.g. indegree two), and semi-unbounded fan-in means that OR-gates may have unbounded fan-in, but AND-gates have bounded fan-in. So we have NC^k (resp. SAC^k, AC^k), $k \geq 1$, as the classes of languages recognized by polynomial size, $O(\log^k n)$ depth bounded, logspace-uniform circuits with bounded (resp. semi-unbounded, unbounded) fan-in.

The following normal form for circuits will be of central importance for some of the proofs in this paper. A circuit C is called *leveled* if all gates of C in depth d receive their inputs only from gates in depth $d-1$. This normal form is easily achieved for the above defined circuit classes.

Proposition 1. *For* NC^k, SAC^k, *and* AC^k *it can be assumed that only leveled circuits are used.*

Proof. (construction) Let C be a (non-leveled) circuit and let $\tilde{C}$ denote an equivalent, leveled circuit to be constructed. For each depth of $\tilde{C}$ we make a replica of each gate of C. This again yields polynomial size and the same depth as C. The replicas are constructed as follows:

(1) In depth 0, there are only inputs and constant gates. The replicas of AND- and OR-gates are gates with constant value 0.
(2) In depth $i > 0$, the replica of an AND-gate (resp. an OR-gate) of C again is an AND-gate (resp. OR-gate) whose inputs are the replicas in depth $i-1$ of the inputs of the "original" gate in C. The replicas of input and constant gates

simply are fan-in two AND-gates whose inputs are the replicas in depth $i-1$ of the "original" gates.

The equivalence of C and $\tilde{C}$ with respect to the recognized languages can be proved by a simple induction on the circuit depth. □

A similar construction is used in (Borodin et al., 1989), where an even stronger normal form is generated. It is easy to see that an analogous result to Proposition 1 also holds for most of the circuit classes defined in the next section.

The second fundamental computational model in this work are *auxiliary push-down automata* (AuxPDAs) introduced by Cook (1971). An AuxPDA is a Turing machine with unrestricted push-down store in addition to the working tape. We will consider AuxPDAs with simultaneous bounds on time and space. Observe that the space on the push-down store is "free," i.e., it does not count for the space bound. In this paper we will concentrate on AuxPDAs with a polynomial running time and a poly-log space working tape.

As in the case of circuits, we will examine AuxPDAs given in some normal form. As usual, we require that accepting computations always end up with an empty push-down, an empty working tape, all heads at a fixed position and an uniquely determined final state. Altogether, this means that we have exactly one accepting configuration. Furthermore, we require the AuxPDAs always to push on or pop from the push-down in each computation step. Observe that these demands do not mean any restrictions for nondeterministic or deterministic AuxPDAs.

An important notion for AuxPDAs is that of *surface configurations* (Cook, 1971). A surface configuration of an AuxPDA consists of the topmost symbol on the push-down store, the actual state, the contents of the working tape, and the positions of the heads. Please note that in this way we exclude the contents of the push-down store except for the topmost symbol. Surface configurations stand in close relation to *profiles* of computations. A profile is a graph which plots push-down height versus running-time. (The name profile was introduced by Vinay (1991).) In profiles for each time step we may enter surface configurations, thus describing a computation fully by its surface configurations and the "push-down behavior."

The classes of languages recognized by deterministic (nondeterministic), logarithmically space bounded AuxPDAs in time $2^{O(\log^k n)}$ are denoted by DAPDA^k (NAPDA^k).

We complete this section with two general notions for complexity classes. A prefix 'F' is used to denote the class of functions instead of the class of languages computed by deterministic (or, as we will see later on, unambiguous) machines. Herein, the output is placed on a special write-only-tape which does not count for the space bound. For example, FP (resp. FAPDA^k) denotes the functional classes corresponding to P (resp. DAPDA^k). For nondeterministic machines (and also for circuits) we make use of the counting operator #. This results in functions computing the number of accepting computations of nondeterministic machines for some fixed input. For example, #P (resp. $\#\mathrm{NAPDA}^k$) are the classes of functions which map input words to the number of accepting computations on an NP- (resp. NAPDA^k-) machine.

3. Unambiguity

In recent time the concept of unambiguity has won considerable attention in sequential as well as in parallel complexity theory (e.g. (Buntrock et al., 1993; Hartmanis and Hemachandra, 1988; Lange, 1993; Rytter, 1987; Valiant, 1976)). Additionally, unambiguity plays an important role in cryptography (Grollmann and Selman, 1988), formal languages (where this concept originally comes from) (Harrison, 1978; Hopcroft and Ullman, 1979), and also shows tight connections to nondeterministic function classes: A commonly accepted functional analog of NP is the class SVNP (single valued NP) introduced by Hartmanis and Yesha (1984), which is defined in terms of machines that output a function value on exactly one accepting path. In fact, this means that these machines have to be unambiguous. In general, a machine is called *unambiguous* if it has at most one accepting computation path for arbitrary input words. Note that it is undecidable whether a machine is unambiguous, see, e.g., (Hartmanis and Hemachandra, 1988). Subsequently, we will refine this notion for the purpose of a more precise handling of parallel complexity classes. In particular, the fine structure between NC^1 and AC^1 will be investigated by means of various unambiguous circuit and AuxPDA classes.

3.1. Unambiguity of circuits. Unambiguous circuits were introduced by Lange (1993)† This was done in order to get a characterization of CREW-PRAMs in terms of circuits, thus ending the "isolation" of this important PRAM class.

In order to define unambiguous circuits, we have to introduce the notion of *vulnerable* gates. An OR-gate (resp. AND-gate) is called vulnerable if it does not receive a 1 (resp. 0) by two or more of its predecessors. Otherwise, the value of the gate is undefined.

Definition 2. (1) $\mathrm{UnambAC}^k$ (resp. $\mathrm{UnambS}^{E}\mathrm{AC}^k$) denotes the class of languages recognized by AC^k- (resp. SAC^k-) circuits which fulfill the additional requirement that all unbounded gates are vulnerable and none of them ever has an undefined value.

(2) $\mathrm{UnambSAC}^k$ is defined similar to $\mathrm{UnambS}^{E}\mathrm{AC}^k$, but here additionally the OR-gates of bounded fan-in have to be vulnerable.

†Lange (1993) changed the notation of unambiguous circuit complexity classes compared to the preliminary version (Lange, 1990; Lange and Rossmanith, 1990; Niedermeier and Rossmanith, 1992). For the sake of standardization we adopt the notion of (Lange, 1993).

In this setting Lange proved UnambAC^k = CREW^k ($k \geq 1$). Here we will concentrate on the two semi-unbounded fan-in classes and investigate their relations to AuxPDAs and the NC-hierarchy.

The above notion of unambiguity for circuits seems to be rather different from the conventional concept for automata mentioned in the beginning. But the requirement for the existence of at most one accepting computation path for automata can be naturally found again in the field of circuits. Here one has to demand that there exists at most one accepting subtree for arbitrary input words. An accepting subtree $T(C)$ of a circuit C is defined analogously to an accepting subtree of an automaton (Venkateswaran, 1991):

- $T(C)$ includes the output gate of C,
- for any AND-gate g included in C, *all* inputs of g in C have to be included in $T(C)$ as inputs of g,
- for any OR-gate g included in C, *exactly one* input of g in C has to be included in $T(C)$ as input of g,
- any constant gate or input gate included in $T(C)$ must have value one.

This leads to the following definition of weakly unambiguous, semi-unbounded fan-in circuits.

Definition 3. The class WeakUnambSAC^k consists of all languages recognized by SAC^k-circuits that have at most one accepting subtree.

Another possibility to define *weakly* unambiguous circuits is to demand the same restrictions as in the strongly unambiguous case, but only for gates within accepting subtrees, not for the whole circuit. For WeakUnambSAC-circuits this would mean that all OR-gates included in accepting subtrees must be vulnerable. With the help of this second way of defining unambiguity for circuits it is possible to define WeakUnambSAC^k and WeakUnambAC^k in an analogous way. One simply demands that within accepting subtrees all *unbounded* fan-in gates are vulnerable.

Finally, we define an extension of circuits belonging to the class UnambSAC^k, i.e., so-called ambiguity bounded circuits. Here, corresponding to the notion of strong unambiguity, we demand that the number of accepting subtrees of each gate is bounded by some function in the input length.

Definition 4. Ambiguous-SAC$^k(a(n))$ is the class of all languages recognized by SAC^k-circuits, where all gates have at most $a(n)$ accepting subtrees.

By definition, Ambiguous-SAC$^k(1)$ coincides with UnambSAC^k.

3.2. Unambiguity of AuxPDAs. Independent from the distinction between weakly and strongly unambiguous circuits, there are also two natural notions of unambiguity for AuxPDAs which we will again call weak and strong unambiguity.

First, we define complexity classes obtained via (conventional) weakly unambiguous AuxPDAs.

Definition 5. The class of languages recognized by logarithmically space bounded and $2^{O(\log^k n)}$ time bounded AuxPDAs, which have at most one accepting computation path, is denoted by UnambAPDA^k.

The unambiguous circuits defined by Lange (and, therefore, CREW-PRAMs) correspond to the notion of strong unambiguity. Therefore, we also take a look at strong unambiguity for automata. An automaton is called *strongly unambiguous* if there is at most one computation path between any two of its configurations. Please note that this includes configurations that are not even reachable from the initial configuration and that this restriction must hold for every possible input word. In terms of AuxPDAs we get the following complexity classes.

Definition 6. The class of languages recognized by $\log n$ space and $2^{O(\log^k n)}$ time bounded strongly unambiguous AuxPDAs is denoted by StUnambAPDA^k.

Observe that strong unambiguity is a concept fairly near to determinism. The additional power strong unambiguity provides in comparison to determinism is fairly small, because in both concepts there is only one path allowed between two arbitrary configurations. Nevertheless, both notions seem to differ due to the fact LOGDCFL = DAPDA^1 (Dymond and Ruzzo, 1986) and the inclusion $LOGUCFL \subseteq$ StUnambAPDA^1 (Theorem 16). The latter inclusion reveals that unambiguity in the world of formal languages (where, anyway, this concept took its origin) corresponds to *strong* unambiguity. The reason for this is the possibility to eliminate useless nonterminals which yields an unambiguous grammar $G = (N, T, P, S)$ for which *all* leftmost derivations $A \overset{*}{\Rightarrow} \alpha, A \in N, \alpha \in (N \cup T)^*$ are unique.

This property was crucial for Rytter's inclusion LOGUCFL $\subseteq$ CREW^1 (Rytter, 1987). Further evidence for the strong unambiguity of formal languages is given by the inclusion of unambiguous linear context-free languages in strongly unambiguous, logarithmic space (that is UnambLIN $\subseteq$ StUL (Buntrock et al., 1991)), which parallels the inclusions DLIN $\subseteq$ L and NLIN $\subseteq$ NL (Kasami, 1972; Ibarra et al., 1988).

Finally, it will prove useful to consider a generalization of strong unambiguity, where we bound the number of computation paths between the configurations.

