Empty Alternation*

Klaus-Jörn Lange
Institut für Informatik, TU München
Arcisstr.21, D-8000 München-2
e-mail: lange@informatik.tu-muenchen.dbp.de

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

Abstract

We introduce the notion of *empty alternation* by investigating alternating automata which are restricted to empty their storage exept for a logarithmically bounded tape before making an alternating transition. In particular, we consider the cases when the depth of alternation is bounded by a constant or a polylogarithmic function. In this way we get new characterizations of classes like $P^{NP[log]}$, AC^k, and SAC^k. Finally, we exhibit characterizations of empty alternation by formal language concepts.

Topics: Computational complexity, automata and formal laguages, theory of parallel and distributed computation.

1 Introduction

Alternation as introduced by Chandra, Kozen, and Stockmeyer in [CKS81] proved to be a very fruitful and versatile parallel concept in complexity theory. It has been combined with both automata and grammar models. The language families described by alternating devices coincide with sequential time or space classes.
Bounding the depth of alternation, i.e. bounding the number of transitions between existential and universal configurations within each computation path, by a constant, results in characterizations of several well-known hierarchies defined by iterated relativizations[1] of sequential time and space. Let us review these relations for logarithmically space bounded Turing machines, polynomially time bounded Turing machines, and for the intermediate model of polynomially time bounded auxilliary push-down automata:

Logarithmic Space: There are two types of relativizations of nondeterministic logarithmic space, that of Ladner and Lynch (see [**?**]) and that of Ruzzo, Simon, and Tompa (see [**?**]), which is more restricted by prohibiting nondeterministic oracle queries. While the former leads to the Polynomial Hierarchy of Meyer and Stockmeyer (introduced in [**?**]), the many-one version of the later one characterizes the "late" Logarithmic

*This research was supported by DFG-SFB 342, Teilprojekt A4 "KLARA".

[1] Starting from a complexity class$X$, e.g.: NP, we define levels of a hierarchy by setting $\Sigma_1^X = X$ and $\Sigma_{i+1}^X = X^{\Sigma_i^X}$

Alternation Hierarchy (see [**?**]).[2] Thus, in the case of logarithmically space bounded computations, alternation is related to the weaker kind of relativization.

Polynomial Time: The two types of relativization mentioned above coincide in the case of nondeterministic polynomially time bounded computations and lead to the Polynomial Hierarchy of Meyer and Stockmeyer. In [**?**] Jenner introduced the Deterministic Polynomial Hierarchy by iterating restricted relativizations of deterministic polynomial time equipped with an nondeterministic polynomial time oracle. This hierarchy does not seem to coincide with the Polynomial Hierarchy but rather shows some similarities to the Logarithmic Alternation Hierarchy - in particular, its collapse. If we look at alternating polynomial time, bounding the depth of alternation by a constant leads to the Polynomial Hierarchy ([Sto76], [Wra76]). We see that in the case of polynomially time bounded computations, alternation is related to the stronger kind of relativization and the corresponding hierarchy.

Polynomial Time of Auxilliary Push-Down Automata: Finally, let us consider polynomially time bounded two-way push-down automata which are augmented by a logarithmically space bounded working tape (see [Coo71], [Sud78]). Bounding the depth of alternation of this device by a constant again leads to the Polynomial Hierarchy ,but (as in the Ladner-Lynch case) shifted by one level ([**?**], [**?**]). It is possible to introduce both a Ladner-Lynch-like and a Ruzzo-Simon-Tompa-like relativization of this device. In the later case not only nondeterminism but also access to the push-down store has to be prohibited during the generation of oracle queries. Here, again, the Ladner-Lynch approach would lead to the Polynomial Hierarchy, while the corresponding Ruzzo-Simon-Tompa hierarchy would behave like the Logarithmic Alternation Hierarchy, in particular it would collapse on its first level down to $LOG(CFL)$, the class of problems reducible to context-free languages.

Thus, we see that in all three cases we have two kinds of hierarchies, a stronger one which coincides with the Polynomial Hierarchy and a weaker one which collapses on a low level. Unless the considered type of storage is just a logarithmically space bounded working tape, alternation is related to the stronger hierarchy.
In this paper we introduce a new kind of restricted alternation, which we call *Empty Alternation*. It is characterized by the property that in moments of alternation all tapes or stacks have to be empty except for one working tape of logarithmic size. While "full" alternation usually corresponds to the stronger hierarchy, Empty Alternation will correspond to the weaker one. Thus, the fact that full alternation and Empty Alternation coincide if there is no additional storage but a logspace tape explains the correspondence of logspace alternation to the Ruzzo-Simon-Tompa Relativization.
In section 2 we will give a short overview on complexity classes obtained by bounding depths of alternations by constants or slowly growing functions. In section 3 the notion of Empty Alternation is introduced and related to full alternation, yielding new characterizations of well-known classes like AC^k, SAC^k, or $\Theta_2^P = L^P$. Finally, we will investigate in section

[2] Wagner introduced in [**?**] a slightly more powerfull type of alternation which characterizes the likewise "late" Logarithmic Oracle Hierarchy defined by the Turing version of the relativization of Ruzzo, Simon, and Tompa.

