Counting and empty alternating pushdown automata*

Klaus Reinhardt
Universität Stuttgart, Institut für Informatik
Breitwiesenstr.22, D-7000 Stuttgart-80, Germany
e-mail: reinhard@informatik.uni-stuttgart.de

Abstract

We show that the class of context free languages CFL is not equal to $\oplus 1-PDA$ $(= \oplus CFL)$, the class of languages, which are recognized by a nondeterministic one-way push-down automaton equipped with parity acceptance. Furthermore we show that $LOG(\oplus CFL) = \oplus AuxPDA_{pt}$ contains all languages, which can be recognized by a uniform weak unambiguous AC^1-circuit introduced in [LR90a]. Therefore, it contains all languages, which can be recognized by a $CREW$-PRAM. We show, that $L^{\#Aux^{log}PDA_{pt}}$ is contained in uniform TC^1, which sharpens a result in [Vin91], where inclusion in NC^2 was shown. At last we show, that AC^k, SAC^k and P can be characterized as the logarithmic closure of certain types of languages. These languages are either those given by linear alternating grammars or they are languages which are recognized by alternating pushdown automata with the restriction that their storage has to be empty if they make an alternating transition. In particular, we consider the cases when the depth of alternation is bounded by a constant or a polylogarithmic function.

1 Introduction

In [Val79] Valiant defined the counting class $\#P$ for polynomial timebounded Turing machines. More generaly we define the counting class $\#X$ as the class of functions f such that there is a nondeterministic automaton or grammar of the kind X (whatever X is) and f tells us for an input i the number $f(i)$ of accepting computation paths of the automaton rsp. leftmost derivations of the grammar. In analogy to the definition of MOD_kL in [BDHM91] we can define the class Mod_kX as the class of of languages A with $\exists f \in \#X\ \forall i\ i \in A \Leftrightarrow f(i) \not\equiv 0\ mod\ k$. According to [GNW90] we can write this as $\{x \not\equiv 0\ mod\ k\}$X and define the classes $NX := \{x > 0\}X$, $co-NX := \{x = 0\}X$, $EX := \{x = y\}X$ and $GX := \{x > y\}X$. Of special interest is $\oplus X := Mod_2X$ [Dam90]. For polynomial timebounded Turing machines ($X = P$), logarithmic spacebounded Turing machines ($X = L$) or polynomial timebounded push-down automata with auxiliary logarithmic spacebounded tape ($X = Aux^{log}PDA_{pt}$) no separation results between all these classes are known. In Section 2 we show some separations for context free grammars ($X = CFL$) rsp. One-way push-down automata without auxiliary tape ($X = 1-PDA$).

*this research has been partially supported by the EBRA working group No. 3166 ASMICS and by DFG-SFB 342, Teilprojekt A4 "KLARA".

There are various connections between formal language classes and complexity classes [Lan89] [Rei89] [Rei90], where we need the logarithmic closure over a class S of languages $LOG(S) := \{Y \mid \exists X \in S \text{ with } Y \leq^{log} X\}$ with $L \leq^{log} L'$, if there is a logarithmic space-bounded transducer T with $x \in L$ if and only if $f_T(x) \in L'$. In Section 3 we will see, that push-down automata with Y-acceptance ($Y \in \{\oplus, N, co-N, E, G, Mod_k\}$) without two-way input and without log-space working tape generate languages which are complete for the complexity class $YAux^{log}PDA_{pt}$ with respect to $\leq^{log}$-reduction. Thus we generalize the equation $NAux^{log}PDA_{pt} = LOG(CFL)$ of Sudborough in [Sud78] to $YAux^{log}PDA_{pt} = LOG(Y1-PDA)$. This result may be interpreted in the sense that in $YAux^{log}PDA_{pt}$-automata the push-down part may be separated from the log-space/two-way part. This decomposition is possible for (full) alternating push-down automata as shown in [Rei89] and [Rei90] improving a result of [JK89]. We want to remark here that this relation does not seem to hold in general: it appears that unambiguous automata do not allow for decompositions of this kind.
There are also various connections to circuit complexity classes [Ruz81], [Ven88], [LR90a] and Parallel Random Access Machines [SV84] ,[AJ93], [NR92] [Ros91]. In Sections 4 and 5 we will add two further connections to the graph describing inclusion structures of complexity classes. The picture will give a overview on the inclusions of the various classes.
A more complicated kind of acceptance is the alternation [CKS81]. This method is important, because it gives an equivalent characterization of the polynomial hierarchy. Applying alternation to push-down automata yields very high complexity classes as the polynomial hierarchy, PSPACE or EXPTIME [Rei89], [LSL84], [LLS84]. This was the reason to introduce in [LR91] the notion of *empty alternation* by investigating alternating automata which are restricted to empty their storage except for a logarithmically space-bounded tape before making an alternating transition. Hereby new characterizations of the classes AC^k, SAC^k and P and applying this to polynomial timebounded Turing machines a new characterization for Wagners class $\Theta_2^P := L^{NP} = P^{NP[log]}$ [Wag88] was obtained. In Section 6 we apply empty alternation to one-way push-down automata (We will write $1-EA\Sigma_aPDA$). Hereby we obtain similar results like those in Section 3 (We will write $EA\Sigma_a^{log}PDA_{pt}$ for ($a-1$ times) empty alternating $Aux^{log}PDA_{pt}$). In Section 7 we apply alternation to linear grammars and obtain similar results like those in Section 6.