Definition 7. (1) An automaton M is $a(n)$ *ambiguity bounded* if there are at most $a(n)$ computation paths between any two configurations of M for all inputs w with $|w| = n$.

(2) The class of languages recognized by $a(n)$ ambiguity bounded AuxPDAs in time $2^{O(\log^k n)}$ and space $\log n$ is denoted by Ambiguous-APDA$^k(a(n))$.

Clearly, by definition we have Ambiguous-$\mathrm{APDA}^k(1) = \mathrm{StUnambAPDA}^k$. Strong unambiguity of AuxPDAs seems to be more adequate for parallel complexity theory than weak unambiguity does. If we want to simulate a machine in some sense efficiently in parallel, it often comes out that it is necessary to have a restriction on the whole computation graph and not only for the parts belonging to accepting computations. This will become clearer when we consider simulations of AuxPDAs by circuits.

4. Simulations between semi-unbounded fan-in circuits and AuxPDAs

In this section we present a simulation of AuxPDAs by circuits where the number of accepting computations exactly transfers to the number of accepting subtrees. This improves the well-known simulation (Venkateswaran, 1991) of AuxPDAs by semi-unbounded fan-in circuits, which incorporates a number of accepting subtrees which exceeds the original number of accepting paths tremendously. This precise simulation makes it possible to prove the equality of the counting versions of SAC^1 and NAPDA^1. In particular, it puts us in the position to give characterizations of weakly and strongly unambiguous AuxPDAs in terms of the corresponding circuits. Furthermore, this simulation technique facilitates the application of a variation of the inductive counting technique (Immerman, 1988; Szelepcsényi, 1988), which yields a simulation of ambiguity bounded AuxPDAs by unambiguous ones within the same time (up to a polynomial) and space (up to a constant factor) bounds.

Subsequently, we precede as follows: At first, we prove three basic lemmata which serve as a basis for the construction of semi-unbounded fan-in circuits simulating AuxPDAs under preservation of the number of accepting computation paths. Afterwards, we give simulations of AuxPDAs by circuits and vice versa, thus proving several characterization results. Finally, we show $\mathrm{LOGUCFL} \subseteq \mathrm{UnambSAC}^1$, which improves the inclusion $\mathrm{LOGUCFL} \subseteq \mathrm{CREW}^1$ given by Rytter (1987) (due to Lange (1993) the latter inclusion can also be stated as $\mathrm{LOGUCFL} \subseteq \mathrm{UnambAC}^1$).

4.1. Computation paths of AuxPDAs — three basic lemmata. In the following we will deal intensively with computation paths. Therefore, it is necessary to introduce some more notation concerning AuxPDAs (Cook, 1971; Ruzzo, 1980).

In order to deal with computation paths, it will be useful to denote paths by their first and last (surface-)configuration and their length. A *(path) description* is a triple (A, B, i) consisting of two surface configurations A and B and an even natural number i. A description is called *realizable* if there exists a path from A to B in exactly i steps, where A and B have same push-down height and the level of the push-down does not go below this level during the computation. Note that because of the requirements of the preliminary section (i.e., AuxPDAs are required to push or to pop in each step), i can only be an even number. In general, (A, B, i) represents several paths of length i between A and B. To construct circuits simulating AuxPDAs, it is essential to split computation paths continuously into shorter and shorter paths until we end up with trivial paths, i.e., two-step transitions. The relation $\vdash$ shows how such a decomposition of paths is done. Let $x = (A, B, i)$, $y = (C, D, j)$, and $z = (E, B, k)$ be path descriptions. Then we write $y, z \vdash x$ and $z, y \vdash x$ iff

(1) The level of the push-down is equal for A, E, and B,
(2) there exists a computation from A to C in one step, pushing a symbol a onto the push-down store during this step,
(3) there exists a computation from D to E in one step, popping a from the push-down store, and
(4) $j + k = i - 2$.

In such a way we can reduce the checking of the realizability of x to the checking of the realizability of smaller paths y and z. In addition, it is important to remark that it is sufficient to utilize surface configurations in path descriptions due to the definition of $\vdash$. Finally, identical push-down heights of A, E, and B in the case of realizability also imply that C and D have same push-down height and, moreover, j and k are always even.

With the help of the decomposition relation $\vdash$ it is already possible to construct a simulating circuit. We only need to check whether one of the path descriptions (S_0, F_0, i) is realizable, where S_0 and F_0 denote the uniquely determined start, resp. final, configuration (with empty push-down store) and i is an even number bounded by the maximum running time of the simulated AuxPDA. Thus, we translate in a straightforward manner path descriptions (A, B, i) into gates $\langle A, B, i\rangle$, whose inputs are determined by the relation $\vdash$. (Observe that $(A, B, 0)$ is realizable iff $A = B$.) This approach fails because the depth of the resulting circuit would not be optimal at all, since we do not use a 'balanced' decomposition of computation paths.

However, we will demonstrate in the next three lemmata that a balanced and unique decomposition of computation paths is possible, thus guaranteeing an optimal depth for the simulating circuits as well as the preservation of the number of accepting computations. The first lemma states that for a *fixed* computation path (A, B, i) there exists an uniquely determined subpath (C, D, i_1) within (A, B, i), which essentially denotes the point which will serve to split (A, B, i) in a well-balanced way in different subpaths.

Lemma 8. *Let (A, B, i) denote a realizable path description for a fixed computation path of length $i \geq 2$ between A and B. Then there exist uniquely determined subpaths (C, D, i_1), (E, F, i_2), and (G, D, i_3) of (A, B, i) such that $(E, F, i_2), (G, D, i_3) \vdash (C, D, i_1)$ and $i_2, i_3 \leq i/2 < i_1$.*

Proof. The proof is based on a "recursive descent" where we make crucial use of the properties of the decomposition relation $\vdash$. Always observe that we speak of one *fixed* computation path between A and B.

If $i = 2$, then according to the definition of $\vdash$, there exists a uniquely determined surface configuration E such that $(E, E, 0), (B, B, 0) \vdash (A, B, 2)$. Thus, especially $(C, D, i_1) = (A, B, i)$ holds.

Now let $i > 2$. According to the definition of $\vdash$ there exist uniquely determined subpaths $(\tilde{E}, \tilde{F}, j_1)$ and $(\tilde{G}, B, j_2)$ such that $(\tilde{E}, \tilde{F}, j_1), (\tilde{G}, B, j_2) \vdash (A, B, i)$. We have to distinguish between two cases, one of them is trivial. We are done if j_1 and j_2 fulfill $j_1, j_2 \le i/2$. If j_1 and j_2 do not meet this condition, then *exactly* one of j_1 and j_2 must be greater than $i/2$. W.l.o.g. assume that $j_1 > i/2$. Now decompose $(\tilde{E}, \tilde{F}, j_1)$ according to $\vdash$ and check whether the lengths of the computation paths of the 'descendants' of $(\tilde{E}, \tilde{F}, j_1)$ are both less than or equal to $i/2$. This process continues until we come to the point where this condition is fulfilled. Obviously, this process terminates since the length of the considered computation paths is at least decreased by two (cf. definition of $\vdash$). In addition, it is also straightforward to see that we end up with uniquely determined (E, F, i_2), (G, D, i_3), and (C, D, i_1) satisfying the required conditions. $\square$

In Lemma 8 we could see that a fixed computation path can be split in three paths. The first two paths are the subpaths (E, F, i_2) and (G, D, i_3) and the third one is the path (A, B, i) with 'gap' (C, D, i_1). This means that the verification of the realizability of (A, B, i) can be reduced to showing that (E, F, i_2), (G, D, i_3), and the path with gap (C, D, i_1) are realizable. Before we get into details about this, let us first formalize the idea of paths with gap.

A (description for a) *path with gap* $(A, (C, D, j), B, i)$ consists of four surface configurations A, B, C, D and two even numbers i and j with $j \le i$. A path with gap $(A, (C, D, j), B, i)$ is called *realizable* iff A and B (resp. C and D) have same push-down heights and there exists a computation path from A to C and one from D to B with total number of steps $j - i$. Again the level of the push-down must not go below the level of A and B during the computation. In particular, $(A, (C, D, i), B, i)$ is realizable iff $(A, B) = (C, D)$.

Now we can generalize the decomposition relation $\vdash$ to computation paths with gap. Unfortunately, we have to distinguish between two cases, since now the gap may be in one of two subpaths. However, both are handled in full analogy to paths without gaps. Let $x = (A, (C, D, j), B, i)$ and, first, let $y = (E, (C, D, j), F, k)$ and $z = (G, B, l)$ or, second, let $y = (E, F, k)$, $z = (G, (C, D, j), B, l)$. Then we write $y, z \vdash x$ and $z, y \vdash x$ iff the level of the push-down is equal for A, G, and B, there exists one step from A to E pushing a symbol a onto the store, and there is one step from F to G popping a from the store and, finally, $k + l = i - 2$. In general, a gap (C, D, j) is interpreted as if the two surface configurations C and D simply were the same, i.e., as if the path from C to D would exist (without checking that). So $(A, (C, D, j), B, i)$ is interpreted as a path of length $j - i$ where C and D are regarded to be 'one' surface configuration. Lemma 9 is the analogue to Lemma 8, just stated for a fixed computation path *with* gap.

Lemma 9. *Let* $(A,(C,D,j),B,i)$, $i-j \geq 2$ *denote a realizable path with gap. Then there exist uniquely determined paths* $y=(E,(C,D,j),F,i_1)$ *and either*

(1) $z_1=(G,(C,D,j),H,i_2)$ *and* $z_2=(I,F,i_3)$, *such that* $z_1,z_2 \vdash y$ *and* $i_2-j \leq (i-j)/2 < i_1-j$ *or*

(2) $z_1=(G,H,i_2)$ *and* $z_2=(I,(C,D,j),F,i_3)$, *such that* $z_1,z_2 \vdash y$ *and* $i_3-j \leq (i-j)/2 < i_1-j$.

Proof. The proof is based on the same idea as the proof of Lemma 8. We just make use of the circumstance that the decomposition relation $\vdash$ uniquely determines both subpaths of a given path (with gap). The only difference compared to Lemma 8 is that now always the paths with the gap are chosen, until we find the y such that the second condition (concerning the length of paths) is true. Furthermore, we use the fact that the second condition uniquely determines the subpaths (with gap) y, z_1, and z_2. □

Lemma 9 will be used to decompose computation paths with gaps in a balanced way. In order to investigate the realizability of $(A,(C,D,j),B,i)$ we confine ourselves to examine the realizability of $(A,(E,F,i_1),B,i)$, z_1, and z_2. Here, both possible subpaths with gap have length less than or equal to half of the length of the whole path with gap $(A,(C,D,j),B,i)$. The arising subpath without gap may have a maximum length of $i-j-2$ and will be split with the help of Lemma 8 in a well-balanced way.