4 relations of Empty Alternation to formal languages. In particular, we will consider the context-free and the linear context-free case.

2 Alternation

We assume the reader to be familiar with the basic notions and results of complexity theory as they are contained in [HoUl79] or [**?**] and [**?**]. In particular, the sets recognizable in logarithmic space, nondeterministic logarithmic space, polynomial time, nondeterministic polynomial time, and polynomial space are denoted by *LOG, NLOG, P, NP,* and *PSPACE.* Let $LOG(X)$ denote the class of all languages reducible to a class X. By $NAuxPDA_{pt}$ and $DAuxPDA_{pt}$ we denote the classes of languages recognizable by nondeterministic and deterministic auxilliary push-down automata (see [Coo71]) in polynomial time. In addition, we use the STA-notation: $STA(f, g, h)$ denotes the set of all languages recognizable by alternating Turing machines, which are $f(n)$ space-bounded, $g(n)$ time-bounded, and make no more than $h(n) - 1$ alternations. Thus, $h(n) = 1$ covers nondeterministic languages and by convention $h(n) = 0$ denotes the deterministic case. In the following we will consider alternating logspace machines, push-down-automata (both with and without polynomial time bound), and polynomially time or space bounded Turing machines with two-way input and bounded depth of alternation, which are allways equipped with a logarithmically space bounded working tape. This, of course, makes no sense (i.e. does not increase computational power) in the case of polynomially time or space bounded Turing machines. But it will be useful when introducing empty alternation. Thus, for $X \in \{LOG, PDA_{pt}, PDA, P, PSPACE\}$ and a function g, where we admit the cases that g is a constant or that g is unbounded which will be indicated by the symbol ω, let $A\Sigma_g^{log}X$ denote the set of all languages recognized by alternating X-machines augmented with a logspace working tape, which make $g(n) - 1$ alternations.
Characterizations of these classes are collected in table **??**. Here, the result $STA(pol, -, pol) = PSPACE$ in Column 5 is due to the result $STA(f, -, g) \subseteq DSPACE(f(f + g))$ for $f(n) \geq log(n)$ of Borodin (cited according to [CKS81]). The cases $A\Sigma_{log^k}^{log}PDA$ and $A\Sigma_{log^k}^{log}PDA_{pt}$ will be settled by the following propositions. The remainig results may be found in [LSL84], [CKS81], [**?**], [Sud78], [**?**], and [Sze88].
The Auxiliary PDA Hierarchy Theorem from [LSL84] can be enhanced in the following way:

Theorem 2.1 *For $s(n) \geq log(n)$ and $a(n) \geq 0$ computable within space $s(n)$ it holds:*

$$\bigcup_c A\Sigma_{a(n)}SPACE(2^{cs(n)}) \subseteq A\Sigma_{1+a(n)}^{s(n)}PDA$$

Idea of the proof: For the simulation of a $A\Sigma_{a(n)}SPACE(2^{cs(n)})$ mashine the PDA first pushes its startconfiguration. To test, whether an existential (universal) configuration belongs to an accepting tree, the PDA existentialy (universaly) guesses an universal (existential) configuration onto the pushdown store, alternates and forgs universaly (existentialy) to test recursively whether this configuration belongs to an accepting tree (or whether an accepting configuration is reachable) or the test of (not-) reachability from the last configuration, which can be done by a $A\Sigma_2^{s(n)}PDA$ in the same way as in [LSL84].

Step of alter-nation	without push-down store	with pushdown store		with polynomial tape	
		polynomial timebound	without timebound	polynomial timebound	without timebound
determ.	L	$LOG(DCFL)$ [Sud78]	P		*PSPACE*
$A\Pi_1$		[Bo et al 88]	P [Coo71]	*co–NP*	
$A\Sigma_1$	NL	$LOG(CFL)$ [Sud78]		*NP*	
$A\Pi_2$	[**?**] [Sze88]	*co–NP* [**?**] [LSL84]		Π_2^p	
$A\Sigma_2$		*NP* [**?**]	*PSPACE* [LSL84]	Σ_2^p	
$\vdots$		$\vdots$		$\vdots$	
$A\Pi_k$		Π_{k-1}^p [**?**]		Π_k^p	
$A\Sigma_k$		Σ_{k-1}^p [**?**]		Σ_k^p	[Imm88] [Sze88]
$\vdots$		$\vdots$		$\vdots$	
$A\Sigma_{log(n)}$	AC^1	$\Sigma_{log(n)}^p$ ⟨Corollary **??**⟩		$\Sigma_{log(n)}^p$	
$\vdots$	$\vdots$	$\vdots$		$\vdots$	
$A\Sigma_{log^k(n)}$	AC^k	$\Sigma_{log^k(n)}^p$ ⟨Corollary **??**⟩	⟨Corollary **??**⟩	$\Sigma_{log^k(n)}^p$	[CKS81]
$\vdots$	$\vdots$	$\vdots$		$\vdots$	
$A\Sigma_\omega$	P [CKS81]	*PSPACE* [**?**]	EXPTIME [LSL84]	*PSPACE* [CKS81]	EXPTIME [CKS81]