2 Separating classes with different context free acceptances

It is easy to see, that using the methods in [HU79], that there is a one to one correspondence between leftmost derivations of a grammar and the accepting paths of a push-down automaton. Thus it holds $YCFL = Y1-PDA$ for $Y \in \{\oplus, N, co-N, E, G, Mod_k\}$.

Theorem 2 $\oplus CFL \not\subseteq CFL$

Proof: The language $\{a^nb^mc^l \mid n \geq m \oplus m \geq l \oplus l \geq n\}$ is a member of $\oplus CFL$ because it is generated by the grammar $\{S \to Cc \mid aA \mid B, C \to Cc \mid aC \mid D, D \to aDb \mid ab, A \to aA \mid E, E \to bE \mid F, F \to bFc \mid bc, B \to Bc \mid G, G \to aGc \mid aHc, H \to bH \mid b\}$. But the assumption $L \in CFL$ leads to a contradiction when pumping the word $a^nb^nc^n$. ∎
In the same way it holds $Mod_kCFL \not\subseteq CFL$ for any k and $ECFL \not\subseteq CFL$ and $GCFL \not\subseteq CFL$ follow from $ECFL \supseteq coCFL$ and $GCFL \supseteq coCFL$.

3 The logarithmic closure over push-down automata with different acceptances

Theorem 3 $Y Aux^{log} PDA_{pt} = LOG(Y1\text{–}PDA)$ *holds for* $Y \in \{\oplus, N, co\text{–}N, E, G, Mod_k\}$

'$\supseteq$' is clear, the idea for '$\subseteq$' is the following: A surface configuration contains the logarithmic spacebounded tape and the state, but not the stack, which means that the number of such transitions is polynomial. The logarithmic spacebounded transducer writes every possible transition on surface configurations of the $Aux^{log}PDA$ in a list. Hereby every item of the list contains a surface configuration written reversely, followed by the information about what is popped or pushed, followed by the following surface configuration in the transition. Then the transducer writes a block marker and repeats the whole list $p(|x|)$ times, where p is the polynomial, which bounds the time of the $Aux^{log}PDA$. Now the $1{-}PDA$ can simulate the $Aux^{log}PDA$ by using the top of the pushdown stack to find from one transition to an appropriate following transition in the next block by comparing the surface configurations (if it guesses the wrong position, it rejects). There will be a one to one correspondence between the accepting paths.

4 Evaluating circuits with a $\oplus$push-down automaton

Theorem 4 $WeakUnambAC^1 \subseteq \oplus Aux^{log} PDA_{pt}$

Proof: $WeakUnambAC^1$ is defined in [LR90a] in the meaning, that an AND-(OR-)gate with unbounded fan-in may have an undefined value, if more than one input signal has the value 0 (1). This means, we need not care about these situations and so we can regard these gates as replaced by $\oplus$ gates, which have the same behavior in the remaining cases. We also regard the OR-gates as replaced by AND- and NOT-gates. An $Aux^{log}PDA_{pt}$ walks through the circuit in a way such that at the moment, when it enters a gate the first time, the number of accepting paths in the sub-computation tree following this point is odd, iff the value of the gate is 1. We get this by induction on the following behavior: Reaching an $\oplus$-gate, the automaton nondeterministically branches to one of the inputs. Reaching a NOT-gate, the automaton nondeterministically branches to go to its input or accept (add 1 modulo 2). Reaching an input signal, the automaton rejects, if it is 0, otherwise the automaton pops a gate and continues there or if the pushdown store is empty accepts. Reaching an AND-gate, the automaton pushes one of its inputs and goes to the other one (this multiplies modulo 2). Since the depth is logarithmic, the runtime is polynomial on every path ∎

