Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 299 (2003) 307–326

Theoretical Computer Science

www.elsevier.com/locate/tcs

Alternating and empty alternating auxiliary stack automata☆,☆☆

Markus Holzer[a,*,1], Pierre McKenzie[b]

[a] *Institut für Informatik, Technische Universität München, Arcisstraße 21, D-80290 München, Germany*
[b] *Département d'IRO, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada*

Received 2 March 2001; received in revised form 28 February 2002; accepted 22 March 2002
Communicated by O. Watanabe

Abstract

We consider variants of alternating auxiliary stack automata and characterize their computational power when the number of alternations is bounded by a constant or unlimited. In this way we get new characterizations of **NP**, the polynomial hierarchy, **PSpace**, and bounded query classes like **co-DP** $=\mathbf{NL}^{\langle \mathbf{NP}[1]\rangle}$ and $\Theta_2\mathbf{P}=\mathbf{P}^{\mathbf{NP}[O(\log n)]}$, in a uniform framework.

1. Introduction

An auxiliary pushdown automaton is a resource-bounded Turing machine with a separate resource unbounded pushdown store (**PD**), that is a last-in first-out (LIFO) storage structure, which is manipulated by pushing and popping. Probably, such machines are best known for capturing **P** when their space is logarithmically bounded [4]

☆ This paper is a completely revised and expanded version of a paper presented at the 25th Conference on Mathematical Foundations of Computer Science (MFCS) held in Bratislava, Slovak Republic, August 28–September 1, 2000.

☆☆ Supported by the National Sciences and Engineering Research Council (NSERC) of Canada and by the *Fonds* pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR) of Québec.

* Corresponding author.
E-mail addresses: holzer@informatik.tu-muenchen.de (M. Holzer), mckenzie@iro.umontreal.ca (P. McKenzie).

[1] Part of the work was done while the author was at Département d'I.R.O., Université de Montréal, C.P. 6128, succ. Centre-Ville, Montreal (Québec), H3C 3J7 Canada.

0304-3975/03/$ - see front matter
PII: S0304-3975(02)00326-2

and for capturing the important class LOG(**CFL**) $\subseteq$ **P** when additionally their time is polynomially bounded [21]. These two milestones in reality form part of an extensive list of equally striking characterizations (see [24, pp. 373–379]). For example, a *stack* (**S**) is a pushdown store allowing its interior content (that is, symbols other than the topmost symbol) to be *read* at any time, a *nonerasing stack* (**NES**) is a stack which cannot be popped, and a *checking stack* (**CS**) is a nonerasing stack which forbids any push operation once an interior stack symbol gets read. Cook's seminal result [4] alluded to above, that

$$\textbf{AuxPD-DSpace}(s(n)) = \textbf{AuxPD-NSpace}(s(n)) = \bigcup \textbf{DTime}(2^{c\cdot s(n)})$$

when $s(n) \geqslant \log n$, is in sharp contrast with Ibarra's [9], who proved that

$$\begin{aligned}
\textbf{AuxS-DSpace}(s(n)) = \textbf{AuxS-NSpace}(s(n)) &= \bigcup \textbf{DTime}(2^{2^{c\cdot s(n)}}),\\
\textbf{AuxNES-DSpace}(s(n)) = \textbf{AuxNES-NSpace}(s(n)) &= \bigcup \textbf{DSpace}(2^{c\cdot s(n)}),\\
\textbf{AuxCS-NSpace}(s(n)) &= \bigcup \textbf{NSpace}(2^{c\cdot s(n)}),\\
\textbf{AuxCS-DSpace}(s(n)) &= \textbf{DSpace}(s(n)),
\end{aligned}$$

where unions are over c and our class nomenclature should be clear (or else refer to Section 2).

In the wake of [3], pushdown stores were also added to *alternating* Turing machines [14,15]. Most notably, **AuxPD**-alternating automata were shown strictly more powerful than their deterministic counterparts, and a single alternation level in a $s(n)$ space-bounded **AuxPD**-automaton was shown as powerful as any constant number. In the spirit of Sudborough's LOG(**CFL**) characterization [21], Jenner and Kirsig [11] further used **AuxPD**-alternating automata with simultaneous resource bounds to capture **PH**, the polynomial hierarchy. Subsequently, Lange and Reinhardt [13] helped shed light on why **AuxPD**-alternating automata are so powerful. They introduced a new concept: a machine is *empty alternating* if it only alternates when all its auxiliary memories and all its tapes, except a logarithmic space-bounded part, are empty. They proceeded to show that time-bounded **AuxPD**-empty alternating automata precisely capture the $\mathbf{AC}^k$-hierarchy.

Alternating auxiliary *stack* automata were also investigated. For example, Ladner et al. [14] showed that alternation provably adds power to otherwise nondeterministic **AuxS**-space-bounded automata. Further results in the case of unbounded numbers of alternations are that **AuxS**- and **AuxNES**-space-bounded automata are then equally powerful, i.e., the ability to erase the stack is in fact inessential.

The aim of the present paper is to just about complete the picture afforded by **AuxS**-, by **AuxNES**-, and by **AuxCS**-automata, in the presence of alternation or of empty alternation. We distinguish between bounded and unbounded numbers of alternations, and we consider arbitrary space bounds. In particular, we investigate **AuxCS**-alternating automata, a model overlooked by Ladner et al. [14]. We also completely answer the question posed by Lange and Reinhardt [13] concerning the power of variants of

Table 1
Complexity classes of auxiliary $\log n$ space-bounded alternating automata (shading represents results obtained in this paper), where unions are over k

	Auxiliary alternating Turing machine with storage X								
		PD-		**S-**		**NES-**		**CS-**	
		Time bound		Time bound		Time bound		Time bound	
Alt.	none	Poly	Without	Poly	Without	Poly	Without	Poly	Without
det.	**L**	LOG(**DCFL**) [2]	**P** [1]	**P**	$\bigcup$**DTime**(2^{n^k}) [4]	**P**	**PSpace** [4]	**L**	**L** [4]
Σ_1	**NL**	LOG(**CFL**) [2]		**NP**		**NP**		**NP** [8]	**PSpace** [4]
Σ_2		**NP** [8]	**PSpace** [7]	Σ_2**P**	$\bigcup$**DTime**(2^{n^k}) $\subseteq$ $\cdot$ $\subseteq$ $\bigcup$**NSpace**(2^{n^k})	Σ_2**P**	**PSpace** $\subseteq$ $\cdot$ $\subseteq$ $\bigcup$**NSpace**(2^{n^k})	Σ_2**P** [8]	**PSpace** $\subseteq$ $\cdot$ $\subseteq$ $\bigcup$**NSpace**(2^{n^k})
Σ_3		Σ_2**P** [8]		Σ_3**P**		Σ_3**P**		Σ_3**P** [8]	
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	
Σ_k	[22,23]	Σ_{k-1}**P** [8]		Σ_k**P**		Σ_k**P**		Σ_k**P** [8]	
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	
A	**P** [5]	**PSpace** [8]	$\bigcup$**DTime**(2^{n^k}) [6]	**PSpace**	$\bigcup$**DTime**$(2^{2^{n^k}})$ [6]	**PSpace**	$\bigcup$**DTime**$(2^{2^{n^k}})$ [6]	**PSpace** [8]	$\bigcup$**DTime**$(2^{2^{n^k}})$

empty alternating auxiliary stack automata. More generally, we refine previous characterizations in several directions. For example, to our knowledge nothing was known about **AuxS**-, **AuxNES**-, and **AuxCS**-automata with a constant number of alternations, with running time restrictions, and/or with the feature of empty alternation. We consider these models here. Our precise results and their relationships with former work are depicted in Tables 1 and 2.