Up to now we only considered *one fixed* computation path (with gap). But in general there are several computation paths guaranteeing the realizability of (A,B,i). In other words, this means that (A,B,i) usually represents several paths. Our aim in the next lemma is to show that the decompositions of Lemma 8 and Lemma 9 preserve the number of paths, that is, for example, the number of paths of length i between A and B can be computed from the number of paths of the decomposition components $(A,(C,D,j),B,i)$, (E,F,i_1), and (G,D,i_2) of (A,B,i) (cf. Lemma 8). Let $\#(A,B,i)$ (resp. $\#(A,(C,D,j),B,i))$ denote the number of paths between A and B of length i (resp. the number of paths between A and B with gap (C,D,j) of length $i-j$). We get the following statement for the decompositions of Lemma 8 (resp. Lemma 9).

Lemma 10.

(1)
$$\#(A,B,i) \;=\; \sum \#(A,(C,D,j),B,i)\cdot\#(E,F,i_1)\cdot\#(G,D,i_2),$$
where the sum is taken over all combinations of surface configurations C, D, E, F, G *and even numbers* j, i_1, *and* i_2 *such that* $(E,F,i_1),(G,D,i_2) \vdash (C,D,j)$ *and* $i_1,i_2 \leq i/2 < j$.

(2)

$$\#(A,(C,D,j)B,i) =$$
$$\sum \#(A,(C_1,D_1,j_1),B,i)\cdot\#(E,(C,D,j),F,i_1)\cdot\#(G,D_1,i_2) +$$
$$\sum \#(A,(C_1,D_1,j_1),B,i)\cdot\#(E,F,i_1)\cdot\#(G,(C,D,j),D_1,i_2),$$

where both sums are taken over all combinations of surface configurations C_1, D_1, E, F, G and even numbers i_1, i_2, and j_1 such that $(E,(C,D,j),F,i_1),(G,D_1,i_2) \vdash (C_1,(C,D,j),D_1,j_1)$ and $i_1 - j \le (i-j)/2 < j_1 - j$ for the first of the two sums and $(E,F,i_1),(G,(C,D,j),D_1,i_2) \vdash (C_1,(C,D,j),D_1,j_1)$ and $i_2 - j \le (i-j)/2 < j_1 - j$ for the second one.

Proof.

(1) As already mentioned before, Lemma 8 provides a unique decomposition of a fixed computation path of length i between A and B in three uniquely determined sub-paths, where one of them has gap (C,D,j) and the other two (E,F,i_1) and (G,D,i_2) 'fill the gap'. Therefore, if we consider the number of all computation paths represented by (A,B,i), we have to look at the number of all paths represented by the three path descriptions $(A,(C,D,j),B,i)$, (E,F,i_1), and (G,D,i_2) obtained from Lemma 8. Observe that different paths represented by (A,B,i) nevertheless may result in same decomposition components, since they can differ inside the subpaths. So a product $\#(A,(C,D,j),B,i)\cdot\#(E,F,i_1)\cdot\#(G,D,i_2)$ stands for the number of paths of length i between A and B, which (in Lemma 8) all yield the same surface configurations C, D, E, F, G and numbers i_1, i_2, and j. Finally, the sum has to be taken over all possibilities of how a path represented by (A,B,i) may be decomposed in *different* components.

(2) A similar argument together with Lemma 9 yields the corresponding claim for path representations with gap.

□

4.2. Exact simulations of AuxPDAs by circuits. In the preceding subsection computation paths were decomposed in a unique and well-balanced way. In this section the main idea is to translate path descriptions into gates that compute the realizability of the respective path descriptions. It will turn out that this method results in semi-unbounded fan-in circuits having a number of accepting subtrees which is the same as the number of accepting computations of the simulated AuxPDAs.

One fundamental result of this work concerns the counting versions of NAPDA^1 and SAC^1. Remember that $\#\mathrm{APDA}^1$ (resp. $\#\mathrm{SAC}^1$) are the classes of functions which map an input word to the number of accepting computations (resp. accepting subtrees) of an (nondeterministic) AuxPDA (resp. semi-unbounded fan-in circuit).

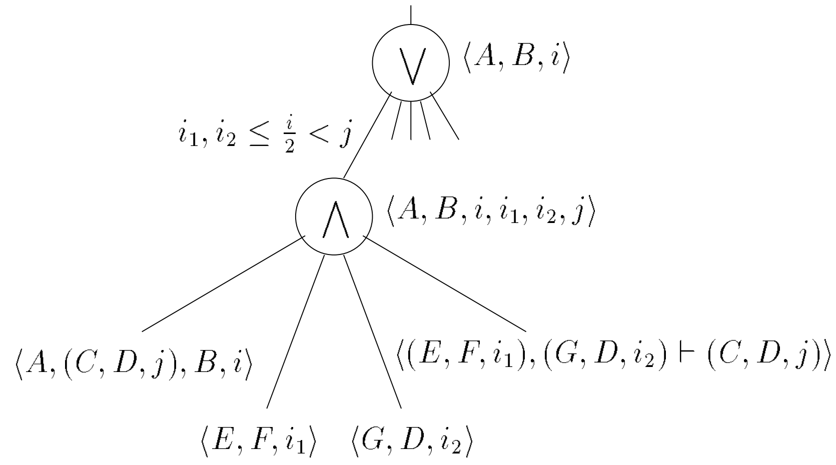


FIGURE 1. Sub-circuit computing the realizability of (A, B, i).

The following lemma provides the first step in the proof of the announced equality of these counting classes. The reverse direction will be given in the next subsection.

Lemma 11. $\#\mathrm{APDA}^1 \subseteq \#\mathrm{SAC}^1$.

Proof. Based on Lemma 10 we construct an SAC^1-circuit C simulating an AuxPDA M with polynomial running time and logarithmic working space. Due to the logarithmic space bound there only exist polynomially many surface configurations of M.

The circuit mainly consists of gates denoted by $\langle A, B, i\rangle$ and $\langle A, (C, D, j), B, i\rangle$ that compute the realizability of the corresponding path descriptions. Remember that M accepts, iff there exists an i (bounded by the running time of M) such that (S_0, F_0, i) is realizable, where S_0 and F_0 denote the uniquely determined initial and end configuration. Consequently, the output gate of C will be an unbounded OR-gate with inputs $\langle S_0, F_0, i\rangle$, where i runs through all even numbers bounded by the running time of M.

It remains to be shown how gates named $\langle A, B, i\rangle$ or $\langle A, (C, D, j), B, i\rangle$ are constructed. At this point, Lemma 10 comes into play. One simply has to translate the sum symbols in Lemma 10 into (unbounded) OR-gates and the multiplication symbols into (bounded) AND-gates. For example,

$$\#(A, B, i) = \sum_{i_1, i_2 \leq i/2 < j} \#(A, (C, D, j), B, i) \cdot \#(E, F, i_1) \cdot \#(G, D, i_2)$$

results in the (sub)circuit of Figure 1. Gates $\langle A, B, i\rangle$ and $\langle A, B, i, i_1, i_2, j\rangle$ are connected iff $i_1, i_2 \leq \frac{i}{2} \leq j$.

The rightmost input of the AND-gate in Figure 1 computes whether the first condition of the first case in Lemma 10 holds. Obviously this can be checked by a circuit of constant size because it only depends on the transition relation of M and constant many bits of the input word. The second condition in Lemma 10, i.e., $i_1, i_2 \leq i/2 < j$ serves to restrict the inputs of the OR-gate to all those numbers which are suitable candidates for a *well-balanced* decomposition of (A, B, i). The translation of the equations for $(A, (C, D, j), B, i)$ of Lemma 10 occurs in full analogy to the above. Clearly, gates like $\langle A, B, 0\rangle$ (resp. $\langle A, (C, D, i), B, i\rangle$) have constant value 1, iff $A = B$ (resp. $(A, B) = (C, D)$) holds and value 0, otherwise.

Since the number of accepting subtrees of an OR-gate is the *sum* over the number of accepting subtrees of its inputs and the number of the accepting subtrees of an AND-gate is the *product* over the number of accepting subtrees of its inputs, due to Lemma 10 we immediately have the following: Each gate $\langle A, B, i\rangle$ of the constructed circuit has as many accepting subtrees as paths of length i between A and B exist. In particular, the above described output gate exactly has as many accepting subtrees as M has accepting computations.

The polynomial size of C directly follows from the polynomial number of surface configurations and the polynomial running time of M. The logarithmic depth is due to the well-balanced decomposition of computation paths provided by Lemma 10 (resp. Lemma 8 and 9) and logspace uniformity is a straightforward consequence of the simplicity of the construction. □

Next, a simulation of weakly and strongly unambiguous AuxPDAs by their circuit counterparts is given. Both these inclusions are consequences of Lemma 11 and mean the first step towards proving the equality of these classes.

Lemma 12. (1) $\mathrm{UnambAPDA}^1 \subseteq \mathrm{WeakUnambSAC}^1$,
(2) $\mathrm{StUnambAPDA}^1 \subseteq \mathrm{UnambSAC}^1$.

Proof.

(1) In Definition 3 of $\mathrm{WeakUnambSAC}^1$-circuits it was explained that they are SAC^1-circuits with at most one accepting subtree. A (weakly) unambiguous AuxPDA has at most one accepting computation and so the claim follows from Lemma 11.

(2) Again we use the construction of Lemma 11. We just have additionally to show that *all* OR-gates of the simulating circuit are vulnerable, i.e., that all OR-gates of the circuit have at most *one* accepting subtree. The strong unambiguity of the simulated AuxPDA M means that there is at most one path between two arbitrary configurations of M. This also implies that there is at most one path with (fixed) gap between two configurations. Thus the realizability of all gates $\langle A, B, i\rangle$ and $\langle A, (C, D, j), B, i\rangle$ can be verified in at most one way, that is, the corresponding gates have at most one accepting subtree. The claim follows because these are the only OR-gates appearing in the construction (except for the output gate).

□

The second part of Lemma 12 can be generalized to ambiguity bounded AuxPDAs (Definition 7). Here one demands that the number of paths between *arbitrary* configurations is bounded by a value $a(n)$. Analogously, ambiguity bounded, semi-unbounded fan-in circuits were defined by restricting the number of accepting subtrees for *all* gates of the circuit (Definition 4). The following proposition is a generalization of part two of Lemma 12.

Proposition 13. *Ambiguous-APDA*$^1(a(n)) \subseteq$ *Ambiguous-SAC*$^1(a(n)^2)$.

Proof. The proof is similar to the proof of Lemma 12. The gates $\langle A, (C, D, j), B, i\rangle$ checking the realizability of paths of length $j - i$ between A and B with a gap between C and D can have at most $a(n)^2$ accepting subtrees. This is due to the fact that those gates actually check the existence of two paths in each case, that is the existence of a path from A to C and one from D to B.