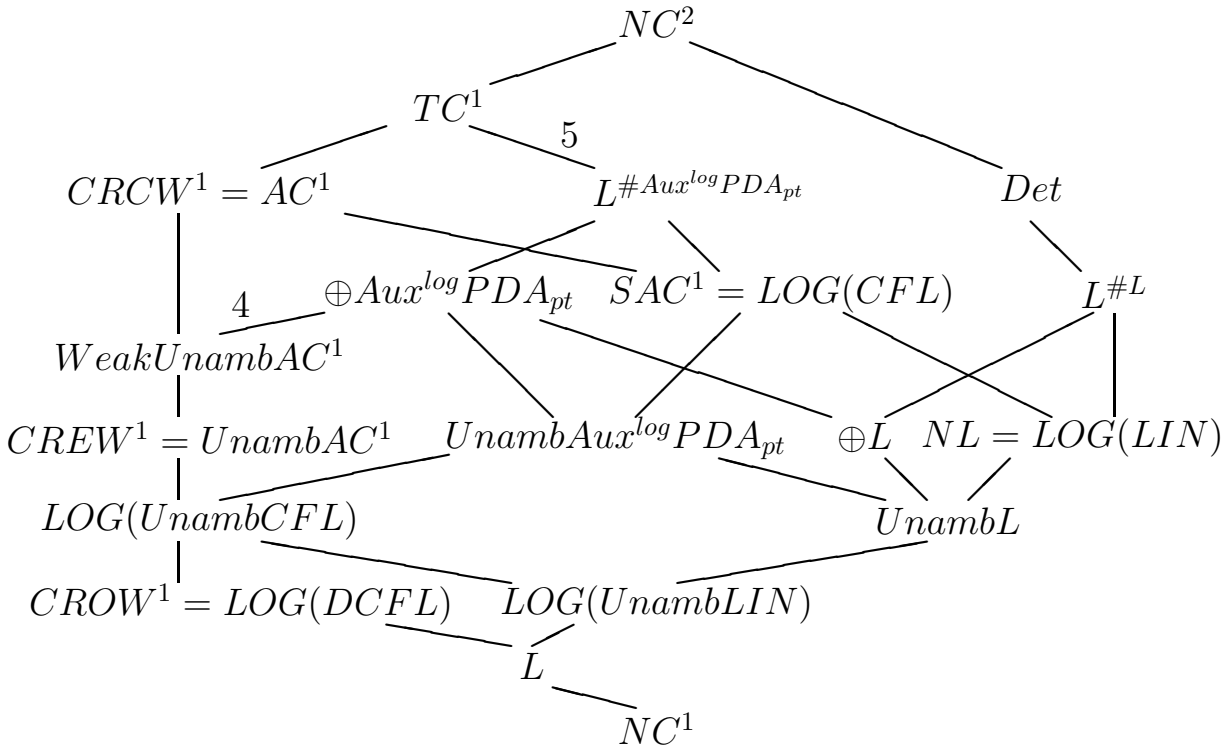
5 Counting accepting paths with a threshold circuit

Definition $L^{\#X} := \{A \mid A$ is log-space Turing-reducible to $bin(f)$ for some $f \in \#X\}$ with $bin(f) := \{(x,i) \mid$ the i-th bit of $f(x)$ is $1\}$.

Theorem 5 $L^{\#Aux^{log}PDA_{pt}} \subseteq TC^1$

Proof: According to [Vin91] [LR90a] [LR90b] [NR91] $\#Aux^{log}PDA_{pt} = \#SAC$, the number of accepting subcircuits. To get this number for an OR-gate (with unbounded fan-in) we have to sum up the numbers of all inputs. To get this number for an AND-gate (with

bounded fan-in) we have to multiply the numbers of the (w.l.o.g. 2) inputs. According to [HHK91] both can be done by a threshold circuit with constant depth, hence a threshold circuit with logarithmic depth can calculate the number for the complete SAC^1-circuit and simulate the logarithmic spacebounded transducer on the result. ■