The technical depth of our results varies from immediate to more subtle extensions to previous work. Indeed a baggage of techniques has developed in the literature, in the form of a handful of fundamental "tricks" underlying all known simulations (examples: Cook's "surface configuration" trick, Ibarra's and former authors' "pointer to a pointer" trick, Ladner et al.'s tricks to simulate alternation). The difficulty, when analyzing models which combine features for which individual tricks are known, is that some of the tricks are a priori incompatible. A typical example arises with **AuxCS**-empty alternating automata: the standard simulation method used to bound the number of alternations clashes with the checking stack property, because the checking stack and the empty alternation appear to be mutually exclusive.

The paper is organized as follows: the next section contains preliminaries, and in Section 3 we investigate **AuxS**-, **AuxNES**-, and **AuxCS**-alternating automata. Then Section 4 is devoted to empty alternation, and finally we summarize our results and highlight the remaining open questions in Section 5.

2. Definitions

We assume the reader to be familiar with the basics of complexity theory as contained in the book of Balcázar et al. [1] and Hopcroft and Ullman [8]. In particular, $\Sigma_{a(n)}\mathbf{SpaceTime}(s(n),t(n))$ ($\Pi_{a(n)}\mathbf{SpaceTime}(s(n),t(n))$, respectively) denotes the class of all languages accepted by $O(s(n))$ space- and $O(t(n))$ time-bounded alternating Turing machines making no more than $a(n)-1$ alternations starting in an existential (universal, respectively) state. Thus, $a(n)=1$ covers nondeterminism and by convention $a(n)=0$ denotes the deterministic case. For simplicity we write **N** instead of Σ_1 and **D** for Σ_0. Moreover, if the number of alternations is unrestricted, we simply replace $\Sigma_{a(n)}$ by **A**. If we are interested in space and time classes only, we simply write $\mathbf{Space}(s(n))$ and $\mathbf{Time}(t(n))$, respectively, in our notations. In what follows, we assume that whenever we refer to a space bound $s(n)$ we implicitly assume that it is space constructible. We consider

$$\mathbf{L} := \mathbf{DSpace}(\log n) \subseteq \mathbf{NL} := \mathbf{NSpace}(\log n) \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSpace}.$$

In particular, $\Sigma_k\mathbf{P}$ and $\Pi_k\mathbf{P}$, for $k \geqslant 0$, denote the classes of the polynomial hierarchy.

In the following we consider Turing machines with two-way input equipped with an auxiliary pushdown, stack, nonerasing stack, and checking stack storage. A pushdown is a LIFO storage structure, which is manipulated by pushing and popping. The origin of the concept is not clear and is attributed by most to Burks et al. [2] and Newell and Shaw [16]. By *stack storage*, we mean a *stack* (**S**), *nonerasing stack* (**NES**),

Table 2
Complexity classes of auxiliary $\log n$ space-bounded *empty* alternating automata (shading represents results obtained in this paper), where unions are over k

	Auxiliary *empty* alternating Turing machine with storage X								
		PD-		**S-**		**NES-**		**CS-**	
		Time bound		Time bound		Time bound		Time bound	
Alt.	none	Poly	Without	Poly	Without	Poly	Without	Poly	Without
det.	**L**	LOG(**DCFL**) [2]	**P** [1]	**P**	$\bigcup$**DTime**(2^{n^k}) [4]	**P**	**PSpace** [4]	**L**	**L** [4]
$\mathbf{E}\Sigma_1$	**NL**	LOG(**CFL**) [2]		**NP**		**NP**		**NP** [8]	**PSpace** [4]
$\mathbf{E}\Sigma_2$		LOG(**CFL**) [9]	**P** [9]	Θ_2**P**	$\bigcup$**DTime**(2^{n^k})	**co-DP**	**PSpace**	**co-DP**	**PSpace**
$\mathbf{E}\Sigma_3$						Θ_2**P**		Θ_2**P**	
⋮									
$\mathbf{E}\Sigma_k$	[22,23]								
⋮									
EA	**P** [5]	**P** [9]							

or *checking stack* (**CS**) as defined earlier. Originally, stack automata were introduced by Ginsberg et al. [5], nonerasing stack automata appear first in Hopcroft and Ullman [7], and checking stack automata in Greibach [6]. The class of languages accepted by $O(s(n))$ space-bounded alternating Turing machines with auxiliary stack is denoted by **AuxS-ASpace**$(s(n))$. The infix **S** is changed to **NES** (**CS**, respectively) for Turing machines with auxiliary nonerasing stack (checking stack, respectively) storage. Deterministic, nondeterministic, bounded alternating classes, and simultaneously space and time-bounded classes are appropriately defined and denoted. We consider

$$\mathbf{L} \subseteq \mathrm{LOG}(\mathbf{DCFL}) := \mathbf{AuxPD\text{-}DSpaceTime}(\log n, \mathrm{pol}\, n)$$

$$\subseteq \mathrm{LOG}(\mathbf{CFL}) := \mathbf{AuxPD\text{-}NSpaceTime}(\log n, \mathrm{pol}\, n) \subseteq \mathbf{P}.$$

For Turing machines augmented with an auxiliary type X storage, the concept of empty alternation was introduced in the context of logspace by Lange and Reinhardt [13]. More precisely, we define an auxiliary storage automaton to be empty alternating if in moments of alternation, i.e., during transitions between existential and universal configurations and vice versa, the auxiliary storage is empty and all transferred information is contained in the state and on the $s(n)$ space-bounded Turing tape. We indicate that a class is defined by empty alternation by inserting a letter **E** in front of $\Sigma_{a(n)}$, $\Pi_{a(n)}$, or **A**. Thus, e.g., the class of all languages accepted by empty $s(n)$ space-bounded alternating Turing machines with an auxiliary storage of type X is denoted by **Aux**X-**EASpace**$(s(n))$, where X is **PD**, **S**, **NES**, or **CS** in this paper.

Finally, we need notation to describe relativized complexity classes, especially adaptive, nonadaptive, and bounded query classes. For a class **C** of languages let **DTime**$(t(n))^{\mathbf{C}}$ be the class of all languages accepted by deterministic $O(t(n))$ time-bounded Turing machines using an oracle $B \in \mathbf{C}$. If the underlying oracle Turing machine is nondeterministic, we distinguish between Ladner–Lynch (LL) [12] and Ruzzo–Simon–Tompa (RST) [19] relativization. In the latter, the oracle tape is written deterministically and the relevant class is for example denoted by $\mathbf{NTime}(t(n))^{\langle \mathbf{C} \rangle}$. It is well known that the polynomial hierarchy may be characterized in terms of oracle access, i.e., $\Sigma_0\mathbf{P} = \mathbf{P}$ and $\Sigma_{k+1}\mathbf{P} = \mathbf{NP}^{\Sigma_k \mathbf{P}}$ for $k \geqslant 0$. By definition, the Turing machine may compute the next query depending on the answers to the previous queries, i.e., it makes adaptive queries. We define the "nonadaptive query class" $\mathbf{DTime}(t(n))^{\mathbf{C}}_{||}$ to be the class of languages accepted by some deterministic $O(t(n))$ time-bounded Turing machine M using an oracle $B \in \mathbf{C}$ such that for all inputs x and all computations on x, a list of all the queries to be made is formed before any querying occurs. We say that M makes parallel queries. Moreover, for a function $r : \mathbb{N} \to \mathbb{N}$ and a class **C** let $\mathbf{DTime}(t(n))^{\mathbf{C}[r]}$ denote the class of all languages accepted by some deterministic $O(t(n))$ time-bounded Turing machine M using an oracle $B \in \mathbf{C}$ such that, for all inputs x and all computations of M on x, the number of queries to the oracle B is at most $r(|x|)$. For further explanation and results on bounded query classes we refer to [23].