According to the ambiguity bound of the simulated AuxPDA there may be at most $a(n)$ paths between A and C and between D and B each time, yielding a maximum of $a(n)^2$ paths represented by $(A, (C, D, j), B, i)$. Clearly, the number of paths represented by path descriptions (A, B, i) is bounded by $a(n)$. Together with arguments used in the second part of the proof of Lemma 12 this results in an upper bound of $a(n)^2$ for the ambiguity of all gates of the simulating circuit. □

Evidently, the claim of Proposition 13 still remains true if one takes the counting versions like in Lemma 11. That is, we also have #Ambiguous-APDA$^1(a(n)) \subseteq$ #Ambiguous-SAC$^1(a(n)^2)$.

4.3. Exact simulations of circuits by AuxPDAs. In contrast to the simulation of AuxPDAs by circuits, the simulation of circuits by AuxPDAs can be obtained by slight modifications of techniques developed by Venkateswaran (1991). These simulations will enable us to prove characterizations of unambiguous and counting versions of AuxPDAs by semi-unbounded fan-in circuits. Note that in the subsequent lemma (in contrast to Lemma 11 and 12) we have simulation results for arbitrary natural powers $k \geq 1$.

Lemma 14. (1) #SAC$^k \subseteq$ #APDA^k,
(2) WeakUnambSAC$^k \subseteq$ UnambAPDA^k,
(3) UnambSAC$^k \subseteq$ StUnambAPDA^k.

Proof. The first two cases can be proved directly by the simulation method of Venkateswaran (1991). In order to prove the more intricate third case it is necessary to make an addition to this technique. In this case the push-down store is utilized to guarantee the strong unambiguity of the simulating AuxPDA.

In the sequel, we give a concise presentation of Venkateswaran's method. Furthermore, in curly brackets we state the additions required for the proof of the third part. Let C be the given, logspace uniform, semi-unbounded fan-in circuit and let M be the simulating AuxPDA. The simulation starts at the output gate of C. For an arbitrary gate g of C, M does the following:

- If g is an OR-gate, { M pushes g marked as 'done' on the push-down, } M guesses one predecessor h of g and checks recursively whether h has value 1.
- If g is an AND-gate, M computes its (constant many) predecessors in a fixed (e.g. lexicographical) order, pushes all of them except the last one on the push-down and recursively verifies that the last gate has value 1.
- If g is an input gate of C, then M rejects, if g has value 0. If g has value 1, M accepts, if the push-down is empty and, otherwise, M pops the topmost push-down element and works on it.
- { If g is a gate marked 'done', then M pops the topmost gate from the push-down and recursively works on it (i.e., g is ignored). }

The correctness of the above described simulation is proved as follows:

(1),(2) For the first two cases it suffices to show that $\#\mathrm{SAC}^k \subseteq \#\mathrm{APDA}^k$ holds, since $\mathrm{WeakUnambSAC}^k \subseteq \mathrm{UnambAPDA}^k$ is the special case where we only have one accepting computation. The inclusion $\#\mathrm{SAC}^k \subseteq \#\mathrm{APDA}^k$ is easily derived from the fact that the above described simulation just guesses the accepting subtrees of the simulated circuit. Thus the simulating AuxPDA has exactly as many accepting computations as accepting subtrees of the circuit exist. The time and space bounds are straightforward from the logspace uniformity, the polynomial size and the polylogarithmic depth of the circuit (see (Venkateswaran, 1991)).

(3) To prove $\mathrm{UnambSAC}^k \subseteq \mathrm{StUnambAPDA}^k$ we need the additions in curly brackets. With them it is possible to show the existence of at most one computation path between two arbitrary configurations of the simulating AuxPDA M. Observe that it is crucial here that the total contents of the push-down store is part of a configuration of M. Due to the additional 'done'-gates we have a partial protocol of the guessed subtree on the store. The strong unambiguity of M is proved by contradiction.

Suppose that M is not strongly unambiguous. Then there must exist a configuration K of M with two immediate successors K_1 and K_2 ($K_1 \neq K_2$) such that K_1 and K_2 themselves have a common successor K_c. It is important that due to the definition of M, configurations K_1 and K_2 must differ with respect to their push-down contents. This results from the fact that the only place in the described simulation where nondeterminism comes into play is the point where M guesses *one* input of an OR-gate. But this guessed input is pushed on the store and consequently K_1 and K_2 must differ in such a guessed gate. So we can assume that K_1 and K_2 differ in their topmost push-down symbols, that is, in two different input gates of an OR-gate. Let us call these two gates g_1 and g_2. If K_1 and K_2 now have a common successor K_c, this in particular means that g_1 and g_2 have to be popped from the push-down before M reaches K_c. Pursuant to the definition of M, gates marked 'done' are popped from the push-down only if they are the roots of accepting subtrees (i.e., g_1 and g_2 evaluate to 1). But this is a contradiction to the precondition that all gates of the simulated circuit have at most one accepting subtree.

□

Most of the results up to now can be summarized in the following theorem. Recently, Lenz (1992) and Vinay (1991) independently proved the first case by quite different methods.

Theorem 15. (1) $\#\mathrm{SAC}^1 = \#\mathrm{APDA}^1$,
(2) $\mathrm{WeakUnambSAC}^1 = \mathrm{UnambAPDA}^1$,
(3) $\mathrm{UnambSAC}^1 = \mathrm{StUnambAPDA}^1$.

Proof. Simply combine Lemma 11 (resp. Lemma 12) and Lemma 14. □

4.4. The recognition of unambiguous CFLs. Finally, we prove the inclusion of the closure of unambiguous context-free languages under log-space many-one reductions in $\mathrm{UnambSAC}^1$. This result has some important consequences. First, it extends Rytter's inclusion $\mathrm{LOGUCFL} \subseteq \mathrm{CREW}^1$ (Rytter, 1987). Note that Lange (1993) characterized CREW^1 by $\mathrm{UnambAC}^1$, a presumably stronger circuit class than $\mathrm{UnambSAC}^1$. In Rytter's algorithm "non-monotone" write accesses were a crucial part of the algorithm. Theorem 15 immediately implies a *monotone* CREW algorithm, since $\mathrm{UnambSAC}^1$-circuits can be simulated by CREW-PRAMs in a monotone way. Second, Theorem 15 also helps to shed some more light on the question whether $\mathrm{LOGUCFL} = \mathrm{UnambAPDA}^1$. As can be seen in Section 5, $\mathrm{UnambSAC}^1$ and its complement are included in $\mathrm{UnambAPDA}^1$. The above equality would imply the closure under complementation of LOGUCFL. Since the latter seems to be unlikely, we conjecture the strict inclusion of LOGUCFL in $\mathrm{UnambAPDA}^1$.

Theorem 16. $\mathrm{LOGUCFL} \subseteq \mathrm{UnambSAC}^1$.

Proof. Due to Theorem 15 it is sufficient to prove $\mathrm{LOGUCFL} \subseteq \mathrm{StUnambAPDA}^1$. Moreover, because $\mathrm{StUnambAPDA}^1$ clearly is closed under (deterministic) logspace many-one reductions, it suffices to show $\mathrm{UCFL} \subseteq \mathrm{StUnambAPDA}^1$.

Let $G = (N, T, P, S)$ be an unambiguous context-free grammar in Chomsky normal form, that is, there are only productions of the form $A \to BC$ or $A \to a$, where $A, B, C \in N$ (set of nonterminals) and $a \in T$ (set of terminals). W.l.o.g. assume every nonterminal to be both reachable and productive. Let $w \in T^*$ be an arbitrary, but fixed input of length n, i.e., $w = a_1 a_2 \ldots a_n$ for $a_i \in T$, $1 \le i \le n$. For $0 \le i \le j \le n$ we set ${}_i w_j := a_{i+1} \ldots a_j$. Thus we have ${}_i w_i = \varepsilon$ and ${}_0 w_n = w$.

The AuxPDA M accepting $L(G)$ works as follows. M starts with the triple $\langle 1, n, S\rangle$ on the working tape. For an arbitrary triple $\langle i, j, A\rangle$ with $i \leq j, A \in N$, M guesses a production $A \to \alpha$, $\alpha \in N^2 \cup T$ and then proceeds as follows:

- If $\alpha = BC$, then M pushes $\langle A \to BC\rangle$ on the store and guesses a number k with $i \leq k \leq j$, pushes $\langle k, j, C\rangle$ on the store and recursively works on the triple $\langle i, k, B\rangle$.
- If $\alpha = a$, then M rejects if ${}_iw_j \neq a$ (especially $j = i + 1$ must hold), accepts if the push-down is empty and recursively checks the next topmost push-down contents, otherwise.
- If M pops a push-down contents of shape $\langle D \to EF\rangle$, then M simply checks the rest of the push-down store (i.e., "ignores" $\langle D \to EF\rangle$).

The push-down contents representing productions with right-hand side consisting of nonterminals ($\langle D \to EF\rangle$) serves for guaranteeing the strong unambiguity of M. As in Lemma 14 it is easily seen that there is only one place where nondeterminism occurs. Here it is the point where we guess a production with a given nonterminal on the left side.

Suppose that there is a configuration K of M with two immediate, different successors K_1 and K_2, such that K_1 and K_2 have a common successor K_c. This leads to a contradiction to the unambiguity of the given grammar G. With arguments analogous to Lemma 14 it is easy to see that K_1 and K_2 must differ in the topmost push-down symbol. There are two cases to distinguish. Either the topmost push-down contents represents (two different) guessed productions $A \to B_1C_1$ (resp. $A \to B_2C_2$) or we have for some production $A \to BC$ two different guesses k_1 and k_2 for the value of k, thus yielding $\langle i, k_1, B\rangle$ and $\langle k_1, j, C\rangle$ or $\langle i, k_2, B\rangle$ and $\langle k_2, j, C\rangle$. Since the handling of the second case is similar to the first one, we only describe the first one. In the first case, clearly, both productions must be popped from the push-down before M reaches K_c. According to the definition of M, these productions are only popped if $A \overset{*}{\Rightarrow} {}_iw_j$. Because each nonterminal of G is *productive* and *reachable*, there must exist $v_1, v_2 \in T^*$ such that $v_1{}_iw_jv_2$ is derived as $S \overset{*}{\Rightarrow} v_1Av_2 \Rightarrow v_1B_1C_1v_2 \overset{*}{\Rightarrow} v_1{}_iw_jv_2$ on the one hand and, on the other hand, derived as $S \overset{*}{\Rightarrow} v_1Av_2 \Rightarrow v_1B_2C_2v_2 \overset{*}{\Rightarrow} v_1{}_iw_jv_2$. This means that we have two different derivation trees for $v_1{}_iw_jv_2$, thus contradicting the unambiguity of G. $\square$

5. Inductive counting on semi-unbounded fan-in circuits

The inductive counting technique of Immerman (1988) and Szelepcsényi (1988) led to one of the most outstanding results in structural complexity theory of the last years: Nondeterministic space is closed under complementation. Soon this method was employed to prove several important results (e.g. (Borodin et al., 1989; Buntrock et al., 1993)). Here we will open a further field of application for this technique by translating the methods of Buntrock et al. (1993) into leveled, semi-unbounded circuits: inductive counting on leveled, semi-unbounded fan-in circuits. Observe that for most of the circuit classes it is no restriction to demand them to be leveled (see Sections 2 and 3). This new variation of inductive counting can always be applied when we have characterizations of AuxPDAs by circuits. We obtain several results: As a first example, we give a new proof for the closure under complementation of SAC^k (Borodin et al., 1989). Second, this method enables us to improve the main result of Buntrock et al. (1993). Polynomially ambiguity bounded AuxPDAs can be simulated by unambiguous ones without time or space penalty. Third, inductive counting applies for proving the inclusion $\mathrm{UnambSAC}^k \subseteq \mathrm{UnambAPDA}^k$, an open question considered to be at least difficult by Lange (1993). Finally, careful analysis of a proof of Borodin et al. (1989) reveals the closure under complementation of the unambiguous circuit class $\mathrm{UnambSAC}^k$.