Table 1: Complexity classes of automata with spacebounded tape

With the help of the *Alternation Bounded Auxiliary PDA Theorem* of [LSL84], which reads

$$A\Sigma_{a(n)}^{s(n)}PDA \subseteq \bigcup_c NSPACE(a(n)2^{cs(n)})$$

for $s(n) \geq log(n)$, we get

Corollary 2.1 $A\Sigma_{log^k}^{log}PDA = \bigcup_c NSPACE(log^k(n)2^{clog(n)}) = PSPACE$

The result from [**?**] page 60 can be extended to the case of nonconstant depth of alternation as follows:

Theorem 2.2 *For $s(n) \geq log(n)$ and $a(n) \geq 1$ computable within space $s(n)$ it holds:*

$$\bigcup_c A\Sigma_{a(n)}^{s(n)}TIME(2^{cs(n)}) = \bigcup_c A\Sigma_{1+a(n)}^{s(n)}PDA_{2^{cs(n)}}$$

Proof: '$\subseteq$' In a similar way as in Theorem **??** with the difference, that the test of (not-) reachability from the last configuration, can be done by an $A\Sigma_2^{s(n)}PDA_{2^{cs(n)}}$ (an $A\Pi_2^{s(n)}PDA_{2^{cs(n)}}$) in the same way as in [LSL84] because only $2^{c's(n)}$ configurations has to be guessed and pushed.
'$\supseteq$' A turing mashine simulates the pushdown automaton directly using a turing tape instead of an push down store. If the pushdown automaton alternates, the turing mashine does the same. If the pushdown automaton alternates the $a(n)$-th time (the turing mashine can calculate $a(n)$ and count the alternations), the turing mashine does not alternate, but to decide whether the reached configuration is the root of an accepting partial subtree is equivalent to simulate an $A\Sigma_1^{s(n)}PDA_{2^{cs(n)}}$ rsp. an $A\Pi_1^{s(n)}PDA_{2^{cs(n)}}$, which can be done deterministicaly in $2^{cs(n)}$ time according to [Coo71]. ■

Corollary 2.2 $A\Sigma_{log^k}^{log}POL = A\Sigma_{log^k}^{log}PDA_{pt}$

3 Empty alternation

If we equippe logarithmically space bounded turing mashines with the possibility of bounded alternation, the result is comparatively small: We just reach nondeterministic LOGSPACE. The situation changes completely, if we increase the power of the underlying machine. If we, for example, augment deterministic LOGSPACE machines with auxilliary push-down stores while still keeping polynomial time bounds, this results in LOG(DCFL), a rather small complexity class contained in SC and in NC and thus in P and in POLYLOGSPACE. But now the addition of depth bounded alternation yields the Polynomial Hierarchy ([**?**],[**?**]). If we take a closer look to this phenomenon, we see, that the underlying machine has now the possibility of pushing polynomially many bits in an existential or universal way onto the push-down store. These bits are then popped, i.e. read one-way, and evaluated again with the help of bounded alternation. This phenomenon and its explanation led us to the concept of empty alternation: We augment machines with several storage types and then add alternation under the restriction that in moments of alternation, that is during transitons between existential and universal configurations, all auxilliary memories are empty and all transferred information is contained in the state and on a logarithmically spacebounded working-tape.
In the following, for $X \in \{LOG, PDA_{pt}, PDA, P, PSPACE\}$ and a function g, where we again admit the cases that g is a constant or that g is unbounded, which will be indicated by the symbol ω, let $EA\Sigma_g^{log}X$ denote the set of all languages recognized by LOGSPACE turing mashines augmented with storage of type X, which make $g(n)-1$ empty alternations. The main results of this chapter are collected in table **??**, which is the 'empty' analogon of table **??**.