6 Empty alternating push-down automata

In [LR91], we have shown $EA\Sigma^{log}_{log^k n}PDA_{pt} = EA\Sigma^{log}_{log^k n}$ $(= A\Sigma^{log}_{log^k n} = AC^k)$ for each k. That is, the additional push-down store does not increase the computational power for empty alternating automata with a logarithmic spacebounded tape and $log^k n$ alternations, as long as we keep a polynomial time bound. In this section we will see, that the other direction holds true in the sense, that empty alternating push-down automata without two-way input and without log-space working tape generate languages which are complete for $EA\Sigma^{log}_{a(n)}PDA$. Thus we again generalize the equation $A\Sigma^{log}_1 PDA_{pt} = LOG(A1\Sigma_1 PDA)$ of Sudborough in [Sud78].
The definitions of alternating push-down automata differ concerning the role of λ-transitions. Fortunately this does not matter when considering empty alternation.

Definition A *1-way-empty-alternating-push-down automaton* is an 8-tuple

$$A = (Z_e, Z_u, \Sigma, \Gamma, \delta, z_0, \$, E)$$

with the set of states $Z = Z_e \cup Z_u$ consisting of existential states Z_e and universal states Z_u, the input alphabet Σ, the push-down alphabet Γ, the transition relation $\delta \subseteq (Z \times \Sigma \times \Gamma) \times (Z \times \Gamma^*)$, the start state z_0, the bottom symbol \$, the final states E, the rejecting states R, the configuration set $C_A = Z \times \Sigma^* \times \Gamma^*$, the start configuration

$\sigma_A(x) = \langle z_0, x, \$\rangle$ and the configuration transition relation $\langle z, x_1 x, gk\rangle \vdash_A \langle z', x, g'k\rangle$ if and only if $z, z' \in Z,\ k, g' \in \Gamma^*,\ g \in \Gamma,\ g' \in \Gamma^*$ and $\langle z, x_1, g, z', g'\rangle \in \delta$. If $z \in Z_a$, then a configuration $\langle z, x, k\rangle \in C_A$ is called a universal configuration. If $z \in Z_e$, then it is called an existential configuration. If $z \in E$ and $x = \lambda$, then it is called accepting.

Since an unaugmented push-down automaton has just a finite memory, it is not possible to count and bind the depth of alternation by an infinitely growing function. Thus we could only treat the cases of either unbounded or constant alternation depth. To cope with that problem, we apply a depth bound not within the automaton, but instead within the language:

Definition Let A be a one-way empty alternating push-down automaton. The $EA\Sigma_{a(n)}$-language of A is the set of all words x accepted by A, which have a finite accepting subtree of configurations within the computation tree of A on x such that

i) The root is the existential start configuration $\sigma_A(x)$,

ii) for every existential configuration c in the tree, there is a d with $c \vdash_A d$ in the tree,

iii) for every universal configuration c in the tree, all d's with $c \vdash_A d$ are in the tree,

iv) the leaves of the tree are accepting,

v) alternations from an existential to an universal configuration or vice versa are *only* allowed, if the push-down-store is *empty* (the push-down store is regarded as empty, if only the bottom symbol \$ is on the push-down store.[1]), and

vi) there are at most $a(n) - 1$ alternations on every path on the tree.

Thus, the $EA\Sigma_\omega$-language of S is $\mathrm{L}(A)$ and for $a(n) \leq b(n)$ the $EA\Sigma_{a(n)}$-language is a subset of the $EA\Sigma_{b(n)}$-language of A.

The set of all $EA\Sigma_{a(n)}$-languages of one-way empty alternating push-down automata is denoted by $1{-}EA\Sigma_{a(n)}PDA$. $1{-}SEA\Sigma_{a(n)}PDA$ is the set of languages which can be recognized by 1-way-empty-semi-unbounded-alternating- push-down automata, which are only allowed to make finitely many steps in universal states before alternating into an existential state.