5.1. The basic inductive counting lemma and a first application. Before presenting the inductive counting method for leveled circuits, we have to point out *what* we will count in circuits. Therefore, the notion of unit of measurement (measure for short) for gates is introduced. A *measure value* m (m-value for short) of a gate g in circuit C with fixed input $w \in \{0,1\}^n$ is a natural number which only depends on the type of g and the measure values of the inputs of g. Herein, input gates of C will always have value 0 or 1 (corresponding to their Boolean value). In addition, the (measure) value of g must be computable in an easy way (preferably with logarithmic space) by an associative and commutative operation (essentially, we only use addition or multiplication). For example, the simple Boolean value of gates on given inputs can be interpreted as a measure. The central lemma of this subsection, which presents the technique of inductive counting on leveled circuits, can now be stated.

Lemma 17. *Let C be a logspace uniform, leveled circuit of size $z(n) \geq n$ with fixed input word $w \in \{0,1\}^n$. Furthermore, let m be a measure such that for all gates of C the m-value is bounded by $a(n)$ and suppose that there exists an AuxPDA algorithm verifying for an arbitrary gate g of C in time $t(n)$ and space $s(n)$ that g has an m-value* at least *as big as a given number.*

Then the m-value of the output gate of C can be ascertained by a nondeterministic AuxPDA with time $z(n)^{O(1)} + (\log a(n) + t(n))z(n)^2$ and space $\max(\log a(n) + \log z(n), s(n))$.

Proof. The key idea of the proof is as follows: Starting with the level of the input gates, the simulating nondeterministic AuxPDA M finds out level by level for all gates of a considered depth their value according to the measure m. In this way, the m-value of the output gate of C will finally be ascertained.

The details are as follows. Let S_i denote the sum over the m-values of all gates at level i (i.e., depth i). Obviously, S_0 is known (resp. easy to compute) because in level 0 we only have the input gates of C (with fixed Boolean values). Now let $i > 0$ and suppose that S_{i-1} is known. In order to ascertain S_i, M has to find out all m-values of gates at level i. To determine the m-value of *one* gate at level i, M procedes in the following way: For each gate g_{i-1} in depth $i-1$, M guesses its m-value v, verifies with the (according to the precondition) given algorithm that g_{i-1} has an m-value at least as big as v, and increases a counter variable Z (which is initialized with zero) by v. If g_{i-1} is an input to g_i (recall that, all inputs of g_i lie at level $i-1$), the m-value of g_{i-1} serves in a straightforward manner to compute the m-value of g_i. Here we make use of the fact that due to the required associativity and commutativity of the operation for computing m-values, M can always do a partial computation for the m-value of g_i in one variable. When M has gone through all gates in level $i-1$, it compares the counter variable Z with the already known sum S_{i-1}. If Z is less than S_{i-1}, then there must exist a gate at level $i-1$ for which M has guessed a too small m-value and consequently M rejects. If Z is equal to S_{i-1} (Z greater than S_{i-1} is impossible), M must have guessed the m-values for all gates in level $i-1$ in the right way and, therefore, the m-value of g_i has been computed correctly. Repeating this procedure for all gates at level i, M finds out the value of the sum S_i. In particular, M finally gets the m-value of the output gate of C.

Because of the assumptions made and especially the logspace uniformity of C (which inter alia is extensively used above to find out all gates at a certain level), the claimed time and space bounds for M can be verified easily. We need time $z(n)^{O(1)} + (\log a(n) + t(n))z(n)^2$ due to the requirements of the uniformity machine, the necessity of summing up ambiguity values, and the verification of guessed ambiguity values, respectively. The space bound $\max(\log a(n) + \log z(n), s(n))$ derives from analogous considerations and the proof is completed. $\square$

A first simple application of Lemma 17 provides a new proof for the closure under complementation of SAC^k (Borodin et al., 1989), which in particular implies the closure under complementation of LOGCFL (Sudborough, 1978; Venkateswaran, 1991).

Corollary 18. (Borodin et al., 1989) $\mathrm{SAC}^k = \text{Co-SAC}^k$.

Proof. W.l.o.g. let the given SAC^k-circuit C be leveled. We just use the trivial measure described in the beginning, that is the Boolean values of the gates. The verification algorithm needed for Lemma 17 is obtained from Venkateswaran's equality $\mathrm{SAC}^k = \mathrm{NAPDA}^k$ (Venkateswaran, 1991). An application of Lemma 17 then yields the desired result, if we define the simulating AuxPDA to accept iff the output gate of C has value 0. Note that in this way we have proved $\text{Co-SAC}^k \subseteq \mathrm{NAPDA}^k$ and $\mathrm{NAPDA}^k = \mathrm{SAC}^k$ provides the claim. $\square$

5.2. Simulating ambiguity bounded AuxPDAs by unambiguous ones. In the proof of Lemma 17 it can be observed that the simulating AuxPDA does not reject, only if the exact measure values (m-values) of all gates of the given circuit are guessed correctly. Since there is only one possibility to make only correct guesses, this implies that, if one has an algorithm for AuxPDAs verifying *correctly guessed* m-values in an *unambiguous* way, the total simulation of Lemma 17 will be unambiguous. In order to get an unambiguous, verifying AuxPDA, it is necessary to find a gate measure which allows a verification of correctly guessed m-values in exactly one way. Subsequently, for two purposes we will present adequate measures which enable unambiguous verifications. In this way, eventually unambiguous simulations of ambiguity bounded circuits (and, thus due to Proposition 13, of ambiguity bounded AuxPDAs) and UnambSAC^k-circuits are obtained.

Let us begin by restating the notion of ambiguity bounded gates. In Section 3 we said that a gate is $a(n)$ ambiguity bounded if it has at most $a(n)$ accepting subtrees. Furthermore, in Section 4 we already mentioned that the ambiguity value of an AND-gate can be computed by multiplying the ambiguity values of its inputs and the ambiguity value of an OR-gate is the sum over the ambiguity values of its inputs. Clearly, the ambiguity value of an input gate simply corresponds to its Boolean value. Having these facts in mind, we are able to prove one of the main results of this paper: Polynomial ambiguity bounded AuxPDAs can be simulated by unambiguous ones without time and space penalty.

Theorem 19. (1) Ambiguous-SAC$^1(n^{O(1)}) \subseteq$ UnambAPDA^1,
(2) Ambiguous-APDA$^1(n^{O(1)}) \subseteq$ UnambAPDA^1.

Proof.

(1) According to the above explanations it suffices to show that for each gate g of the given circuit C a correctly guessed ambiguity value for g can be verified unambiguously. If this can be done by an AuxPDA in polynomial time and logarithmic space, the claim follows with the help of Lemma 17, since w.l.o.g. C can be assumed as leveled. The AuxPDA M verifies a guessed ambiguity value a for a gate g in the following manner:
   - If g is an OR-gate, M ascertains in some fixed (e.g. lexicographical) order all input gates of g and guesses for all of them a corresponding ambiguity value such that the *sum* of these equals a. Then M recursively verifies the ambiguity values of the input gates different from zero.
   - If g is an AND-gate, M ascertains in some fixed order the (constant many) input gates of g and guesses for all of them a corresponding ambiguity value such that the *product* of these equals a. Then M recursively verifies the nonzero ambiguity values of the input gates.
   - If g is an input gate, M rejects, if $g = 0$ and $a \geq 1$, or if $g = 1$ and $a > 1$. If the push-down store is empty and $g = 1$ and $a = 1$, then M accepts. Otherwise M recursively works on the topmost push-down contents.

   The logarithmic space bound is straightforward and with little effort a polynomial running time can be proved for M. Furthermore, the unambiguity of M for correctly guessed ambiguity values a is a direct consequence of the fact that in this case M always has exactly one possibility to guess correctly the ambiguity values of the input gates each time. Note that if a is guessed too small, M has more than one possibility to do the verification and, because of that, no longer works unambiguously. But recall that in this case the AuxPDA of Lemma 17 will finally reject, since then $Z = S_{i-1}$ does not hold. Using the above described unambiguous AuxPDA M for the verification algorithm, application of Lemma 17 now provides the desired result: The AuxPDA simulating the ambiguity bounded circuit C ascertains the ambiguity value of the output gate of C and accepts, iff it is greater than 0.

(2) Proposition 13 yields Ambiguous-APDA$^1(n^{O(1)}) \subseteq$ Ambiguous-SAC$^1(n^{O(1)})$ and thus the second part of Theorem 19 follows by application of part one.

□

Note that the inclusion

$$\text{NSPACE-AMBIGUITY}(\log n, n^{O(1)}) \subseteq \text{UnambAPDA}^1 \qquad (*)$$

was in particular proved in (Buntrock et al., 1993), restricting the unambiguity only between reachable configurations. Theorem 19 is one possible improvement over (*). Buntrock, Jenner, Lange, and Rossmanith (1991) improved the upper bound of (*): NSPACE-AMBIGUITY$(\log n, n^{O(1)}) \subseteq$ DAPDA^1.

The simulating AuxPDA of Theorem 19 computes the number of accepting subtrees of the output gate of the simulated circuit. This means that it can compute the function mapping input words to the number of accepting subtrees (resp. accepting computations (Proposition 13)). So we even have the following result:

Corollary 20. *(1) #Ambiguous-SAC$^1(n^{O(1)}) \subseteq$ FUnambAPDA^1,*
(2) #Ambiguous-APDA$^1(n^{O(1)}) \subseteq$ FUnambAPDA^1.

One application of Theorem 19 is that CFLs generated by grammars which possess at most a polynomial number of derivation trees for arbitrary words can be recognized by unambiguous AuxPDAs.

Corollary 21. *The word problem of polynomially ambiguous CFLs is contained in* UnambAPDA^1.