3.1 Empty Alternation and push-down automata

In this section we study the concept of empty alternation for machines equipped with an additional LOGSPACE tape (i.e. for 'unaugmented' machines), with one push-down store while maintaining a poynomial time bound, and with one push-down store without any restriction of the running time. Obviously, for unaugmented machines empty alternation coincides with (full) alternation, which reads $EA\Sigma_g^{log}LOG = A\Sigma_g^{log}LOG$ for

<table>
<tr><td rowspan="2">Step of alter-nation</td><td rowspan="2">without push-down store</td><td colspan="2">with pushdown store</td><td colspan="2">with polynomial tape</td></tr>
<tr><td>polynomial timebound</td><td>without timebound</td><td>polynomial timebound</td><td>without timebound</td></tr>
<tr><td>determ.</td><td>L</td><td>$LOG(DCFL)$ [Sud78]</td><td colspan="2">P</td><td>$PSPACE$</td></tr>
<tr><td>$A\Pi_1$</td><td rowspan="10">NL</td><td rowspan="3">[Bo et al 88]
$LOG(CFL)$ [Sud78]</td><td rowspan="10">P [Coo71]</td><td>co–NP</td><td rowspan="10"></td></tr>
<tr><td>$A\Sigma_1$</td><td>NP</td></tr>
<tr><td>$EA\Pi_2$</td><td rowspan="8"></td></tr>
<tr><td>$EA\Sigma_2$</td><td rowspan="2">[Sze88]</td></tr>
<tr><td>$\vdots$</td></tr>
<tr><td>$EA\Pi_k$</td><td rowspan="5"></td></tr>
<tr><td>$EA\Sigma_k$</td><td rowspan="3">⟨Theorem **??**⟩</td></tr>
<tr><td>$\vdots$</td></tr>
<tr><td>$EA\Sigma_{log(n)}$</td><td colspan="2">AC^1 ⟨Theorem **??**⟩</td></tr>
<tr><td>$\vdots$</td><td colspan="2">$\vdots$</td><td colspan="2"></td></tr>
<tr><td>$EA\Sigma_{log^k(n)}$</td><td colspan="2">AC^k ⟨Theorem **??**⟩</td><td colspan="2">Θ_2^p</td></tr>
<tr><td>$\vdots$</td><td colspan="2">$\vdots$</td><td colspan="2"></td></tr>
<tr><td>$EA\Sigma_\omega$</td><td colspan="2">[CKS81] P ⟨Corollary **??**⟩</td><td>⟨Corollary **??**⟩</td><td>⟨Theorem **??**⟩</td><td>⟨Theorem **??**⟩</td></tr>
</table>

Table 2: Complexity classes of automata with logarithmic spacebounded tape and empty alternation

$g \in \{O(1), log^k n, \omega\}$. Thus, $EA\Sigma_\omega^{log} LOG = P$. The following result shows, that this relation holds even for machines augmented with an unrestricted push-down store.

Theorem 3.1 $EA\Sigma_\omega^{log} PDA \subseteq P$

Proof: According to [Coo71] it can be decided in polynomial time for two configurations K_1, K_2 with empty pushdown store, whether $K_1 \stackrel{*}{\underset{M}{\vdash}} K_2$ without alternation. If M without loss of generality only stops with empty pushdown store, then for input x it can be calculated recursively for all such configurations in the calculation of $M(x)$, whether there is a partial accepting subtree under that configuration. Because this has to be calculated once for each of the polynomially many configurations, this can be done in polynomial time. ■

Contrast this with $A\Sigma_\omega^{log} PDA = EXPTIME$ in [LSL84]. By Cooks characterization of P by auxilliary push-down automata in [Coo71] Theorem **??** yields

Corollary 3.1 $EA\Sigma_g^{log} PDA = P$ *for* $g \in \{O(1), log^k, \omega\}$

Another consequence of Theorem **??** follows from $P = EA\Sigma_\omega^{log} LOG \subseteq EA\Sigma_\omega^{log} PDA_{pt}$:

Corollary 3.2 $EA\Sigma_\omega^{log} PDA_{pt} = P$

Finaly the nature of polylogarithmically bounded empty alternation of polynomially time bounded pushdown automata is characterized by

Theorem 3.2
$EA\Sigma^{log}_{log^k n}PDA_{pt} = AC^k$ *for* $k \geq 1$

Proof: $AC^k = EA\Sigma^{log}_{log^k n}LOG \subseteq EA\Sigma^{log}_{log^k n}PDA_{pt} : \surd$ The converse follows from Lemma **??** below. ■

This result indicates the comparatively small computational power of Empty Alternation when dealing with polynomially time bounded auxilliary push-down automata: the addition of a push-down store does not increase the power of a logspace machine as long as the depth of alternation is at least logarithmically growing.
In an obvious way it is posible to introduce *semiunbounded* empty alternation (compare with [Ven87]) which yields classes named $SEA\Sigma^{log}_{g}X$. For the semiunbounded circuit class SAC^k we refer to [Ven87].