Using the result of [BCD$^+$88], it is easy to see, that we then have

Theorem 6.1 *$LOG(1{-}EA\Sigma_k PDA) = LOG(CFL)$ for each k.*

Theorem 6.2

1. $AC^k = EA\Sigma^{log}_{log^k n}PDA_{pt} = LOG(1{-}EA\Sigma_{log^k n}PDA)$
2. $SAC^k = SEA\Sigma^{log}_{log^k n}PDA_{pt} = LOG(1{-}SEA\Sigma_{log^k n}PDA)$
3. $P = LOG(1{-}EA\Sigma_\omega PDA) = LOG(1{-}SEA\Sigma_\omega PDA)$

Proof: Since $1{-}(S)EA\Sigma_{a(n)}PDA$ is contained in $(S)EA\Sigma^{log}_{a(n)}PDA_{pt}$, and the later is closed under log-reducibility, and because $(S)AC^k = (S)EA\Sigma^{log}_{log^k n}PDA_{pt}$, it suffices to exhibit some $(S)AC^k$-complete set, which is generated as $(S)EA\Sigma_{log^k n}$-language by an one-way empty alternating push-down automaton. This is done in Lemma 6.1 and Lemma 6.2 below. ∎

Lemma 6.1 *$AC^k \subseteq LOG(1{-}EA\Sigma_{log^k n}PDA)$*

[1] A is not allowed to push further \$ symbols.

Proof: We define the following formal language CFE_k, which is in case of $k = \omega$ just a variation of the context-free-emptiness problem.

> $CFE_k := \{w \mid w =< S >^R \#\&vu$,where v is a concatenation of all $< A > \%\# < B_1 >^R \# < B_2 >^R \#...\# < B_i >^R \#\&$ for a production $A \to B_1B_2..., B_i \in P$ and u is a concatenation of all $< T > \&$ for terminals $T \in \Sigma$ for an encoding $<>$ of $\Sigma \cup V$ and a grammar $G = (\Sigma, V, P, S)$ with a word in $L(G)$ with a derivation tree not deeper than $O(log^k(|w|)$ using the productions in the order of v.$\}$

CFE_k is complete for AC_k:
The idea behind the AC^k-hardness of CFE_k is to transform conjunctions $B = B_1 \wedge B_2 \wedge \ldots \wedge B_n$ into context-free rules $B \Rightarrow B_1B_2 \ldots B_n$ and disjunctions $A = A_1 \vee A_2 \vee \ldots \vee A_n$ into (the set of) rules $A \Rightarrow A_1|A_2| \ldots |A_n$.
Let L be accepted by an AC^k circuit family. Using the unifying machine of this family, we can construct a log-space computable function f which maps an input x to an unbounded fan-in circuit of log^k depth, which evaluates to 1, iff $x \in L$. A logarithmic transducer T can convert such a circuit to a context-free grammar in the following way: We assume w.l.o.g., that the output gate is OR and that there are only connections from OR gates to AND gates and vice versa. If A is the output of an OR-gate and $B_1, ...B_i$ are the inputs of an AND-gate with its output in this OR-gate, then the production $A \Rightarrow B_1...B_i$ has to be in the grammar. The productions have to be in a monotone order, which means, that an encoding of a production with A on the left side has to be after the encoding of a production with A on the right side; this can simply be done by repeating the productions $log^k n$ times. All inputs having the value 1 are the terminals and the output of the circuit is the start-symbol of the grammar. A gate of the circuit has value 1 iff a word consisting of terminals can be derived from the corresponding variable, so T reduces L to CFE_k.
An alternating 1-way push-down automaton M recognizing CFE_ω works as follows: M chooses existentially a production and tests the reverse order encoding of the variable with the push-down store in existential states and thus emptying the push-down store. Then M chooses the variable on the right side of a production in an universal state without using the push-down store. M starts with $< S >$ on the push-down store. Clearly the $EA\Sigma_{log^k}$-language of M is CFE_k.
The following is the formal definition:

$$M = (\{z_0, z_1, z_2, z_a\}, \{z_u\}, \Sigma \cup \{\#, \%, \&\}, \Sigma \cup \{\$\}, \delta, z_0, \$, \{z_a\})$$

with

$$\begin{aligned}\delta := \{ \quad & (z_0, x, g, z_0, xg) \mid x \in \Sigma, g \in \Sigma \cup \{\$\}, \quad (z_0, \#, g, z_1, g), \\ & (z_1, y, g, z_1, g) \mid y \in \Sigma \cup \{\#, \%, \&\}, \quad (z_1, \&, g, z_2, g), \\ & (z_2, g, g, z_2, \lambda), \quad (z_2, \&, \$, z_a, \$), \quad (z_2, \%, \$, z_u, \$), \\ & (z_u, x', \$, z_u, \$), \quad (z_u, \#, \$, z_0, \$), \quad (z_u, \#, \$, z_u, \$), \\ & (z_u, \&, \$, z_a, \$), \quad (z_a, y, \$, z_a, \$)\}\end{aligned}$$

■

Lemma 6.2 $SAC^k \subseteq LOG(1\text{–}SEA\Sigma_{log^k n}PDA)$

Proof: Restricting the grammars to be in Chomsky normal form settles the semi-unbounded case: the bounded fan-in of AND-gates corresponds to the restriction to 2 variables on the right hand side of the productions of the simulated grammar.