Proof. In Theorem 16 the inclusion UCFL $\subseteq$ UnambSAC^1 (respectively UCFL $\subseteq$ StUnambAPDA^1 (and therefore, LOGUCFL $\subseteq$ StUnambAPDA^1)) was proven. It can be observed that this construction also serves for showing the inclusion of polynomially ambiguous CFLs in Ambiguous-APDA$(n^{O(1)})$. Assuming the correctness of the latter, the claim follows by application of Proposition 13 and Theorem 19. Let us shortly indicate the correctness of the stated inclusion. For the recognition of polynomially ambiguous CFLs we use exactly the same AuxPDA M as we did for the recognition of unambiguous CFLs in Theorem 16. To show that the ambiguity of M now remains polynomially bounded is again done by contradiction:

Suppose that there are more than $p(n)$ paths between two configurations K_1 and K_2 of M, where p is a polynomial that bounds the number of derivation trees. In a way similar to Theorem 16 it follows that for all paths from K_1 to K_2 there must be *one* terminal string v, which is recognized during the transitions from K_1 to K_2. Making again use of the assumption that all nonterminals of the underlying grammar are both productive and reachable, it can be concluded in a way analogous to Theorem 16 that there must exist more than $p(n)$ derivation trees for some terminal string generated by the grammar. This contradicts the assumption and the claim follows. $\square$

From Theorem 16 and (Lange, 1993) we have that unambiguous CFLs can be recognized by CREW-PRAMs with a polynomial number of processors in logarithmic time. This result was already given by Rytter (1987), showing that more precisely n^7 processors suffice. Meanwhile, Rossmanith and Rytter (1992) reduced the number of processors to n^6 and proved that even finitely ambiguous CFLs can be recognized by this PRAM-class.

If one improved Corollary 21 by showing that polynomial ambiguous CFLs are included in StUnambAPDA^1, then together with Theorem 15 it would follow that even polynomially ambiguous CFLs can be recognized by CREW-PRAMs in logarithmic time with a polynomial number of processors. At least Corollary 21 implies that polynomially ambiguous CFLs can be recognized by *robust* PRAMs (Hagerup and Radzig, 1990) within the same complexity bounds. This is a straightforward consequence of the characterization of UnambAPDA^1 by WeakUnambSAC^1-circuits given in Theorem 15. One just has to simulate these weakly unambiguous circuits by the robust PRAM in the usual way (see, for example, (Lange, 1993)).

5.3. Further applications and results. A second application of Lemma 17 allows us to show UnambS^{E}AC^k $\subseteq$ UnambAPDA^k for $k \geq 1$. In this way we get the first nontrivial upper bound for UnambS^{E}AC-circuits demanded in (Lange, 1993). Lemma 17 itself will serve to prove UnambS^{E}AC^1 $\subseteq$ UnambAPDA^1. Because UnambS^{E}AC^1-circuits can have a super-polynomial ambiguity, it is necessary to introduce a new measure for gates in order to obey a logarithmic space bound (cf. Lemma 17). For this we make use of the so-called *saturation* of gates.

A gate is considered as saturated, if it is an input gate with value 1 or if it is an OR-gate with at least two inputs with value 1. The following definition generalizes this concept by making the saturation of a gate dependent on the saturation of its inputs. Let C be an UnambS^{E}AC-circuit and g be any gate in C. Then the saturation of g is defined as follows.

- If g is an input gate of C, then it has saturation 1, if $g = 1$ and saturation 0, otherwise.

- If $g = \mathrm{AND}(g_1, g_2)$, then its saturation is the sum of the saturations of g_1 and g_2, if both these saturations are greater than 0 and the saturation of g is 0, otherwise.
- If g is an OR-gate (of arbitrary fan-in), then we have to consider two cases. If g evaluates to 0, then its saturation is defined to be 0. If g has exactly one input with saturation greater than 0 (i.e., g has at most one input which evaluates to 1), then the saturation of g is defined as the saturation of the gate at this input. Otherwise, if g has two 1-inputs, then g is a bounded OR-gate and the saturation of g is the sum over the saturations of all the inputs of g plus 1 (because g itself is a saturated OR-gate).

Note that the output gate of circuit C on input x has saturation greater than 0, iff C accepts x. Obviously, the saturation is polynomially bounded for all the gates of UnambSAC^1-circuits. The essential request behind this definition again is to be able to supply an AuxPDA verifying correctly guessed saturation values unambiguously.

Lemma 22. UnambSAC^1 $\subseteq$ UnambAPDA^1.

Proof. W.l.o.g. we assume the given UnambSAC^1-circuit C to be leveled. It suffices for applying Lemma 17 to show that a correctly guessed saturation value of a gate can be verified in polynomial time and within logarithmic space by an unambiguous AuxPDA, because the saturation measure is polynomially bounded for all gates of UnambSAC^1-circuits. The verification algorithm exactly follows the definition of saturation and the proof is done analogously to Theorem 19, where the verification algorithm followed the definition of ambiguity for gates. Clearly, the (unambiguous) AuxPDA resulting from Lemma 17 accepts iff the output gate of C has saturation greater than 0. $\square$

Lemma 22 serves as the fundamental ingredient for the proof of the generalized result:

Theorem 23. UnambSAC^k $\subseteq$ UnambAPDA^k *for* $k \geq 1$.

Proof. A leveled circuit of depth $O(\log^k n)$ can be regarded as a circuit of $O(\log^{k-1} n)$ circuit layers, each of depth $\log n$ (i.e., each of these circuit layers is an UnambSAC^1-circuit). The separation into different layers can be done easily because of the w.l.o.g. assumed leveled-ness and logspace uniformity of the simulated circuit. The essential trick is that we now do a simulation for each of these layers similar to that of Lemma 22. Here the problem arises that, when simulating such an UnambSAC^1-circuit layer by an AuxPDA M, in general we do not have automatically the values of the input gates at disposal. Therefore, M recursively computes those values each time they are needed. (Note that the simulation starts in the highest (that is, output-) UnambSAC^1-layer of the given circuit.)

Observe that at a transition from one circuit layer to another in some respect we forget information. That is, to compute the value of an input gate g of some circuit layer i, a (re)computation which actually provides the saturation of g is performed. But then M is only interested in whether g has saturation greater than 0 (i.e., g has value 1) or g has saturation 0 (i.e., g has value 0). It is necessary to "forget" the actual value at this point because, otherwise, the saturation values were no longer polynomially bounded.

Nevertheless, the whole simulation obviously remains unambiguous. The logarithmic space bound and the correctness of the simulation are straightforward and such what remains to be shown is the time bound $2^{O(\log^k n)}$ $(= n^{O(\log^{k-1} n)})$: For each Unamb$S\!\!\!\!E$AC^1-circuit layer only polynomially many recursive calls are performed by M due to the polynomial time bound of the simulation of Unamb$S\!\!\!\!E$AC^1-circuits of Lemma 22. Observe that in polynomial time at most a polynomial number of questions to the input bits of the circuit take place whose determination leads to the recursive calls. Thus, the recursion depth of $O(\log^{k-1} n)$ yields a total running time of $n^{O(\log^{k-1} n)}$. $\square$

We separated an Unamb$S\!\!\!\!E$AC^k-circuit in layers of depth $O(\log n)$. This is the only possibility we had, because, if the layers were chosen "thicker" than $O(\log n)$, then the space needed by the simulating AuxPDA M would become greater than $O(\log n)$ and if the layers were chosen "thinner" than $O(\log n)$, then the simulation time would become greater than $2^{O(\log^k n)}$: Assume that we separate C in circuit layers of arbitrary depth D. Let $T(D)$ (resp. $S(D)$) denote the time (resp. space) that are needed for the simulation of such a circuit layer with the help of the techniques of Lemma 22. Then $T(D) = \max(2^{O(D)}, n^{O(1)})$ and $S(D) = \max(O(D), O(\log n))$. (Observe that for $D > 0$ we have to assume a polynomial size for each of those circuit layers.) Evidently, $D = O(\log n)$ is optimal to gain a simulation in polynomial time and logarithmic space. For the running time $t(n)$ of the whole simulation analogous to Theorem 23 it holds

$$t(n) = T(D)^{\log^k n/D} \geq n^{O(\log^k n/D)} = 2^{O(\log^{k+1} n/D)}.$$

Consequently, if $D = o(\log n)$, then $t(n) = \omega(2^{O(\log^k n)})$.

In the end of this section, the closure under complementation for strongly unambiguous, semi-unbounded fan-in circuits is investigated. Whereas there is little hope to prove the closure under complementation for UnambSAC^k, the construction of Borodin et al. (1989) for complementing circuits fully applies to Unamb$S\!\!\!\!E$AC^k. Since this construction does not work for UnambSAC^k, we take a different approach via Lemma 17. By a slight modification of Lemma 17 it is possible to show that Co-UnambSAC^1 is included in ReachUnambAPDA^1 (see (Buntrock et al., 1991)), the class of languages recognized by polynomial time and logarithmic space bounded AuxPDAs, where each configuration is *reachable* from the start configuration by at most one computation path. Of course, by definition it holds StUnambAPDA$^1 \subseteq$ ReachUnambAPDA$^1 \subseteq$ UnambAPDA^1. Recall that in Theorem 15 it was proved that UnambSAC^1 and StUnambAPDA^1 coincide, so the following theorem could also be stated for strongly unambiguous AuxPDAs instead of UnambSAC^1-circuits.

Theorem 24. Co-UnambSAC$^1 \subseteq$ ReachUnambAPDA^1.

Proof. Let C be the given UnambSAC^1-circuit, which w.l.o.g. is assumed to be leveled. As mentioned before, a slightly modified version of Lemma 17 is applied. The gate measure will be the simple Boolean values of the (evaluated) gates. Only one addition to the algorithm of Lemma 17 is made. After nondeterministically guessing a gate value, the simulating AuxPDA additionally pushes this guessed value (0 or 1) on the push-down store. Clearly, guessed values are verified with the strongly unambiguous AuxPDA from the equality UnambSAC^1 = StUnambAPDA^1 (Theorem 15). Similar to Theorem 16 the idea behind this pushing of additional information (i.e., guessed values) on the store is to record paths of guesses in order to guarantee the unique reachability of configurations (from the start configuration). In these paths of guesses the history of nondeterministic decisions of the simulating AuxPDA M (except for the strongly unambiguous verifying algorithm) is protogeniced. These paths of guesses only are popped from the push-down, when M has found out the value of the output gate g of C. (M finally accepts, iff g evaluates to 0.) This is due to the fact that in Lemma 17 the push-down store is only needed for the verification of guessed values. Since the configuration where M has ascertained the value of g is reachable by exactly one computation path (with an uniquely determined, corresponding path of guesses), we can easily conclude that all configurations of M are reachable by at most one computation path from the start configuration. Thus application of such modified Lemma 17 yields the statement of Theorem 24. □

Perhaps one could be tempted to assume that the above simulation ('modified Lemma 17') can even be done in a strongly unambiguous manner (and we can conclude UnambSAC^1 = Co-UnambSAC^1). But simple considerations show that this is not true because for (unreachable) configurations where normally by inductive counting ascertained numbers are pretended wrongly there can be several ways leading to the accepting configuration.