Theorem 3.3
$SEA\Sigma^{log}_{log^k n}PDA_{pt} = SAC^k$ *for* $k \geq 1$

Proof: $SEA\Sigma^{log}_{log^k n}LOG \subseteq SEA\Sigma^{log}_{log^k n}PDA_{pt} : \surd$ The other direction follows from Lemma **??** below. ■

Lemma 3.1 $EA\Sigma^{log}_{log^k n}PDA_{pt} \subseteq AC^k$ *and* $SEA\Sigma^{log}_{log^k n}PDA_{pt} \subseteq SAC^k$

Proof: An empty alternating push-down automaton with $h\,log(n)$ spacebounded tape can be simulated by an AC^k (SAC^k) circuit which calculates with $O(|x|^{2h+2})$ subcircuits for every pair of surface-konfigurations $\langle K_1, K_2\rangle$ with empty push-down-store whether K_2 is reachable from K_1 without any alternation. Since this can be done in $LOG(CFL)$ or $LOG(co{-}CFL)$ it can also be done by a SAC^1-circuit because of [Bo et al 88]. Then the circuit recursively calculates in each level j of the log^k levels for every surface-configuration K_i the bit $c_{i,j}$, which is 1, if it has an accepting tree of depth j. For an accepting (rejecting) node, this is 1 (0), for an existential node

$$c_{i,j} = \bigvee_l (c_{l,j-1} \wedge \langle K_l, K_i\rangle)$$

and for an universal configuration

$$c_{i,j} = \bigwedge_l (c_{l,j-1} \vee \overline{\langle K_l, K_i\rangle}),$$

in case of a semiunbounded push-down automaton there are only finitely many l's in a conjunction, so the whole construction results in a SAC^k circuit. ■

3.2 Empty Alternation and Turing-Tapes

If we consider machines with two or more auxilliary push-down stores, it is easy to see, that from the aspect of compexity these are equivalent to turing tapes. That is why we will consider empty alternation of polynomial time and of polynomial space in this subsection. In the case of polynomial time we will get a characterization of the class $\Theta^p_2 := L^{NP}$. This class was named by Wagner, who gave several representations of the classes $\Theta^p_{k+1} := L^{\Sigma^p_k}$. In particular he showed:

Theorem 3.4 *[?]:* $\Theta^p_{k+1} = P^{\Sigma^p_k[log]}$ *for each* k

which reads for $k = 1$:

Theorem 3.5 *[?]:* $\Theta^p_2 = P^{NP[log]}$

Here $P^{A[log]}$ referes to classes defined by polynomial time bounded oracle machines, which are allowed to ask at most $O(log(n))$ queries.

Theorem 3.6 $EA\Sigma^{log}_2 P = EA\Sigma^{log}_\omega P = \Theta^p_2$

Proof: $EA\Sigma^{log}_2 P \subseteq EA\Sigma^{log}_\omega P$ is trivial, the rest follows from Theorem **??**, Lemma **??** and Lemma **??**, below. ■

Lemma 3.2 $EA\Sigma^{log}_\omega P \subseteq L^{NP}$

Proof: Let $K(x)$ be the set of those configurations of an $EA\Sigma^{log}_\omega P$-mashine M on input x, where the not logarithmic spacebounded tape is empty. Thus $|K(x)|$ is bounded by a polynomial. We consider the language

$$L_1 := \{(x, K_1, K_2) \mid K_1, K_2 \in K(x) \text{ and } K_1 \text{ leads to } K_2 \text{ without any alternation } \}.$$

Obviously, we have $L_1 \in NP$. Now, the number $n(x) := |L_1 \cap (\{x\} \times K(x) \times K(x))|$ is polynomially bounded and can be computed by a logspace machine with oracle L_1. We define

$$L_2 := \{(x, n) \mid x \in L(M) \text{ or } n < n(x)\}.$$

First we observe $L_2 \in NP$: The NP-mashine N for this oracle language can guess, verify and store on input (x, n) all n different pairs (K_1, K_2) with $K_1 \vdash^x_M K_2$. If $n = n(x)$, then the list is complete, otherwise N rejects. Now it guesses which configurations belong to the accepting partial tree T and tests for all universal configurations K_a in T, whether all K with $K_a \vdash^x_M K$ are in T or whether K_a is accepting and for all existential configurations K_e in T, whether there is a K with $K_a \vdash^x_M K$ in T or whether K_e is accepting. We assume w.l.o.g., that the tape, which is not logarithmic spacebounded is empty, if the machine accepts or stops.
On the other hand, we see, that $L(M)$ can be recognized by a logspace Machine with oracle L_2, if the number $n(x)$ is known. Hence, by building the disjoint (i.e. marked) union of L_1 and L_2 as an oracle, we can accept $L(M)$ by an L^{NP}-machine. ■