Obviously, all these definitions concerning oracle Turing machines and classes can be adapted in a straightforward manner to space-bounded Turing machines, where the oracle tape is not subject to the space bound. The class $\Theta_2\mathbf{P}$ introduced by Papadimitriou and Zachos [18] is defined as $\mathbf{L}^{\mathbf{NP}}$—though in a form that at that time was not

known to be equivalent to the definition just given. It equals $\mathbf{L}^{\mathbf{NP}[O(\log n)]}$, $\mathbf{P}^{\mathbf{NP}[O(\log n)]}$, $\mathbf{L}_{||}^{\mathbf{NP}}$, and $\mathbf{P}_{||}^{\mathbf{NP}}$ [23, p. 844, Theorem 8.1], contains $\Sigma_1\mathbf{P} \cup \Pi_1\mathbf{P} = \mathbf{NP} \cup \mathbf{co\text{-}NP}$ and is contained in $\Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$—even in $\Delta_2\mathbf{P}$. Finally, the class **DP**, difference polytime, is the class of all languages—first studied by Papadimitriou and Yannakakis [17]—that can be written as the difference of two **NP** languages. Obviously, difference polytime $\mathbf{DP} = \{ A \cap B \mid A \in \mathbf{NP} \text{ and } B \in \mathbf{co\text{-}NP} \}$ and its complement $\mathbf{co\text{-}DP} = \{ A \cup B \mid A \in \mathbf{NP} \text{ and } B \in \mathbf{co\text{-}NP} \}$.

3. Alternating auxiliary stack automata

In this section we consider alternating auxiliary stack automata with and without runtime restrictions.

3.1. Automata without runtime restrictions

Ladner et al. [14] showed that

$$\mathbf{AuxS\text{-}ASpace}(s(n)) = \mathbf{AuxNES\text{-}ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{c\cdot s(n)}})$$

if $s(n) \geqslant \log n$. Auxiliary checking stacks were not considered—not even mentioned—in that paper. We complete the picture by showing that when the number of alternations is unbounded, $s(n)$ space-bounded alternating checking stacks are as powerful as stacks and nonerasing stacks. The following proof follows the lines of Ladner et al. [14, p. 152, Theorem 5.1].

Theorem 1. *Let* X *be a stack storage and* $s(n) \geqslant \log n$, *then*

$$\mathbf{Aux}\text{X}\mathbf{\text{-}ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{c\cdot s(n)}}).$$

Proof. The inclusion from left to right follows from the result of Ladner et al. [14]. For the converse inclusion it suffices to prove

$$\bigcup_c \mathbf{ASpace}(2^{2^{c\cdot s(n)}}) \subseteq \mathbf{AuxCS\text{-}ASpace}(s(n))$$

because $\mathbf{ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})$ if $s(n) \geqslant \log n$, which is due to Chandra et al. [3].

In principle we proceed as in the step-by-step simulation of an alternating $2^{2^{c\cdot s(n)}}$ space-bounded Turing machine M by an auxiliary nonerasing stack automaton, by nondeterministically guessing *ID*'s of M into the stack and after each guess verifying that they combine into a valid and successful computation. Since the *ID*'s are words of length $2^{2^{c\cdot s(n)}}$ each symbol of an *ID* is preceded by a binary address, which runs from 0 to $2^{2^{c\cdot s(n)}} - 1$. Since the length of the address is only $2^{c\cdot s(n)}$, thus $O(s(n))$ storage is sufficient to record a pointer to a particular bit position within an address. Therefore,

an $s(n)$ space-bounded alternating checking stack can verify that two physically consecutive addresses are numerically consecutive, and it can be checked that the address of a symbol deep in the checking stack which is being scanned in read mode matches the address of the symbol on top of the checking stack. This matching ability is used by the alternating checking stack to verify that the guessed sequence is valid and successful using universal branching, because one branch of the computation can read from the checking stack, while another branch retains the information for future use. Thus, even a checking stack is enough to do the simulation. For a detailed description we refer the reader to Ladner et al. [14, p. 152, Lemma 5.2]. □

In case $s(n) = \log n$ we obtain the following corollary.

Corollary 2. *Let* X *be a stack storage, then*

$$\mathbf{Aux}\text{X-}\mathbf{ASpace}(\log n) = \bigcup_k \mathbf{DTime}(2^{2^{n^k}}).$$

Now, let us take a look at variants of alternating auxiliary stack automata with a bounded number of alternations.

Theorem 3. *Let* X *be a stack storage. If* $s(n) \geqslant \log n$ *and* $k \geqslant 1$, *then*

$$\mathbf{Aux}\text{X-}\Sigma_k\mathbf{Space}(s(n)) \subseteq \bigcup_c \mathbf{NSpace}(2^{2^{c\cdot s(n)}}).$$

Proof. Recall that the alternation-bounded PDA Theorem of Ladner et al. [15, pp. 100–104, Section 5] shows that for constant k

$$\mathbf{Aux}\mathbf{PD}\text{-}\Sigma_k\mathbf{Space}(s(n)) \subseteq \bigcup_c \mathbf{NSpace}(2^{c\cdot s(n)}) \tag{1}$$

if $s(n) \geqslant \log n$. The main idea in the intricate argument used by Ladner et al. to prove (1) is to generalize the notion of realizable pairs of auxiliary pushdown automata surface configurations. A pair (ID, U), where ID is a surface configuration and U a *set* of (popping) surface configurations, is realizable if there is a consistent computation tree (including all children at a universal node and precisely one child at an existential node) whose root is labeled ID, whose leaves have labels which are members of U, such that ID and all surface configurations in U have the same pushdown height, and the pushdown does not dip below this height during the computation described by the tree. The definition of a realizable pair (ID, U) gives rise to a recursive nondeterministic algorithm, which is roughly bounded by the space to store a set of surface configurations, that is $2^{c\cdot s(n)}$ for some constant c.