In contrast to UnambSAC^k, whose closure under complementation seems to be unlikely, UnambSEAC^k is closed under complementation.

Theorem 25. UnambSEAC^k = Co-UnambSEAC^k *for* $k \geq 1$.

Proof. It can be observed that the construction of Borodin et al. (1989) for complementing SAC^k-circuits directly transfers to UnambSEAC^k-circuits. The essential point is that all unbounded OR-gates in the construction for the complementing circuit of Borodin et al. (1989) can be replaced by vulnerable unbounded OR-gates, i.e., all of them have at most one input evaluating to one. Moreover, the so-called THRESHOLD-gates additionally needed there can be replaced by monotone NC^1-circuits (Ajtai et al., 1983; Borodin et al., 1989), thus unbounded OR-gates are not necessary in this case. □

6. Normal forms for AuxPDAs

In this section, we will utilize the characterizations of AuxPDAs by semi-unbounded fan-in circuits to prove some normal form results. First, we deal with the restriction of push-down heights in particular for unambiguous AuxPDAs and, second, we introduce the notion 'oblivious' for AuxPDAs and show that in the most interesting cases it is no restriction to demand obliviousness. In addition, oblivious and unambiguous AuxPDA classes will prove to coincide with $\mathrm{WeakUnambSAC}^k$ and $\mathrm{UnambSAC}^k$ for arbitrary k. In this way, we extend the results of Section 4, where only a characterization for $k = 1$ was given.

6.1. AuxPDAs with restricted push-down height. For Turing machines there is great interest in simultaneous resource bounds, i.e., restricting time and space bounds at the same time. As far as AuxPDAs are concerned, one most of the time deals with simultaneous bounds on running-time and working space. But what about the unlimited push-down store? There has also been a lot of research to restrict the size of the push-down store. Mager (1969) showed that a push-down store suffices whose size is exponential in the space bound. Harju (1979) showed (also see (Ruzzo, 1980) for an alternative proof) that deterministic AuxPDAs with polynomial running-time and logarithmic working-tape can be simulated by deterministic AuxPDAs with logarithmic space and $O(\log^2 n)$ push-down height. However, the simulation yields a super-polynomial running-time. But later on, Dymond and Ruzzo (1986) proved the above result where even the polynomial running-time can be preserved. The dual result for nondeterministic AuxPDAs (with also preservation of the polynomial running-time) was shown earlier by Ruzzo (1980).

Subsequently, we will restrict push-down height for nondeterministic, unambiguous, and strongly unambiguous AuxPDAs. For this purpose, we make use of the characterization of AuxPDAs by semi-unbounded fan-in circuits. This is done in the following way. Assume that we have a semi-unbounded fan-in circuit C of depth $d(n)$ and size $z(n)$ simulating an AuxPDA M. Then we again simulate C by an AuxPDA N in the usual way (cf. (Venkateswaran, 1991)): Starting at the output gate, for AND-gates we examine both children and for OR-gates only one guessed child. Because we only need to store constant many parameters (with a space requirement of $O(\log z(n))$) of the recursive calls, a push-down height of $O(d(n) \cdot \log z(n))$ is immediate. Furthermore, N has the same time and space complexity as M. Our first

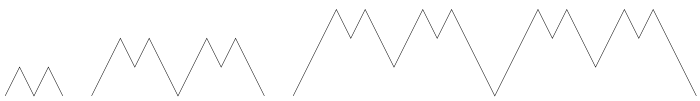


FIGURE 2. W-cycles.

application of the described technique yields an alternative proof for a result due to Ruzzo (1980).

Theorem 26. (Ruzzo, 1980) *A language L is accepted by a NAuxPDA in* $\log n$ *space and* $2^{O(\log^k n)}$ *time iff L is accepted by such a machine which, furthermore, uses at most* $O(\log^{k+1} n)$ *push-down height.*

Proof. Just make use of the technique described above, using Venkateswaran's equality $\mathrm{SAC}^k = \mathrm{NAPDA}^k$. □

Making use of two of the main results of this paper, we further gain the proposed restriction of the push-down heights for unambiguous and strongly unambiguous AuxPDAs. Unfortunately, we have such a result only for polynomial time AuxPDAs. Note that in the case of unambiguous AuxPDA's the result of Theorem 27 already was obtained by Buntrock (1989).

Theorem 27. *L is accepted by an unambiguous (resp. strongly unambiguous) AuxPDA in* $\log n$ *space and polynomial time iff L is accepted by such a machine which, furthermore, uses at most* $O(\log^2 n)$ *push-down height.*

Proof. It is sufficient to utilize the equality $\mathrm{WeakUnambSAC}^1 = \mathrm{UnambAPDA}^1$ (resp. $\mathrm{UnambSAC}^1 = \mathrm{StUnambAPDA}^1$) given in Theorem 15 in combination with the above described technique. □

6.2. Obliviousness for AuxPDAs. An automaton is called *oblivious* if the movements of all its heads are independent from the input except its length. This property is easily achieved for space bounded Turing machines: Roughly speaking, we just always move the heads to and from the two ends of the respective tape contents.

For AuxPDAs obliviousness is not so easy to attain because of the push-down store head. But here the characterization of AuxPDAs by circuits applies. The main idea again is to simulate a circuit by an AuxPDA. If the circuit is strictly alternating (i.e., for all $i \geq 0$, all gates on level $2i+1$ are OR-gates and all gates on level $2i+2$ are AND-gates) and leveled, then the profile will also be very regular. This special shape of a profile is called W-*cycle*[‡] (cf. Figure 2). Because we only consider logspace and at least polynomial time AuxPDAs, for the input and working tapes of the AuxPDAs we can use the above mentioned technique for space bounded Turing machines and, therefore, we altogether get an oblivious AuxPDA.

[‡]This name is taken from numerical mathematics, where it is used in the theory of multi-grid methods.

To simulate a circuit C (which itself simulates a given AuxPDA M), we employ nearly the same technique as in the preceding subsection. The only difference is that when we evaluate an AND-gate g (which w.l.o.g. shall have exactly two inputs), we do this in a slightly modified way. First, we push the left input gate of g on the store, then we evaluate it, afterwards we pop it from the store, and, finally, we compute the right input gate of g and do the analogous computation for this right gate. Note that we only need the store for the evaluation of AND-gates. Because of the 'symmetry' of both the sub-circuits of the AND-gate this altogether yields a profile in which the following holds. If we divide it into two equal parts (left and right) both are symmetric to each other and this also holds for a recursive division of these parts. In this way, we gain profiles in W-cycle form (Figure 2), i.e., a special case of obliviousness. The following classes with prefix 'W-cycle' are defined in the intuitive way.

Theorem 28. (1) $\mathrm{NAPDA}^k = (\mathsf{W}\text{-cycle})\mathrm{NAPDA}^k$.
(2) $\mathrm{UnambAPDA}^1 = (\mathsf{W}\text{-cycle})\mathrm{UnambAPDA}^1$.
(3) $\mathrm{StUnambAPDA}^1 = (\mathsf{W}\text{-cycle})\mathrm{StUnambAPDA}^1$.

Proof. To prove Theorem 28 we make use of the equations $\mathrm{NAPDA}^k = \mathrm{SAC}^k$ (Venkateswaran, 1991), $\mathrm{UnambAPDA}^1 = \mathrm{WeakUnambSAC}^1$, and $\mathrm{UnambSAC}^1 = \mathrm{StUnambAPDA}^1$ (Theorem 15). According to Proposition 1 and subsequent remarks circuits of all these classes can be assumed to be leveled. Furthermore, it only needs little effort to see that all circuits of these classes can be made strictly alternating by at most doubling the depth. Now the simulation technique described above Theorem 28 provides the desired result. □

The question whether $\mathrm{UnambAPDA}^k = \mathrm{WeakUnambSAC}^k$ and $\mathrm{StUnambAPDA}^k = \mathrm{UnambSAC}^k$ hold for $k > 1$, remained open in Section 4. We cannot fully answer this question, but if we confine the consideration to W-cycle-oblivious AuxPDA classes, we get the desired equality for arbitrary k.

Theorem 29. (1) $(\mathsf{W}\text{-cycle})\mathrm{UnambAPDA}^k = \mathrm{WeakUnambSAC}^k$.
(2) $(\mathsf{W}\text{-cycle})\mathrm{StUnambAPDA}^k = \mathrm{UnambSAC}^k$.

Proof. The '$\supseteq$'-directions only require a straightforward generalization of the proof of Theorem 28, where essentially $\mathrm{WeakUnambSAC}^1 \subseteq (\mathsf{W}\text{-cycle})\mathrm{UnambAPDA}^1$ (resp. $\mathrm{UnambSAC}^1 \subseteq (\mathsf{W}\text{-cycle})\mathrm{StUnambAPDA}^1$) was shown, to arbitrary $k \geq 1$.

To prove the reverse direction, we make use of the 'totally symmetric' shape of the profiles for W-cycle-oblivious AuxPDAs. The essential advantage of these W-cycle-oblivious AuxPDAs is that we always can separate a profile into two equal sized paths. Thus it is *not* necessary to store information about the length of the computation paths in order to get a balanced and unique decomposition of profiles (and thus, computation paths). We consider pairs of surface configurations in order to construct the simulating circuit. Mainly we need gates named $\langle A, B\rangle$ that compute

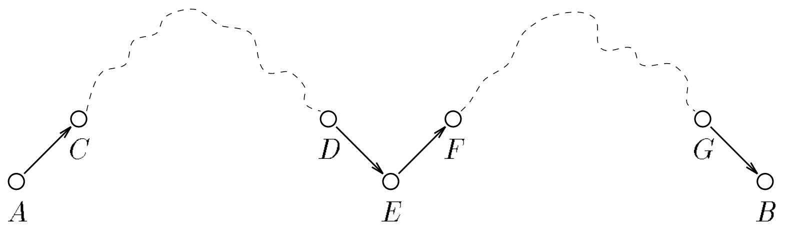


FIGURE 3. Decomposition components of a W-cycle profile.

whether there exists a computation from surface configuration A to B, where the level of the push-down store is the same for A and B and does not go below this level during the computation (cf. Figure 3). These gates are defined as

$$\langle A, B\rangle \equiv \exists_{C,\ldots,G}\langle C, D\rangle \wedge \langle F, G\rangle \wedge \langle A\to C, D\to E\rangle \wedge \langle E\to F, G\to B\rangle,$$

where, for example, $\langle A \to C, D \to E\rangle$ computes whether there is one push-step from A to C and one pop-step from D to E (where first a symbol a is pushed and then popped). Of course, $\langle A, A\rangle$ has value 1 for every surface configuration A. The output gate of the simulating circuit then is $\langle S, F\rangle$, where S (resp. F) are the uniquely defined start (resp. accepting end) (surface) configurations (both with empty push-down store) of the simulated AuxPDA.