Lemma 3.3 $P^{NP[log]} \subseteq EA\Sigma^{log}_2 P$

Proof: Let $L \in P^{NP[log]}$ by an oracle machine M with oracle SAT. An $EA\Sigma^{log}_2 P$-machine A simulates M twice:
In the first simulation A starts in an existential state and simulates the deterministic steps of M. If M asks the i-th oracle question '$v_i \in SAT$?', then A guesses the answer and stores it as the i-th bit on the logarithmic spacebounded tape. If the answer 'Yes' is guessed, then A simulates the NP-mashine B for SAT on v_i and rejects, if B rejects. If the answer 'No' is guessed, the verification is postponed to the second phase of alternation. Then A

continues the simulation of M. If M accepts, then A alternates into an universal state and starts the second phase of the simulation by simulating again the deterministic steps of M. If M asks the i-th oracle question '$v_i \in SAT$?', then A looks up the answer from the logarithmic spacebounded tape. If the answer is 'No', then A simulates universally the $co-NP$-mashine C for $UNSAT$ on v_i and rejects, if C rejects. Then A continues the simulation of M. If M accepts, then A accepts. ■

Finally, we shortly consider the case of polynomial space. By a result of Borodin (cited in [CKS81]) we have $A\Sigma_{pol}^{PSPACE} = PSPACE$. While $A\Sigma_{\omega}^{PSPACE} = EXPTIME$, empty alternation does not lead beyond $PSPACE$:

Theorem 3.7 $EA\Sigma_{\omega}^{log}PSPACE = PSPACE$

Proof: It can be decided with polynomial space for two configurations K_1, K_2 , whether $K_1 \overset{*}{\underset{M}{\vdash}} K_2$ without alternation. If M without loss of generality only stops with all auxilliary tapes empty, then for input x it can be calculated recursively for all such configurations in the calculation of $M(x)$, whether there is a partial accepting subtree under that configuration. Because this has to be calculated once for each of the polynomially many configurations, this can be done in polynomial space. ■

4 Formal language concepts and empty alternation

So far, we only investigated two-way automata augmented with a logarithmically space bounded working tape. Now, we will consider automata which read their input only once and which are not equipped with an additional logspace working tape. Obviously, this is not a restriction for polynomially time or polynomially space bounded Turing machines. Thus, it remaines to treat the case of case of push-down automata. Full alternation of push-down automata, that is alternating push-down automata which may alternate with an nonempty push-down store, were investigated in [**?**], [**?**] and [**?**]. In the first part of this section we will consider Empty Alternation of unaugmented one-way push-down automata, that is of alternating push-down automata which have to empty their push-down store before any alternation. Searching for an equivalent grammar system, we will introduce in the second part of this section the notion of an *alternating linear context-free grammar*.

4.1 Empty alternating pushdown automata

In Theorem **??**, we saw $EA\Sigma_{log^k n}^{log}PDA_{pt} = EA\Sigma_{log^k n}^{log}LOG$ for each k. That is, the additional push-down store does not increase the computational power, as long as we keep a polynomial time bound. In this subsection we will see, that the other direction holds true in the sense, that empty alternating push-down automata without two-way input and without logspace working tape generate languages complete for $EA\Sigma_{a(n)}^{log}PDA_{pt}$. Thus we will generalize the equations $NAuxPDA_{pt} = LOG(CFL)$ and $DAuxPDA_{pt} = LOG(DCFL)$ of Sudborough in [Sud78]. This result may be interpreted in the sense that in $EA\Sigma_{a(n)}^{log}PDA_{pt}$-automata an one-way push-down part may be separated from a two-way logspace part. This decomposition is also possible for fully alternating pushdown automata as shown in [**?**] and [**?**], slightly generalizing a result of [**?**]. It could be remarked here that this relation does not seem to hold in general: e.g. unambiguous automata seemingly do not allow for decompositions like that.

Definition An *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.

The definitions of alternating push-down automata differ concerning the role of λ-transitions. Fortunately these things do not matter when considering empty alternation.
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 treat the cases of either unbounded or constant alternation depth, only. To cope with that problem, we apply a depth bound not within the automaton, but instead within the language:

Definition Let A be an 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

- The root is the existential start configuration $\sigma_A(x)$,
- for every existential configuration c in the tree, there is a d with $c \vdash_A d$ in the tree,
- for every universal configuration c in the tree, all d's with $c \vdash_A d$ are in the tree,
- the leaves of the tree are accepting,
- alternations from an existential to an universal configuration or vice versa are *only* allowed, if the push-down-store is *empty* (the push-downstore is regarded as empty, if only the bottom symbol \$ is on the push-down store.[3]), and
- there are at most $a(n) - 1$ alternations on every path on the tree.