In the case of an auxiliary stack automaton we cannot use this algorithm directly, but we may adapt it for our needs. Since the major difference between a pushdown and a stack is that the latter is also allowed to read but not to change the interior content, we have to face and overcome this problem. When in read mode, an alternating auxiliary stack automaton M can be viewed as an alternating *finite* automaton with $2^{c\cdot s(n)}$ states for some constant c (the input to the automaton is its stack content). Thus, the behavior

of M in this mode can be described by a table whose entries are sets of surface configurations. This idea was also used by several authors, see, e.g., Ladner et al. [14, p. 148], where the tables are called *response sets*. An entry (ID, U) in the table T means that there is a computation tree whose root is labeled ID, all leaves have labels in U, the only way out of a configuration in U is to push or pop, and the whole computation described by the tree is done in read mode. Obviously, there are at most $2^{2^{c' \cdot s(n)}}$ tables for some constant c'. We call a pair (ID, T) an extended surface configuration if ID is an ordinary surface configuration and T is a table as described above.

Now we alter the algorithm of Ladner et al. for auxiliary pushdown automata, so that it works on extended surface configurations and sets, by adapting the derivation relations accordingly. A careful analysis shows that it may be implemented on a Turing machine with a constant number of alternations. The space is bounded by the number of extended surface configurations which is roughly $2^{2^{c'' \cdot s(n)}}$ for some constant c''. Since the constant-bounded alternation hierarchy for space-bounded Turing machines collapses by Immerman [10] and Szelepcsényi [22], we obtain our result. □

3.2. Automata with restricted running time

Now we turn our attention to simultaneous space and time bounds. Here we obtain the following, which can be shown by a phase-by-phase simulation preserving the number of alternations on both sides.

Theorem 4. *Let* X *be a stack storage. If* $s(n) \geqslant \log n$, *then*

$$\bigcup_c \textbf{Aux}\text{X-}\textbf{ASpaceTime}(s(n), 2^{c \cdot s(n)}) = \bigcup_c \textbf{ATime}(2^{c \cdot s(n)}).$$

Proof. The inclusion from left to right is obvious. For the other inclusion it is sufficient to show how to simulate an alternating $2^{c \cdot s(n)}$, for some c, time-bounded Turing machine M by an alternating auxiliary checking stack automaton with appropriate space and time bounds.

The auxiliary checking stack automaton A simulates the alternating phases of M as follows: (1) If M starts in an existential (universal, respectively) configuration, then A also does and deterministically writes the initial configuration of M onto the checking stack. (2) If the automaton A tries to simulate an existential (universal, respectively) phase of M, the machine A begins existentially (universally, respectively) writing a sequence of at most $2^{c \cdot s(n)}$ configurations on the checking stack. Of course, A has not yet checked that the sequence represents a valid computation of M. Note that, in the universal case, all sequences up to the given length will be generated on different branches. Now we distinguish two cases: (2.1) If the last guessed configuration is a halting configuration of M, then we immediately check validity, which can be done deterministically in $O(s(n))$ space. In case A was simulating an existential (universal, respectively) phase, A accepts (rejects, respectively) if the guessed sequence of configurations is an accepting (rejecting, respectively) computation of M, otherwise A

rejects (accepts, respectively). (2.2) Let ID be the last configuration guessed and assume that it is nonhalting. Then A alternates, continues the simulation of the next phase along one branch, where the information in the checking stack remains for future use with no read operation having been performed along that branch, and checks whether the guessed sequence is a valid computation along the other branch. The acceptance condition for this "checking" branch is as follows: Assume that A was simulating an existential (universal, respectively) phase of M. Then the checking branch is accepting (rejecting, respectively) if the guessed sequence is a valid existential (universal, respectively) computation, otherwise this branch is rejecting (accepting, respectively). Note that as in case (2.1) the necessary checks can be done deterministically in the appropriate space bound. This completes the description of A.

Obviously, the automaton A accepts the same language as M and uses only $O(s(n))$ space. Moreover, since A simulates M phase by phase, the running time of A is bounded by $2^{c' \cdot s(n)}$ for some constant c'. Thus the claim follows. □

Since $\mathbf{PSpace} = \mathbf{ATime}(\operatorname{pol} n)$ we conclude:

Corollary 5. *If* X *is a stack storage, then*

$$\mathbf{Aux}\text{X-}\mathbf{ASpaceTime}(\log n, \operatorname{pol} n) = \mathbf{PSpace}.$$

Note that in the previous proof the number of alternations is preserved during the simulations, but we have to be careful in the deterministic case, since the simulation for the inclusion from right to left uses at least a nondeterministic auxiliary stack automaton. Thus, we can merely state:

Corollary 6. *Let* X *be a stack storage. If* $s(n) \geqslant \log n$ *and* $k \geqslant 1$, *then*

$$\bigcup_c \mathbf{Aux}\text{X-}\Sigma_k\mathbf{SpaceTime}(s(n), 2^{c \cdot s(n)}) = \bigcup_c \Sigma_k\mathbf{Time}(2^{c \cdot s(n)}).$$

But what about the deterministic case? The following theorem answers this question for auxiliary stack and nonerasing stack automata and can be shown by step-by-step simulations.

Theorem 7. *Let* $s(n) \geqslant \log n$. *We have*:

(i) *Let* X *be a stack or nonerasing stack, then*

$$\bigcup_c \mathbf{Aux}\text{X-}\mathbf{DSpaceTime}(s(n), 2^{c \cdot s(n)}) = \bigcup_c \mathbf{DTime}(2^{c \cdot s(n)}).$$

(ii) $\bigcup_c \mathbf{AuxCS\text{-}DSpaceTime}(s(n), 2^{c \cdot s(n)}) = \mathbf{DSpace}(s(n))$.

Proof. (1) The inclusion from left to right is obvious, and the converse follows by a simple step-by-step simulation of the $2^{c \cdot s(n)}$ time-bounded deterministic Turing machine M writing the configuration sequence to the (nonerasing) stack. If ID is the configuration on top of the stack, the successor of ID can be computed deterministically in

$2^{c\cdot s(n)}$ time and $s(n)$ space, and added to the stack. Thus, the overall running time is bounded by $2^{c'\cdot s(n)}$, for some c'.

(2) The inclusion from right to left is obvious. The other direction is seen as follows: Trivially,

$$\textbf{AuxCS-DSpaceTime}(s(n), 2^{c\cdot s(n)}) \subseteq \textbf{AuxCS-DSpace}(s(n))$$

and the latter class is contained in $\textbf{DSpace}(s(n))$, if $s(n) \geqslant \log n$, which was shown by Ibarra in [9]. □

The results in this subsection in the case $s(n) = \log n$ yield an alternate characterization of the polynomial hierarchy, to be contrasted with that given by Jenner and Kirsig [11, p. 94, Theorem 3.4] in terms of auxiliary pushdown automata, where

$$\textbf{AuxPD-}\Sigma_{k+1}\textbf{SpaceTime}(\log n, \operatorname{pol} n) = \Sigma_k\mathbf{P}$$

for $k \geqslant 1$ was shown.

Corollary 8. (1) *Let* X *be a stack or nonerasing stack. For* $k \geqslant 0$, *we find*

$$\textbf{Aux}\text{X-}\Sigma_k\textbf{SpaceTime}(\log n, \operatorname{pol} n) = \Sigma_k\mathbf{P}.$$

(2) *For checking stacks we have*

$$\textbf{AuxCS-}\Sigma_0\textbf{SpaceTime}(\log n, \operatorname{pol} n) = \textbf{DSpace}(\log n)$$

and

$$\textbf{AuxCS-}\Sigma_k\textbf{SpaceTime}(\log n, \operatorname{pol} n) = \Sigma_k\mathbf{P} \quad \textit{if } k \geqslant 1.$$

Note that [11, pp. 98–99] claims the case $k \geqslant 1$ of part 2 of Corollary 8 as well as the auxiliary checking stack characterization of **PSpace** given in Corollary 5.