We define $CFEC_k$ analogously to CFE_k with the difference, that the grammar must be in Chomsky normal form. $CFEC_k$ can be recognized by an automaton like in Lemma 6.1, which makes only one step in an universal state by deciding which variable on the right side of a production is used. $CFEC_k$ is complete for SAC_k. The bounded fan-in of AND-gates corresponds with the restriction to 2 variables on the right side of the productions. The formal definition is:

$$M = (\{z_0, z_1, z_2, z_3, z_a\}, \{z_u\}, \Sigma \cup \{\#, \%, \&\}, \Sigma \cup \{\$\}, \delta, z_0, \$, \{z_a\})$$

with

$$\begin{array}{rll} \delta := \{ & (z_0, x, g, z_0, xg) \mid x \in \Sigma, g \in \Sigma \cup \{\$\}, & (z_0, \#, g, z_1, g), \\ & (z_1, y, g, z_1, g) \mid y \in \Sigma \cup \{\#, \%, \&\}, & (z_1, \&, g, z_2, g), \\ & (z_2, g, g, z_2, \lambda), \quad (z_2, \&, \$, z_a, \$), & (z_2, \%, \$, z_u, \$), \\ & (z_u, \#, \$, z_0, \$), \quad (z_u, \#, \$, z_3, \$), & \\ & (z_3, x, \$, z_3, \$), \quad (z_3, \#, \$, z_0, \$), & (z_a, y, \$, z_a, \$)\}. \end{array}$$

■

7 Alternating linear Grammars

We now consider alternating grammars. There are several possibilities to do this for alternating context free grammars. So for alternating context free grammars in [Mor89] it holds $LOG(\lambda - freeACFL) = PSPACE$ according to [CT90], but for alternating (also λ-free) alternating context free grammars in [Rei89] it holds $LOG(CFL\Sigma_\omega) = EXPTIME$. In any case there are close relations to alternating push-down automata and complexity classes. But it seems difficult to introduce the concept of empty alternation within the framework of grammars. Instead of trying to do so, we will consider in this subsection alternating linear (context free) grammars, since linear grammars seem somehow to work without auxiliary storage, which implies in this case the coincidence of alternation and empty alternation. (Compare this with the $NSPACE(log(n))$-completeness of linear languages, and with the fact that for logarithmic space alternation and empty alternation coincide).
We now turn to alternating linear grammars:

Definition According to [Rei89] and [Rei90] we regard an alternating grammar G as an 8-tuple

$$G = (V, V_e, V_a, \Sigma, P_e, P_a, S, F) \text{ with } S, F \in V_e \cap V_a,$$

for $k > 0$ we define:

$$\begin{array}{c} SenF\Sigma_0(G) := SenF\Pi_0(G) := \{S\} \\ SenF\Sigma_1(G) := \{\alpha \in (V_a \cup \Sigma)^* \mid S \underset{P_e}{\overset{*}{\Longrightarrow}} \alpha\} \\ SenF\Pi_1(G) := \{\alpha \in (V_e \cup \Sigma)^* \mid \forall \beta \in \{S, F\}(\beta \underset{P_a}{\overset{*}{\Longrightarrow}} \alpha \ \rightarrow \beta = S)\} \\ SenF\Sigma_{k+1}(G) := \{\alpha \in (V_a \cup \Sigma)^* \mid \exists \beta \in (V_e \cup \Sigma)^* \cap SenF\Pi_k(G) \text{ with } \beta \underset{P_e}{\overset{*}{\Longrightarrow}} \alpha\} \\ SenF\Pi_{k+1}(G) := \{\alpha \in (V_e \cup \Sigma)^* \mid \forall \beta \in (V_a \cup \Sigma)^*(\beta \underset{P_a}{\overset{*}{\Longrightarrow}} \alpha \ \rightarrow \beta \in SenF\Sigma_k(G)\}. \end{array}$$