It should be remarked here, that the $EA\Sigma_{a(n)}$ language of A is a subset of L(A) which is the $EA\Sigma_\omega$ language of A.
The set of all $EA\Sigma_{a(n)}$-languages of one-way empty alternating push-down automata is denoted by $1\text{–}EA\Sigma_{a(n)}PDA$. $1\text{–}SEA\Sigma_{a(n)}PDA$ is the set of languages which can be recognized by an 1-way-empty-semiunbouded-alternating- Σ_k-push-down automaton, which is only allowed to make finitely many steps in universal states before alternating into an existential state.

Using the result of [Bo et al 88], it is easy to see, that we then have

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

Theorem 4.1 *$LOG(1\text{–}EA\Sigma_k PDA) = LOG(CFL)$ for each k.*

Concerning bounded depth of alternation we get:

Theorem 4.2

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

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 of Lemma **??**, it suffices to exhibit some $(S)AC^k$-complete set, which is genereated as $(S)EA\Sigma_{log^k n}$-language by an one-way empty alternating pushdown automaton. This is done in Lemma **??** and Lemma **??** below. ■

Lemma 4.1 *$AC^k \subseteq LOG(1\text{–}EA\Sigma_{log^k n} PDA)$*

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 $A \to B_1 B_2..., B_i \in P$ 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: Let L be accepted by an AC^k circuit family. Using the unifying machine of this family, we can construct a logspace 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 $A \to B_1...B_i$ has to be in the grammar. The productions have to be in a monotone order, that 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 pushdown automaton M recognizing CFE_ω works as follows: M chooses existentialy 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. ■

Lemma 4.2 $SAC^k \subseteq LOG(1\text{–}SEA\Sigma_{log^k n}PDA)$

Proof: 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 than one of Lemma **??**, which makes only one step in an universal state by decing 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 to the restriction to 2 variables on the right hand side of the productions of the simulated grammar.

■

The results of Theorem **??** might be interpreted as a kind of computational equivalence between two-way input and a logarithmically space bounded working tape on the one hand, and one-way input and a push-down store on the other hand in the case of depth of Empty Alternation which grows at least logarithmically.

4.2 Alternating linear Grammars

We now consider alternating grammars. There are several possibilities to define alternating context free grammars. So for the alternating context free grammars in [**?**] we have $LOG(ACFL^{left}_{\lambda-free}) = PSPACE$,but for the alternating (even λ-free) alternating context free grammars in [**?**] $LOG(CFL\Sigma_\omega) = EXPTIME$ holds. In any case there are close relations to alternating push-down automata and corresponding complexity classes. On the other hand, 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 auxilliary storage, which implies in this case the coincidence of alternation and empty alternation. (Compare this with the $NSPACE(log(n))$-completeness of the class of linear languages, and with the fact that for logspace alternation and empty alternation coincide). 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 [**?**] reveals the Σ^p_k-completeness of $1-A\Sigma_{k+1}PDA_{1-TURN}$ i.e., it shows a 'strong' behaviour. To come to an adequate machine characterization of alternation, Chen and Toda defined in [**?**] 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 $NSPACE(log(n))$. We now turn to alternating linear grammars:

Definition According to [**?**] and [**?**] we define an alternating grammar G to be an 8-tuple

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

and for $k > 0$ we define:

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

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

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

Alternating linear languages are now defined by:

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

Then we have:

Theorem 4.3 *1.* $LOG(LIN\Sigma_\omega) = P$

2. $LOG(LIN\Sigma_k) = NL$ *for* $k \geq 1$.

The first part coincides with the equation $LOG(linearACFL) = P$ of [**?**] and is a consequence of the proof of the next theorem.
In the case of bounded alternation we get further characteriztions of AC^k.

Theorem 4.4 $LOG(LIN\Sigma_{log^k n}) = AC^k$

Proof:
'$\subseteq$': In analogy to the proof of Lemma **??**, 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 for $CFEM_k$, which simulates the work of the empty alternating pushdown automaton in Lemma **??**, 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 simultanious 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. ■

It is possible to characterize SAC^k classes by alternating linear grammars which have no recursion in their universal productions. The interested reader may find the explicit notations and constructions in [**?**].