4. Empty alternating auxiliary stack automata

Lange and Reinhardt [13] exhibit a close connection between (RST) relativized complexity classes and empty alternation. Here we obtain similar results for empty alternating auxiliary stack automata. We start with a lemma generalizing a result of Lange and Reinhardt [15, p. 501, Theorem 10].

Lemma 9. *Let* X *be a stack storage and assume that* $s(n) \geqslant \log n$. *If we have* $\textbf{Aux}\text{X-}\textbf{NSpace}(\log n) \subseteq \mathbf{C}_1$ *and* $\textbf{Aux}\text{X-}\textbf{NSpace}(\log n, \operatorname{pol} n) \subseteq \mathbf{C}_2$, *for some* $\mathbf{C}_1$ *and* $\mathbf{C}_2$, *then*

(1) $\textbf{Aux}\text{X-}\textbf{EASpace}(s(n)) \subseteq \bigcup_c \textbf{DTime}(2^{c\cdot s(n)})^{\mathbf{C}_1}_{||}$ *and*
(2) $\bigcup_c \textbf{Aux}\text{X-}\textbf{EASpaceTime}(s(n), 2^{c\cdot s(n)}) \subseteq \bigcup_c \textbf{DTime}(2^{c\cdot s(n)})^{\mathbf{C}_2}_{||}$.

Proof. We only prove the first statement. A similar construction proves the second. Let M be an empty alternating $s(n)$ space-bounded automaton with auxiliary storage

type X. For each x, we denote the set of all $s(|x|)$ space-bounded configurations of M such that the auxiliary storage is empty by $C(x)$. Obviously, $|C(x)| \leqslant |x| \cdot c^{s(|x|)}$, for some constant c.

Consider the naturally induced alternating graph accessibility problem on a graph $G=(V,E)$ with nodes $V = C(x) \cup \{accept, reject\}$, where the union is disjoint, such that $ID \in C(x)$ is an existential (universal, respectively) node if and only if ID is an existential (universal, respectively) configuration. The edge relation is defined as $(ID_1, ID_2) \in E$ if and only if ID_1 and ID_2 belong to $C(x)$ and there is a computation of M on input x such that ID_2 can be reached from ID_1 without any alternation. Moreover, $(ID, accept) \in E$ if and only if ID is in $C(x)$ and there is a computation of M on input x such that an accepting halting configuration (not necessarily in $C(x)$) is reached. Analogously, we have $(ID, reject) \in E$ if a rejecting halting configuration can be reached. Obviously, the edge relation can be computed by a nondeterministic $s(n)$ space-bounded automaton with auxiliary storage X. Thus, E is in $\mathbf{C}_1$.

With one parallel query to E we can compute the alternating graph structure, whose accessibility problem is then solved by a deterministic $2^{c' \cdot s(n)}$ time-bounded Turing machine, such that $|x| \cdot c^{s(|x|)} \leqslant 2^{c' \cdot s(|x|)}$, for some constant c'. This proves the stated claim. □

4.1. Automata without runtime restrictions

With the help of the above lemma we can now extend the argument in [13, p. 498, Theorem 5] and show that the empty alternation hierarchy for auxiliary stack automata collapses to its nondeterministic level.

Theorem 10. *Let* X *be a stack storage. If* $s(n) \geqslant \log n$, *then*

$$\mathbf{Aux}\text{X-}\mathbf{EASpace}(s(n)) = \mathbf{Aux}\text{X-}\mathbf{NSpace}(s(n)).$$

Proof. The inclusion from right to left is obvious. For the other direction let M be an empty alternating $s(n)$ space-bounded automaton with an auxiliary stack (nonerasing stack, checking stack, respectively) storage. By the results of Ibarra [9] we have

(1) $\mathbf{AuxS\text{-}NSpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{c \cdot s(n)}})$,
(2) $\mathbf{AuxNES\text{-}NSpace}(s(n)) = \bigcup_c \mathbf{DSpace}(2^{c \cdot s(n)})$, and
(3) $\mathbf{AuxCS\text{-}NSpace}(s(n)) = \bigcup_c \mathbf{NSpace}(2^{c \cdot s(n)})$

if $s(n) \geqslant \log n$. Observe that $\bigcup_c \mathbf{NSpace}(2^{c \cdot s(n)})$ equals $\bigcup_c \mathbf{DSpace}(2^{c \cdot s(n)})$ by Savitch's theorem [20].

By standard simulation techniques we find that

$$\bigcup_c \mathbf{DTime}(2^{c \cdot s(n)}) \bigcup_{||k} \mathbf{DTime}(2^{n^k}) \subseteq \bigcup_c \mathbf{DTime}(2^{2^{c \cdot s(n)}})$$

and

$$\bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})_{||}^{\mathbf{PSpace}} \subseteq \bigcup_c \mathbf{DSpace}(2^{c\cdot s(n)}),$$

which implies by Lemma 9 and the above mentioned results (1)–(3) that

$$\mathbf{Aux}\text{X-}\mathbf{EASpace}(s(n)) \subseteq \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})_{||}^{\mathbf{C}} \subseteq \mathbf{Aux}\text{X-}\mathbf{NSpace}(s(n)),$$

where $\mathbf{C} = \mathbf{Aux}\text{X-}\mathbf{NSpace}(\log n)$, for all three types of auxiliary stack automata considered. □

For $s(n) = \log n$ we obtain from Theorem 10 the following corollary.

Corollary 11. (1) *We have*

$$\mathbf{AuxS\text{-}EASpace}(\log n) = \bigcup_k \mathbf{DTime}(2^{n^k}).$$

(2) *Let* X *be a nonerasing or checking stack, then*

$$\mathbf{Aux}\text{X-}\mathbf{EASpace}(\log n) = \mathbf{PSpace}.$$

4.2. Automata with restricted running time

Obviously, since empty alternating Σ_0 and Σ_1 machines are nothing other than ordinary deterministic and nondeterministic automata, we profit from the results in Subsection 3.2 and refer in these cases to Corollary 6 and Theorem 7. Moreover, for an unbounded number of alternations the upper bound was already settled in Lemma 9. For the lower bound, we obtain the following, where we have to distinguish between stack and nonerasing (checking) stack automata. We start with auxiliary stack automata and adapt an argument from [13, p. 501, Theorem 10].

Lemma 12. *Let* $s(n) \geqslant \log n$, *then*

$$\mathbf{DSpace}(s(n))^{\mathbf{NP}[\mathrm{O}(s(n))]} \subseteq \bigcup_c \mathbf{AuxS\text{-}E}\Sigma_2\mathbf{SpaceTime}(s(n), 2^{c\cdot s(n)}).$$

Proof. Let M be a $s(n)$ space-bounded oracle Turing machine with an oracle $A \in \mathbf{NP}$, making at most $\mathrm{O}(s(n))$ queries to A. Further, since $\mathbf{NP} = \mathbf{AuxS\text{-}NSpaceTime}(\log n, \mathrm{pol}\, n)$ by Corollary 8, let M_A and $M_{\bar{A}}$ be the auxiliary automata accepting A and its complement, respectively. Note that $M_{\bar{A}}$ is a machine with universal states only.