Since a linear grammar G can have sentential forms with more than one variable in $SenF\Pi_1(G)$, $SenF\Sigma_2(G), ...$, we define the ω-set as a union of all odd sets:

$$SenF\Sigma_\omega(G) := \bigcup_{k \text{ odd}} SenF\Sigma_k(G).$$

Alternating linear languages can be defined as:

$$L\Sigma_k(G) := SenF\Sigma_k(G) \cap \Sigma^*$$
$$LIN\Sigma_{log^k n} :=$$
$$\{L \mid \text{ there is an alternating linear grammar } G \text{ with } L\Sigma_{2O(log^k n)+1}(G) := L\}$$
$$LIN\Sigma_\omega := \{L \mid \text{ there is an alternating linear grammar } G \text{ with } L\Sigma_\omega(G) := L\}$$

The corresponding grammar for SAC^k is an alternating linear grammar with no recursion in the universal productions. Then we have:

Theorem 7.1 $LOG(LIN\Sigma_\omega) = P$, $LOG(LIN\Sigma_k) = NL$ *for* $k \geq 1$.

The first part coincides with $LOG(linearACFL) = P$ according to [CT90], the proof is equivalent to the proof of the next theorem.
In the case of bounded alternation we get further characterizations of AC^k and SAC^k.

Theorem 7.2 $LOG(LIN\Sigma_{log^k n}) = AC^k$

Proof:
'$\subseteq$': In analogy to the proof of in [LR91], there are only linear many sentential forms.
'$\supseteq$': We define the following complete language $CFEM_k := \{w^R \& w \mid w \in CFE_k\}$. $CFEM_k$ is complete for AC^k, in the same way as CFE_k. We can construct an alternating linear grammar G for $CFEM_k$, which simulates the work of the empty alternating push-down automaton in Lemma 6.1, where a path from the start-configuration to an accepting configuration corresponds to a reduction of a word to the start-symbol. The comparison of two (unary) encodings, which was done with the push-down store is now made by a simultaneous reduction of two parts in the two halfs of the word. Hereby the simulation always changes to the other half, when the next encoding has to be found. The formal definition for G is:

$$(\{S, F, T, M\}, \{S, F, T, C, C'\}, \{S, F, M, A, A', B, B'\}, \Sigma \cup \{\&, \#, \%, \&\}, P_e, P_a, S, F)$$

with

$$\begin{array}{lll} P_e := \{ & S \to T \mid M, & \text{and } P_a := \{ \quad M \to \&, A \to \&, A' \to \&, \\ & M \to xMx \mid x \in \Sigma \cup \{\#, \%, \&\}, & \qquad A' \to \#C, \\ & T \to C \mid C' \mid Tx \mid xT \mid x \in \Sigma \cup \{\#, \%, \&\} & \qquad C \to zC \mid z \in \Sigma \cup \{\#\}, \\ & A \to Ax \mid x \in \Sigma \cup \{\#, \%, \&\}, & \qquad A \to \#C', \\ & B \to A\& \mid yBy \mid y \in \Sigma, & \qquad C' \to C'z \mid z \in \Sigma \cup \{\#\}\} \\ & C \to \%B, & \\ & A' \to xA' \mid x \in \Sigma \cup \{\#, \%, \&\}, & \\ & B' \to \&A' \mid yB'y \mid y \in \Sigma, & \\ & C' \to \%B'\} & \end{array}$$

■

Finally, let us mention some relations to automata. The machine model corresponding to the linear languages is the *one-turn* push-down automaton, which starts its computations in a 'push'-mode, where no symbols may be popped from the stack and then enters a final 'pop'-mode, in which no more symbols may be pushed onto the stack. Unfortunately, a closer look at the constructions in [Rei89] reveals the Σ_k^p-completeness of

$1-A\Sigma_{k+1}PDA_{1-TURN}$ i.e., it shows a 'strong' behavior. To come to an adequate machine characterization of alternation, Chen and Toda defined in [CT90] a state-free alternating pushdown automata, which is forced to pop a symbol in every step after entering the 'pop'-mode. On the other hand, the 1-turn restriction is very severe when considering empty alternating push-down automata, as their accepted languages seem to coincide with the Boolean closure of linear languages, which is contained in $NLOG$.

Acknowledgement Thanks to Carsten Damm, Volker Diekert, Klaus-Jörn Lange and Peter Rossmanith for many interesting discussions and helpful remarks.