Discussion and Open Questions

We introduced the concept of *Empty Alternation* as a restriction of the usual 'full' alternation and exhibited close connections to questions of how to relativize complexity classes and about the collapses of hierarchies. As a result new representations of many well-known complexity classes have been obtained. Since alternation is a very powerful mechanism, it seems reasonable not only to restrict the concept itself, but also the device, it is applied to. In this way, relations between formal languages and complexity could be generalized. This leaves open to investigate these relations with respect to other models of formal language theory. First candidates should here be all types of stack automata, since the relations between their deterministic, nondeterministic, (fully) alternating, and auxilliary versions show a pattern very similar to that of push-down automata. Another interesting question would be to determine both an alternation type and an automata model which together characterize the NC^k classes not as time classes, but directly by the depth of alternation.

Acknowledgement We thank Peter Rossmanith, Werner Ebinger and Volker Diekert for many helpfull remarks, and Prof. Dr. W. Knödel, who made this joint work possible.

References

[BDG88] J. Balcázar and J. Diáz and J. Gabárro: Structural Complexity Theory I; Springer 1988.

[BDG90] J. Balcázar and J. Diáz and J. Gabárro: Structural Complexity Theory II; Springer 1990.

[Bo et al 88] A.Borodin. S.A. Cook, P.W.Dymond, W.L. Ruzzo, M.Tompa; Two applications of complementation via inductive counting, 3rd Structure in Complexity Theory.

[Bun87] G. Buntrock: On the Robustness of the Polynomial Time Hierarchy, Technische Universität Berlin, Technischer Bericht, Nr.: 87-11,1987.

[CKS81] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer; Alternation, Journ. of the ACM 28,1(1981), 114-133.

[Coo71] S.A. Cook: Characterizations of push-down machines in terms of timebounded computers, Journ. of the ACM 18,1(1971), 4-18

[CT90] Z.-Z. Chen, S. Toda: Grammatical Characterizations of P and PSPACE, transactions of the IEICE, Sep. 1990.

[Imm88] N. Immerman: Nondeterministic space is closed under complementation, SIAM Journ. Comput. 15, 5 (1988), 935-938.

[Jen87] B. Jenner: Die Deterministische Polynomielle Hierarchie, Diplomarbeit, Universität Hamburg, 1987.

[JK89a] B. Jenner, B. Kirsig: Characterizing the polynomial hierarchy by alternating auxiliary push-down automata. Theoretical Informatics and Applications, 1989, 87-99.

[JK89b] B. Jenner and B. Kirsig: Alternierung und Logarithmischer Platz,Dissertation, Universität Hamburg,1989.

[LLS84] R E. Ladner, R. J. Lipton, L. J. Stockmeyer: Alternating Pushdown and Stack Automata,SICOMP 13, FEB 1984,135–155.

[LL76] R. Ladner and N. Lynch: Relativization of questions about log space computability, Math. Systems Theory 10(1976),19-32.

[LSL84] R.E. Ladner,L.J.Stockmeyer,R.J. Lipton: Alternation bounded auxiliary pushdown automata, Information and Control 62(1984), 93-108.

[Lan86] K.-J. Lange: Two Characterizations of the Logarithmic Alternation Hierarchy, Proc. of 12th MFCS 233, LNCS, Springer 1986, 518–526.

[MSt72] A.R. Meyer and L.J. Stockmeyer: The equivalence problem for regular expressions with squaring requires exponential time, Proc. of 13th Annual IEEE Symp. Switching and Automata Theory (1972), 125-129.

[Mor89] E. Moriya: A grammatical characterization of alternating push-down automata, TCS 67 (1989) 75-85.

[Rei89] K. Reinhardt: Hierarchien mit alternierenden Kellerautomaten, alternierenden Grammatiken und finiten Transducern, Diplomarbeit, Universität Stuttgart, 1989.

[Rei90] K. Reinhardt: Hierarchies over the context-free Languages, Proc. of 6th IMYCS, LNCS, 464, Springer 1990, 214-224.

[Ruz81] W. Ruzzo: On Uniform Circuit Complexity, JCSS 22 (1981) 365-338.

[RST84] W. Ruzzo and J. Simon and M. Tompa: Space – Bounded hierarchies and probabilistic computations, JCSS 28(1984),216-230.

[Sze88] R. Szelepcsenyi: The Method of forced enumeration for nondeterministic automata, Acta Informatica 26(1988), 96-100.

[Sud78] I.H. Sudborough: On the tape complexity of deterministic context-free languages, Journ. of the ACM 25, 3 (1978),405-414.

[Wag87] K. Wagner: The number of alternations, Universität Augsburg, Technischer Bericht, Nr.: 133, 1987.

[Wag88] K. Wagner: Bounded Query Computation, Proceedings of the 3rd annual Structure 88, 260-277.

[Wra76] C. Wrathall: Complete sets and the polynomial-time hierarchy, Theoret. Comp. Sci. 3 (1976), 23-33.

[Ven87] H. Venkateswaran: Properties that characterize LOGCFL. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, 141-150, New York,May 1987.