We simulate M with an auxiliary stack automaton M' as follows. First M' existentially guesses all answers to the $\mathrm{O}(s(n))$ queries to the oracle A and writes them on the space-bounded tape. Then it starts simulating the deterministic steps of M. If M asks the ith oracle question, M' looks up the answer from the space-bounded tape. In case the look up is "yes", the machine M' simulates the machine M_A for oracle set A, and rejects if M_A does. The simulation of this machine is possible, because of the assumption on A. Otherwise, i.e., the answer looked up is "no", the verification is

postponed. Then M' continues the simulation of M. If M accepts, M' empties its stack, then universally alternates, and restarts M's simulation. Now the "yes" query answers are taken for granted. Thus, it remains to verify the "no" answers. This is done by simulating $M_{\bar{A}}$ which accepts the complement of A, where M' rejects if $M_{\bar{A}}$ rejects. Then M' continues with the simulation of M, and accepts if the original machine M accepts. It is easy to see that M' only alternates twice and satisfies the required space and time bounds. For the latter observe, that the length of the oracle query is at most $2^{c\cdot s(n)}$, for some constant c. Thus, the space and time bound, respectively, needed for the simulation of M_A or $M_{\bar{A}}$ on an oracle query are bounded by $c'\cdot s(n)$ and $2^{c''\cdot s(n)}$, respectively, for some constants c' and c''. □

The difficulty in the nonerasing (checking) stack case is that after verifying a "yes" answer to a query, one can not empty the stack anymore. Thus, the empty alternating machine can not universally alternate in order to verify the remaining "no" answers. The next lemma shows that with one more alternation, we can overcome this problem.

Lemma 13. *Let* X *be a nonerasing or checking stack and* $s(n)\geqslant \log n$. *Then*

$$\mathbf{DSpace}(s(n))^{\mathbf{NP}[\mathrm{O}(s(n))]} \subseteq \bigcup_c \mathbf{Aux}\mathrm{X}\text{-}\mathbf{E}\Sigma_3\mathbf{SpaceTime}(s(n), 2^{c\cdot s(n)}).$$

Proof. We first proceed as in the proof of Lemma 12 by first guessing the query answers to the oracle, and writing them on the space-bounded tape. Then the simulation of the deterministic steps of the original machine are done, where all answers to the query questions are looked up from the tape. If the original machine accepts, we can alternate universally, because up to this point the stack was not used. In the universal phase, the "yes" and "no" answers are verified in parallel, as follows. The deterministic simulation of the original machine is restarted. If the ith oracle question is asked, the machine branches universally, verifying (1) the answer to the oracle along one branch and (2) continuing the simulation along the other. If the look-up answer is "yes", then in (1) the machine alternates existentially and runs the auxiliary stack automaton for the oracle A. Otherwise, i.e., the look-up answer is "no", in (1) the machine runs the auxiliary stack automaton for the complement of A. Finally, the machine accepts if the original machine accepts.

As we see, the simulation is restructured in such a way that the stack is used only at the very end of each branch. The price we have to pay for that is an additional alternation compared to Lemma 12. Thus, three alternations are sufficient in the case of nonerasing and checking stacks. Moreover, it is easy to verify that the required space and time bounds are satisfied. □

The above two lemmata together with Lemma 9 show that for $s(n)\geqslant \log n$ we have sandwiched

$$\bigcup_c \mathbf{AuxS}\text{-}\mathbf{E}\Sigma_2\mathbf{SpaceTime}(s(n), 2^{c\cdot s(n)})$$

and

$$\bigcup_c \mathbf{Aux}\text{X-}\mathbf{E}\Sigma_3\mathbf{SpaceTime}(s(n), 2^{c\cdot s(n)}),$$

where X is a nonerasing or checking stack, as well as the corresponding unbounded alternating classes in between $\mathbf{DSpace}(s(n))^{\mathbf{NP}[\mathrm{O}(s(n))]}$ and the nonadaptive query class $\bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})_{||}^{\mathbf{NP}}$. Note that the **NP** oracle in the latter class is because of Corollary 6. In the case $s(n) = \log n$, these bounded query classes are known to be equal due to Wagner [23, p. 838, Corollary 3.7.1]. Fortunately, this generalizes to arbitrary space bounds greater than $\log n$, as a careful analysis of translational methods reveals.

Theorem 14. *If* $s(n) \geqslant \log n$, *then*

$$\mathbf{DSpace}(s(n))^{\mathbf{NP}[\mathrm{O}(s(n))]} = \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})_{||}^{\mathbf{NP}}.$$

Proof. For a language L over the alphabet Σ we define the mapping

$$T_s(L) = \{\, w\#^{2^{s(|w|)}-|w|} \mid w \in L \,\},$$

where $\# \notin \Sigma$. Then by easy calculations we find that

$$L \in \mathbf{DSpace}(s(n))^{\mathbf{NP}[\mathrm{O}(s(n))]} \quad \text{if and only if} \quad T_s(L) \in \mathbf{L}^{\mathbf{NP}[\mathrm{O}(\log n)]}.$$

Since $\mathbf{L}^{\mathbf{NP}[\mathrm{O}(\log n)]} = \mathbf{P}_{||}^{\mathbf{NP}}$, which is due to Wagner [23, p. 838, Corollary 3.7.1], language $T_s(L)$ is also a member of $\mathbf{P}_{||}^{\mathbf{NP}}$. Hence, by translational methods again, we have

$$T_s(L) \in \mathbf{P}_{||}^{\mathbf{NP}} \quad \text{if and only if} \quad L \in \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})_{||}^{\mathbf{NP}}.$$

This proves the stated claim. □

As an immediate corollary we obtain:

Corollary 15. *Let* X *be a stack storage. If* $s(n) \geqslant \log n$, *then*

$$\bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})_{||}^{\mathbf{NP}} = \bigcup_c \mathbf{Aux}\text{X-}\mathbf{E}\Sigma_k\mathbf{SpaceTime}(s(n), 2^{c\cdot s(n)})$$

$$= \bigcup_c \mathbf{Aux}\text{X-}\mathbf{EASpaceTime}(s(n), 2^{c\cdot s(n)}),$$

where $k = 2$ *if* X *is a stack and* $k = 3$ *if* X *is a nonerasing or checking stack.*

For the special case $s(n)=\log n$ the above corollary results in a characterization of $\Theta_2\mathbf{P}$, using $\Theta_2\mathbf{P}=\mathbf{P}_{||}^{\mathbf{NP}}=\mathbf{L}^{\mathbf{NP}[O(\log n)]}=\mathbf{P}^{\mathbf{NP}[O(\log n)]}$ proven by Wagner [23, p. 844, Theorem 8.1]. A similar result was obtained by Lange and Reinhardt [13, p. 501, Theorem 10] for empty alternating polytime machines.

Corollary 16. *Let* X *be a stack storage, then we have*

$$\Theta_2\mathbf{P}=\mathbf{Aux}\text{X-}\mathbf{E}\Sigma_k\mathbf{SpaceTime}(\log n,\operatorname{pol} n)$$

$$=\mathbf{Aux}\text{X-}\mathbf{EASpaceTime}(\log n,\operatorname{pol} n),$$

where $k=2$ *if* X *is a stack and* $k=3$ *if* X *is a nonerasing or checking stack.*

Thus, it remains to classify one empty alternation on auxiliary nonerasing and checking stack automata. Again, we find a close connection with bounded query classes, restricting the oracle access to a single question only.