References

[AJ93] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107:3–30, 1993.

[BCD+88] A. Borodin, S. A. Cook, P. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of complementation via inductive counting. In *Proc. of 3d Structure*, 1988.

[BDHM91] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace-MOD-classes. In C. Choffrut and M. Jantzen, editors, *Proc. of 8th STACS*, number 480 in LNCS, Hamburg, Germany, February 1991. Springer.

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

[CT90] Z.-Z. Chen and S. Toda. Grammatical characterizations of P and PSPACE. *Transactions of the IEICE*, E 73(9), September 1990.

[Dam90] C. Damm. Problems complete for $\oplus$L. In J. Dassow and J. Kelemen, editors, *Proc. of 6th IMYCS*, number 464 in LNCS, pages 214–224. Springer, 1990.

[GNW90] T. Gundermann, N.A. Nasser, and G. Wechsung. A survey on counting classes. In *Proc. of 5th Structure*, pages 140–153, 1990.

[HHK91] T. Hofmeister, W. Hohberg, and S. Köhling. Some notes on threshold circuits, and multiplication in depth 4. In L. Budach, editor, *Proc. of 8th FCT*, number 529 in LNCS, pages 230–239, Gosen, Germany, September 1991. Springer.

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

[JK89] B. Jenner and B. Kirsig. *Alternierung und Logarithmischer Platz*. Dissertation, Universität Hamburg, 1989.

[Lan89] K.-J. Lange. Complexity theory and formal languages. In *Proc. of 5th IMYCS*, number 381 in LNCS, pages 19–36. Springer, 1989.

[LLS84] R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13:135–155, February 1984.

[LR90a] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In B. Rovan, editor, *Proc. of 15th MFCS*, number 452 in LNCS, pages 399–406, Banská Bystrica, Czechoslovakia, August 1990. Springer.

[LR90b] K.-J. Lange and P. Rossmanith. *Two Results on Unambiguous Circuits*. SFB-Bericht 342/3/90 A, I9006, Institut für Informatik, Technische Universität München, 1990.

[LR91] K.-J. Lange and K. Reinhardt. *Empty alternation*. Manuscript, 1991.

[LSL84] R. Ladner, L. Stockmeyer, and R. Lipton. Alternation bounded auxiliary pushdown automata. *Information and Control*, 62:93–108, 1984.

[Mor89] E. Moriya. A grammatical characterization of alternating push-down automata. *Theoret. Comput. Sci.*, 67:75–85, 1989.

[NR91] I. Niepel and P. Rossmanith. Uniform circuits and exclusive read PRAMs. In S. Biswas and K. V. Nori, editors, *Proceedings of the 11th Conference on Foundations of Software Technology and Theory of Computer Science*, number 560 in LNCS, pages 307–318, New Delhi, India, December 1991. Springer.

[NR92] R. Niedermeier and P. Rossmanith. Unambiguous simulations of auxiliary pushdown automata and circuits. In I. Simon, editor, *Proceedings of the 1st Symposium on Latin American Theoretical Informatics*, number 583 in LNCS, pages 387–400, São Paulo, Brazil, April 1992. Springer.

[Rei89] K. Reinhardt. *Hierarchien mit alternierenden Kellerautomaten, alternierenden Grammatiken und finiten Transducern*. Diplomarbeit, Universität Stuttgart, Breitwiesenstr. 22, D-70565 Stuttgart, September 1989.

[Rei90] K. Reinhardt. Hierarchies over the context-free languages. In J. Dassow and J. Kelemen, editors, *Proc. of 6th IMYCS*, number 464 in LNCS, pages 214–224. Springer, 1990.

[Ros91] P. Rossmanith. The owner concept for PRAMs. In C. Choffrut and M. Jantzen, editors, *Proc. of 8th STACS*, number 480 in LNCS, pages 172–183, Hamburg, Germany, February 1991. Springer.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[Sud78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.

[SV84] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, May 1984.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Ven88] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. In K.V. Nori and S. Kumar, editors, *Proceedings of the 8th Conference on Foundations of Software Technology and Theory of Computer Science*, number 338 in LNCS, pages 175–192, Pune, India, December 1988. Springer.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proc. of 6th Structure*, pages 270–285, 1991.

[Wag88] K. Wagner. Bounded query computation. In *Proc. of 3rd Structure*, pages 260–277, 1988.