The correctness and the weak (resp. strong) unambiguity of such defined circuits is shown similar as in the proofs of Lemma 11 and 12, respectively. Because there are only polynomially many surface configurations and we recursively divide profiles into two equal sized paths, a polynomial size and a depth of $O(\log^k n)$ of the circuit suffice. □

7. Conclusion and open questions

In summary, we feel that we have shed some more light on the concept of unambiguity in the realm of NC. We hope that the results of this paper clarified relations between AuxPDAs and semi-unbounded fan-in circuits. Furthermore, it should have become apparent how useful characterizations of AuxPDAs by semi-unbounded fan-in circuits are in order to employ inductive counting methods or to gain normal form results. We have come to the conclusion that strong unambiguity seems to be a concept more suitable for the consideration of classes within the NC-hierarchy than (the conventional) weak unambiguity is. This impression derives from the facts that CREW-PRAMs are characterized by a strongly unambiguous circuit class (Lange, 1993) as well as LOGUCFL shows tight relations to strongly unambiguous circuits and AuxPDAs. The usefulness of strong unambiguity and related concepts can informally be explained if one thinks of simulations of such restricted automata. A simulation 'in parallel' often deals with the whole computation graph of the simulated machine, so restrictions only for accepting computation paths often do not suffice for the possibility of efficient simulations. To conclude, Figure 4 comprises several of the results of Sections 4 and 5 settled in the NC-hierarchy between NC^1 and AC^1.

Several results obtained in Section 4 only hold for polynomial time ('$k = 1$') computations. Thus naturally the question arises whether these results can be generalized to super-polynomial time bounds ('$k > 1$'). The same question can be posed for the normal form results of Section 6. For Section 5 the general question for further applications of inductive counting on circuits emerges.

Acknowledgement. We thank Gerhard Buntrock and Birgit Jenner for fruitful discussions. In addition, we are grateful to the anonymous referees for several corrections, insightful remarks, and their comprehensive reports. We are particularly indebted to Klaus-Jörn Lange for making us familiar with the considered problems, for his support and encouragement, and for allowing us to include results of (Lange and Rossmanith, 1990).

References

Ajtai, M., Komlós, J., and Szemerédi, E. (1983). Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19.

Àlvarez, C. and Jenner, B. (1993). A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30.

Balcázar, J., Díaz, J., and Gabarró, J. (1990). *Structural Complexity Theory I and II*. Springer.

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

Borodin, A., Cook, S. A., Dymond, P. W., Ruzzo, W. L., and Tompa, M. (1989). Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578.

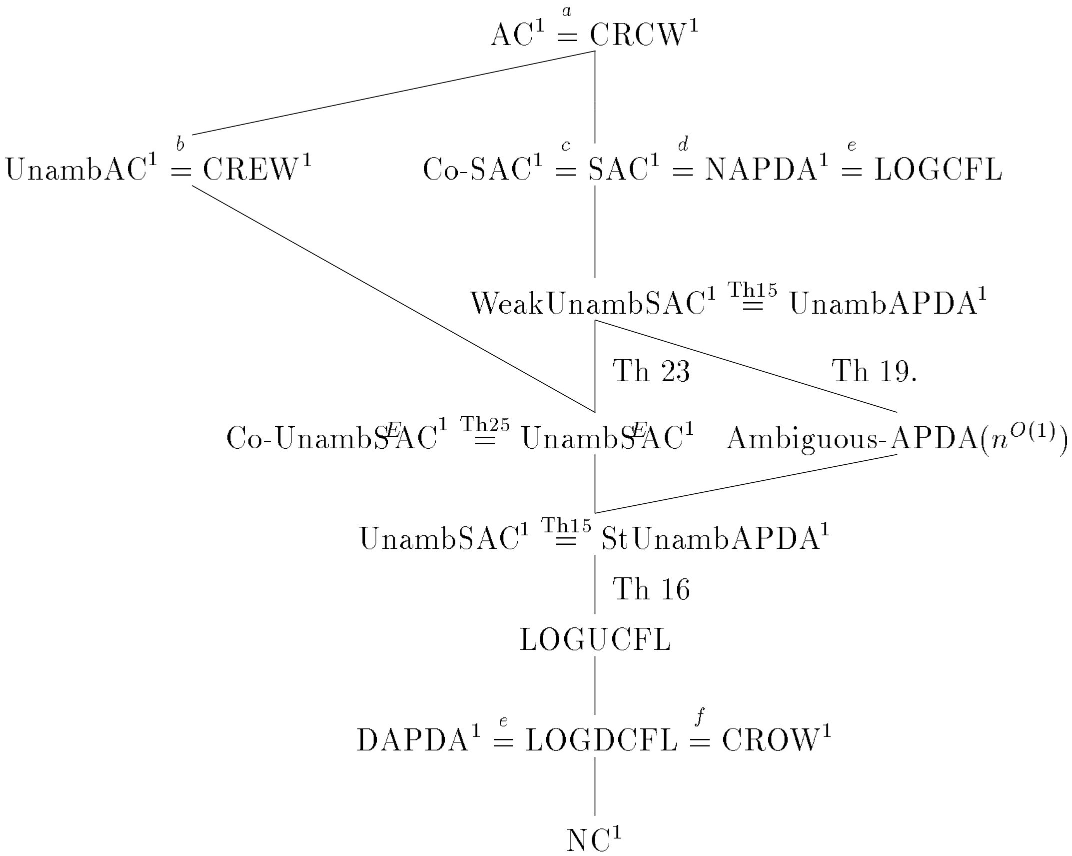
Buntrock, G. (1989). *Logarithmisch platzbeschränkte Simulationen*. Dissertation, TU Berlin. (in German).

Buntrock, G., Hemachandra, L. A., and Siefkes, D. (1993). Using inductive counting to simulate nondeterministic computation. *Information and Computation*, 102:102–117.

Buntrock, G., Jenner, B., Lange, K.-J., and Rossmanith, P. (1991). Unambiguity and Fewness for Logarithmic Space. In Budach, L., editor, *Proc. of 8th Conference on Fundamentals of Computer Science*, number 529 in Lecture Notes in Computer Science, pages 168–179, Gosen, Federal Republic of Germany. Springer-Verlag.

Chandra, A. K., Kozen, D., and Stockmeyer, L. (1981). Alternation. *Journal of the ACM*, 28:114–133.

Cook, S. A. (1971). Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18:4–18.



[a](Stockmeyer and Vishkin, 1984)
[b](Lange, 1993)
[c](Borodin et al., 1989)
[d](Venkateswaran, 1991)
[e](Sudborough, 1978)
[f](Dymond and Ruzzo, 1986)

FIGURE 4. Old and new relationships for classes between NC^1 and AC^1.

Cook, S. A. (1979). Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proc. of 11th ACM Symposium on Theory of Computing*, pages 338–345.

Dymond, P. W. and Ruzzo, W. L. (1986). Parallel RAMs with owned global memory and deterministic language recognition. In *Proc. of 13th Colloquium on Automata, Languages and Programming*, number 226 in Lecture Notes in Computer Science, pages 95–104. Springer-Verlag.

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

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

Goldschlager, L. M. (1982). A universal interconnection pattern for parallel computers. *Journal of the ACM*, 29(3):1073–1086.

Grollmann, J. and Selman, A. (1988). Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17:309–335.

Hagerup, T. and Radzig, T. (1990). Every robust CRCW PRAM can efficiently simulate a PRIORITY PRAM. In *Proc. of 2d ACM Symposium on Parallel Algorithms and Architectures*, pages 117–124.

Harju, T. (1979). A simulation result for the auxiliary pushdown automaton. *Journal of Computer and System Sciences*, 19:119–132.

Harrison, M. A. (1978). *Introduction to Formal Language Theory*. Addison-Wesley.

Hartmanis, J. and Hemachandra, L. A. (1988). Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58:129–142.

Hartmanis, J. and Yesha, Y. (1984). Computation times of NP sets of different densities. *Theoretical Computer Science*, 34:17–32.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Ibarra, O. H., Jiang, T., and Ravikumar, B. (1988). Some subclasses of context-free languages in NC^1. *Information Processing Letters*, 29:111–117.

Immerman, N. (1988). Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17(5):935–938.

Karp, R. M. and Ramachandran, V. (1990). Parallel algorithms for shared-memory machines. In (van Leeuwen, 1990), chapter 17, pages 869–941.

Kasami, T. (1972). A note on computing time for recognition of languages generated by linear grammars. *Information and Control*, 10:209–214.

Lange, K.-J. (1990). Unambiguity of circuits. In *Proc. of 5th Conference on Structure in Complexity Theory*, pages 130–137.

Lange, K.-J. (1993). Unambiguity of circuits. *Theoretical Computer Science*, 107:77–94.

Lange, K.-J. and Rossmanith, P. (1990). Characterizing Unambiguous Augmented

Pushdown Automata by Circuits. In Rovan, B., editor, *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in Lecture Notes in Computer Science, pages 399–406, Banská Bystrica, Czechoslovakia. Springer-Verlag.

Lenz, K. (1992). *Die Komplexität Boolscher Funktionen in Schaltkreisen über der Basis* $\{\oplus, \wedge\}$. Dissertation, Johann Wolfgang Goethe–Universität. (in German).

Mager, G. (1969). Writing pushdown acceptors. *Journal of Computer and System Sciences*, 3:233–247.

Niedermeier, R. and Rossmanith, P. (1992). Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits. In Simon, I., editor, *Proceedings of the 1st Symposium on Latin American Theoretical Informatics*, number 583 in Lecture Notes in Computer Science, pages 387–400, São Paulo, Brazil. Springer-Verlag.

Parberry, I. (1987). *Parallel Complexity Theory*. Pitman.

Pippenger, N. (1979). On simultaneous resource bounds. In *Proc. of 20th IEEE Symposium on Foundations of Computer Science*, pages 307–311.

Rossmanith, P. and Rytter, W. (1992). Observations on $\log n$ time parallel recognition of unambiguous context-free languages. *Information Processing Letters*, 44:267–272.

Ruzzo, W. L. (1980). Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21:218–235.

Ruzzo, W. L. (1981). On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383.

Rytter, W. (1987). Parallel time $O(\log n)$ recognition of unambiguous context-free languages. *Information and Computation*, 73:75–86.

Stockmeyer, L. and Vishkin, U. (1984). Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422.

Sudborough, I. H. (1978). On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414.

Szelepcsényi, R. (1988). The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284.

Valiant, L. (1976). The relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23.

van Leeuwen, J., editor (1990). *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*. Elsevier.

Venkateswaran, H. (1991). Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404.

Vinay, V. (1991). Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proc. of 6th Conference on Structure in Complexity Theory*, pages 270–285.

Wagner, K. and Wechsung, G. (1986). *Computational complexity*. Reidel Verlag, Dordrecht and VEB Deutscher Verlag der Wissenschaften, Berlin.

Rolf Niedermeier, Peter Rossmanith, Technische Universität München, Fakultät für Informatik, Arcisstr. 21, 80333 München, Fed. Rep. of Germany