Theorem 17. *Let* X *be a nonerasing or checking stack storage. If* $s(n)\geqslant\log n$, *then*

$$\bigcup_c \mathbf{Aux}\text{X-}\mathbf{E}\Sigma_2\mathbf{SpaceTime}(s(n),2^{c\cdot s(n)})=\mathbf{NSpace}(s(n))^{\langle\mathbf{NP}[1]\rangle}.$$

Proof. First we prove the inclusion from right to left. Let M be a nondeterministic $s(n)$ space-bounded Turing machine with an oracle $A\in\mathbf{NP}$ queried exactly *once*. Further, since $\mathbf{NP}=\mathbf{Aux}\text{X-}\mathbf{NSpaceTime}(\log n,\operatorname{pol} n)$ by Corollary 8, let M_A and $M_{\bar{A}}$ be the auxiliary automata accepting A and its complement, respectively. Observe that $M_{\bar{A}}$ is a machine with universal states only.

We simply simulate M's nondeterministic moves by an auxiliary nondeterministic automaton M' with storage X. If M starts writing deterministically on the oracle tape, the *ID* of M is stored on the space-bounded worktape, M' guesses the answer to that oracle question, stores it in the finite control, and proceeds with the simulation of M. If M accepts, the verification of the oracle answers has to be done. In case the answer was "yes", the machine M starts the simulation of M_A using *ID* to deterministically reconstruct the oracle question. If M_A accepts, then M' accepts. The "no" answer is checked by M by alternating universally—the stack storage is empty, because it was not used up to now—and simulating $M_{\bar{A}}$. Here M' accepts, if $M_{\bar{A}}$ does. Thus, two empty alternations suffice. Observe, that the length of the oracle query is at most $2^{c\cdot s(n)}$, for some constant c. Thus, the space and time bound, respectively, needed for the simulation of M_A or $M_{\bar{A}}$ on an oracle query are bounded by $c'\cdot s(n)$ and $2^{c''\cdot s(n)}$, respectively, for some constants c' and c''.

For the converse inclusion let M be a $s(n)$ space and $2^{c\cdot s(n)}$ time-bounded empty alternating Σ_2 auxiliary nonerasing (checking) stack automaton. Let $C(x)$ be the set of all configurations of M on input x with empty auxiliary nonerasing (checking) stack

storage. Define the oracles

$$A_{\exists,M} = \{\, ID\#^{2^{s(n)}-|ID|} \mid M \text{ starting in } ID \in C(x) \text{ accepts with at most } 2^{c\cdot s(n)} \text{ exist. moves only} \,\}$$

and

$$A_{\forall,M} = \{\, ID\#^{2^{s(n)}-|ID|} \mid M \text{ starting in } ID \in C(x) \text{ accepts with at most } 2^{c\cdot s(n)} \text{ univ. moves only} \,\}.$$

For the simulation we consider two cases. Either M first writes to the stack in the existential or in the universal phase. Note that in the former case M will in fact never have had the chance to alternate at all. Thus, the simulation by a $s(n)$ space-bounded nondeterministic Turing machine M' runs as follows: Machine M' nondeterministically guesses whether the first write instruction is performed during the existential or universal phase of M. In the former case, M' asks the question "$ID_0\#^{2^{s(n)}-|ID_0|} \in A_{\exists,M}$", where ID_0 is the initial configuration of M, and accepts if the answer is "yes". In the latter case, where the stack is not used in existential moves, M' simulates M step-by-step until it reaches the ID where the alternation appears, and asks the complement of $A_{\forall,M}$. If the answer to the question "$ID\#^{2^{s(n)}-|ID|} \in \mathbf{co}\text{-}A_{\forall,M}$" is "yes", then M' rejects. Otherwise, it accepts. Since $A_{\exists,M}$ and the complement of $A_{\forall,M}$ both belong to **NP** our claim follows. □

Finally, consider the class $\mathbf{NSpace}(s(n))^{\langle \mathbf{NP}[1]\rangle}$ in more detail. In general

$$\mathbf{DSpace}(s(n))^{\mathbf{C}} \subseteq \mathbf{NSpace}(s(n))^{\langle \mathbf{C}\rangle} \subseteq \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})^{\mathbf{C}}$$

for any **C** if $s(n) \geqslant \log n$, since we use RST relativization. Trivially, the first inclusion also holds in the case of bounded query classes, in particular for *one* query to the oracle. The second inclusion is not known to carry over to the setting of bounded query classes, but in our case, the next theorem implies

$$\mathbf{NSpace}(s(n))^{\langle \mathbf{NP}[1]\rangle} \subseteq \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})^{\mathbf{NP}[2]}.$$

Indeed we can characterize the class $\mathbf{NSpace}(s(n))^{\langle \mathbf{NP}[1]\rangle}$ as the complement of a "difference" set.

Theorem 18. *If* $s(n) \geqslant \log n$ *and* $\mathbf{C} = \bigcup_c \mathbf{NTime}(2^{c\cdot s(n)})$, *then*

$$\mathbf{NSpace}(s(n))^{\langle \mathbf{NP}[1]\rangle} = \{\, A \cup B \mid A \in \mathbf{C} \text{ and } B \in \mathbf{co}\text{-}\mathbf{C} \,\}.$$

Proof. First we prove the inclusion from right to left. Consider a language $A \cup B$ with $A \in \mathbf{C}$ and $B \in \mathbf{co}\text{-}\mathbf{C}$. We construct the oracle set

$$A \oplus \mathbf{co}\text{-}B = \{\, 1x\#^{2^{s(|x|)}-|x|} \mid x \in A \,\} \cup \{\, 2x\#^{2^{s(|x|)}-|x|} \mid x \in \mathbf{co}\text{-}B \,\}$$

as the marked union of A and the complement of B padded by # symbols to an exponential length in $s(n)$. Note that $A \oplus \mathbf{co}\text{-}B$ belongs to the class **NP**. Then a nondeterministic $s(n)$ space-bounded Turing machine on input x decides membership in $A \cup B$ by existentially guessing the word $1x\#^{2^{s(|x|)}-|x|}$ or $2x\#^{2^{s(|x|)}-|x|}$ onto the oracle tape. If the answer to the question "$1x\#^{2^{s(|x|)}-|x|} \in A \oplus \mathbf{co}\text{-}B$" is "yes", the machine accepts. It also accepts if the answer to the question "$2x\#^{2^{s(|x|)}-|x|} \in A \oplus \mathbf{co}\text{-}B$" is "no". Otherwise, it rejects. Thus, one question to an **NP** oracle suffices to decide $x \in A \cup B$.

For the converse inclusion let M be a nondeterministic $s(n)$ space-bounded Turing machine with an oracle $A \in \mathbf{NP}$ queried exactly *once*. Further let M_A be the nondeterministic polynomial time-bounded machine accepting A.

We modify M's behavior such that the oracle query is asked at the very end of each computation. This is done as follows: If M starts writing deterministically on the oracle tape, the *ID* is stored on the space-bounded worktape, M guesses the answer to that oracle question, stores it in the finite control, and proceeds with the computation. If M wants to accept, the oracle is questioned using *ID* to recompute the question. In case the answer coincides with the previously stored guess, then the machine accepts. Otherwise, it rejects. Moreover, by introducing dummy queries, which have to be answered "yes", we may assume that on each computation the oracle is queried once.

From now on we assume that M is in the normal form described above. Next we define sets

$$A_{\text{yes}} = \{\, x \mid \text{there is a computation path of } M \text{ along which } M \text{ accepts } x \text{ with a single oracle query, answered "yes"} \,\}$$

and

$$A_{\text{no}} = \{\, x \mid \text{there is a computation path of } M \text{ along which } M \text{ accepts } x \text{ with a single oracle query, answered "no"} \,\}.$$

Obviously, the language $A_{\text{yes}} \cup A_{\text{no}}$ equals the set accepted by M with oracle A queried exactly once. Thus, it remains to show that $A_{\text{yes}} \in \mathbf{C}$ and $A_{\text{no}} \in \mathbf{co}\text{-}\mathbf{C}$.

To check $x \in A_{\text{yes}}$ a nondeterministic Turing machine M' guesses a reachable *ID*, where M starts writing deterministically on the oracle tape and whose "yes" answer leads to acceptance. Then the machine deterministically verifies in time $2^{c' \cdot s(n)}$, for some constant c', that the *ID* is reachable from the initial configuration. If so M' starts the simulation of M_A using *ID* to deterministically reconstruct the oracle question. If M_A accepts, then M' accepts. Otherwise, machine M' rejects. The length of the oracle query is at most $2^{c'' \cdot s(n)}$, for some constant c''. Hence it is easy to see that the simulation of M_A on an oracle query meets the required exponential time bound. Thus, machine M' is $2^{c''' \cdot s(n)}$ time bounded, for some constant c'''. Hence, $A_{\text{yes}} \in \mathbf{C}$.

For the set A_{no} we proceed in a similar way. The complement of A_{no} is accepted by a nondeterministic Turing machine by cycling through all reachable *ID*'s, where

M starts writing deterministically on the oracle tape and whose "no" answer leads to acceptance, verifying that all answers to the oracle are "yes". Analogously to above this can be done by simulating machine M_A on all questions is sequence, hence leading to a nondeterministic Turing machine operating in $2^{c'''\cdot s(n)}$ time, for some constant c'''. Thus, $A_{\text{no}} \in \textbf{co-C}$ which proves our claim. □

For our favourite space bound $s(n) = \log n$ this results in:

Corollary 19. $\textbf{NL}^{\langle \textbf{NP}[1] \rangle} = \textbf{co-DP}$.

As an immediate consequence of the above corollary together with Theorem 17 we obtain

Corollary 20. *Let* X *be a nonerasing or checking stack, then*

$$\textbf{co-DP} = \textbf{Aux}\text{X-E}\Sigma_2\textbf{SpaceTime}(\log n, \operatorname{pol} n).$$

5. Conclusions

Tables 1 and 2 summarize the known complexity class characterizations based on **AuxS**-, **AuxNES**-, and **AuxCS**-alternating and empty alternating automata. The picture is complete in the case of empty alternating automata. In the case of alternating automata, the picture is complete, with only three exceptions involving bounded numbers of alternation greater than one (table entry $\mathbf{A} \subseteq \cdot \subseteq \mathbf{B}$). These entries indicate that the corresponding space-bounded **AuxS**-, **AuxNES**-, and **AuxCS**-alternating automata classes are somewhere between **A** and **B**, but what is their precise status?

Acknowledgements

Thanks to Lane Hemaspaandra and Klaus W. Wagner for the hint to prove Theorem 18. Also thanks to an anonymous referee for his valuable comments on Section 4, including the observation that some of our earlier arguments (now corrected) did not apply to nonlogarithmic space bounds directly.

References

[1] J.L. Balcázar, J. Díaz, J. Gabarró, Structural Complexity I, EATCS Monographs on Theoretical Computer Science, Vol. 11, Springer, Berlin, 1988.
[2] A.W. Burks, D.W. Warren, J.B. Wright, An analysis of a logical machine using parenthesis-free notation, Math. Tables Other Aids Comput. 8 (46) (1954) 53–57.
[3] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, J. ACM 28 (1) (1981) 114–133.
[4] S.A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, J. ACM 18 (1) (1971) 4–18.
[5] S. Ginsburg, S.A. Greibach, M.A. Harrison, One-way stack automata, J. ACM 14 (2) (1967) 389–418.

[6] S.A. Greibach, Checking automata and one-way stack languages, J. Comput. System Sci. 3 (2) (1969) 196–217.
[7] J.E. Hopcroft, J.D. Ullman, Nonerasing stack automata, J. Comput. System Sci. 1 (2) (1967) 166–186.
[8] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading, MA, 1979.
[9] O.H. Ibarra, Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata, J. Comput. System Sci. 5 (2) (1971) 88–117.
[10] N. Immerman, Nondeterministic space is closed under complementation, SIAM J. Comput. 17 (5) (1988) 935–938.
[11] B. Jenner, B. Kirsig, Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata, RAIRO—Inform. Théorique Appl./Theoret. Inform. Appl. 23 (1) (1989) 87–99.
[12] R.E. Ladner, R.J. Lipton, L.J. Stockmeyer, Alternating pushdown and stack automata, SIAM J. Comput. 13 (1) (1984) 135–155.
[13] R. Ladner, N. Lynch, Relativization of questions about log space computability, Math. System Theory 10 (1976) 19–32.
[14] R.E. Ladner, L.J. Stockmeyer, R.J. Lipton, Alternation bounded auxiliary pushdown automata, Inform. Control 62 (1984) 93–108.
[15] K.-J. Lange, K. Reinhardt, Empty alternation, in: Proc. 9th Conf. on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 841, Springer, Kosice, Slovakia, 1994, pp. 494–503.
[16] A. Newell, J.C. Shaw, Programming the logic theory machine, in: Proc. 1957 Western Joint Computer Conf., Institute of Radio Engineers, New York, 1957, pp. 230–240.
[17] C.H. Papadimitriou, M. Yannakakis, The complexity of facets (and some facets of complexity), J. Comput. System Sci. 28 (2) (1984) 244–259.
[18] C.H. Papadimitriou, S. Zachos, Two remarks on the power of counting, in: A.B. Cremers, H.-P. Kriegel (Eds.), Proc. 6th GI Conf. on Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 145, Springer, Dortmund, Germany, 1983, pp. 269–276.
[19] W.L. Ruzzo, J. Simon, M. Tompa, Space-bounded hierarchies and probabilistic computations, J. Comput. System Sci. 28 (2) (1984) 216–230.
[20] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comput. System Sci. 4 (2) (1970) 177–192.
[21] I.H. Sudborough, On the tape complexity of deterministic context-free languages, J. ACM 25 (3) (1978) 405–414.
[22] R. Szelepcsényi, The method of forced enumeration for nondeterministic automata, Acta Inform. 26 (3) (1988) 279–284.
[23] K.W. Wagner, Bounded query classes, SIAM J. Comput. 19 (5) (1990) 833–846.
[24] K. Wagner, G. Wechsung, Computational Complexity, Mathematics and its Applications (East Europeans series), VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.