Alternating and Empty Alternating Auxiliary Stack Automata*

Markus Holzer and Pierre McKenzie

Département d'I.R.O., Université de Montréal, C.P. 6128,
succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
email: {holzer,mckenzie}@iro.umontreal.ca

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 $\mathbf{NL}^{\langle \mathbf{NP}[1] \rangle}$ 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. Probably, such machines are best known for capturing **P** when their space is logarithmically bounded [2] and for capturing the important class LOG(**CFL**) $\subseteq$ **P** when additionally their time is polynomially bounded [11]. These two milestones in reality form part of an extensive list of equally striking characterizations (see [13, pages 373–379]). For example, a *stack* (**S**) is a pushdown store allowing its interior content 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 [2] alluded to above, that

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

when $s(n) \geq \log n$, is in sharp contrast with Ibarra's [3], who proved that

$$\mathbf{AuxS\text{-}DSpace}(s(n)) = \mathbf{AuxS\text{-}NSpace}(s(n)) = \bigcup \mathbf{DTime}(2^{2^{c\cdot s(n)}}),$$

$$\mathbf{AuxNES\text{-}DSpace}(s(n)) = \mathbf{AuxNES\text{-}NSpace}(s(n)) = \bigcup \mathbf{DSpace}(2^{c\cdot s(n)}),$$

$$\mathbf{AuxCS\text{-}NSpace}(s(n)) = \bigcup \mathbf{NSpace}(2^{c\cdot s(n)}),$$

$$\mathbf{AuxCS\text{-}DSpace}(s(n)) = \mathbf{DSpace}(s(n)),$$

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

* 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.

In the wake of [1], pushdown stores were also added to *alternating* Turing machines [7, 8]. 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 [11], Jenner and Kirsig [5] further used **AuxPD-** alternating automata with simultaneous resource bounds to capture **PH**, the polynomial hierarchy. Subsequently, Lange and Reinhardt [9] helped shed light on why **AuxPD-** alternating automata are so powerful. They introduced of 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 showed 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.* [7] 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.* [7]. We also completely answer the question posed by Lange and Reinhardt [9] concerning the power of variants of *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" forming the basis of 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 use of the checking stack and empty alternation appears 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 a (few) remaining open questions. Some of the proofs are deferred to the journal version.

2 Definitions

We assume the reader to be familiar with the basics of complexity theory as contained in Wagner and Wechsung [13]. 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 $\mathbf{N}$ instead of Σ_1 and $\mathbf{D}$ for Σ_0. Moreover, if the number of alternations is unrestricted, we simply replace $\Sigma_{a(n)}$ by $\mathbf{A}$. If we are interested in space and time classes only, we simply write $\mathbf{Space}(s(n))$ and $\mathbf{Time}(s(n))$, respectively, in our notations. 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 $k \geq 0$, 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* (**PD**) is a last-in first-out (LIFO) storage structure, which is manipulated by pushing and popping. By *stack storage*, we mean a *stack* (**S**), *nonerasing stack* (**NES**), or *checking stack* (**CS**) as defined earlier. The class of languages accepted by $O(s(n))$ space bounded alternating Turing machines with auxiliary stack is denoted by $\mathbf{AuxS\text{-}ASpace}(s(n))$. The infix **S** is changed to **NES** (**CS**, respectively) if we consider 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.

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 [9]. 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 $\mathbf{E}$ in front of $\Sigma_{a(n)}$, $\Pi_{a(n)}$, or $\mathbf{A}$. 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 $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{EASpace}(s(n))$.

Finally, we need some more notations on relativized complexity classes, especially on adaptive, non-adaptive, and bounded query classes. For a class $\mathbf{C}$ of languages let $\mathbf{DTime}(t(n))^{\mathbf{C}}$ 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) [6] and Ruzzo-Simon-Tompa (RST) [10] relativization. In the latter, the oracle tape is written deterministically and is 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 $k \geq 0$. 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 "non-adaptive 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 queries is formed before any of them is made. We say that M makes parallel queries. Moreover, for a function $r : \mathbb{N} \to \mathbb{N}$ and a class $\mathbf{C}$ 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 less than or equal $r(|x|)$. For further explanation and results on bounded query classes we refer to [14].

Obviously, all these definitions concerning oracle Turing machines and classes can be adapted in a straight-forward manner to space bounded Turing machines—there the oracle tape is not subject to the space bound. The class $\Theta_2\mathbf{P}$ is defined as $\mathbf{L}^{\mathbf{NP}}$. It equals $\mathbf{L}^{\mathbf{NP}[O(\log n)]}$, $\mathbf{P}^{\mathbf{NP}[O(\log n)]}$, $\mathbf{L}_{||}^{\mathbf{NP}}$, and $\mathbf{P}_{||}^{\mathbf{NP}}$ [14, page 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 $\Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$.

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.* [7] showed $\mathbf{AuxS\text{-}ASpace}(s(n)) = \mathbf{AuxNES\text{-}ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{2^{c \cdot s(n)}}})$ if $s(n) \geq \log n$. Surprisingly, auxiliary checking stacks were not considered—not even mentioned—in this 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. Due to the lack of space we omit the proof of the following theorem, which follows the lines of Ladner *et al.* [7, page 152, Theorem 5.1].

Theorem 1. $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{2^{c \cdot s(n)}}})$ *if* X *is a stack storage and* $s(n) \geq \log n$. □

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

Theorem 2. $\mathbf{Aux}\mathrm{X}\text{-}\Sigma_k\mathbf{Space}(s(n)) \subseteq \bigcup_c \mathbf{NSpace}(2^{2^{c \cdot s(n)}})$ *if* X *is a stack storage,* $k \geq 1$*, and* $s(n) \geq \log n$.

Sketch of Proof. Recall that the alternation bounded PDA Theorem of Ladner *et al.* [8, page 100–104, Section 5] shows that for constant k and $s(n) \geq \log n$, $\mathbf{AuxPD\text{-}}\Sigma_k\mathbf{Space}(s(n)) \subseteq \bigcup_c \mathbf{NSpace}(2^{c \cdot s(n)})$. The main idea in the lengthy argument used by Ladner *et al.* to prove their theorem is to generalize the notion of realizable pairs of surface configurations. A pair (ID, U), where ID is a surface configuration ID and U a *set* of (popping) surface configurations, is realizable if there is a 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, ID and all surface configurations in U have the same pushdown height, and the height of the pushdown does not go below this level during the computation. 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 case of an auxiliary stack automaton we can not 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 (reading the stack content). Thus, the behaviour 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.* [7, page 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 and entering push or pop mode, and the whole computation 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 bound 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 [4] and Szelepcsényi [12] we obtain our result. $\square$

3.2 Automata with Restricted Running Time

Now let‘s turn our attention to simultaneous space and time bounds. There we find the following situation, which can be shown by a phase-by-phase simulation, preserving the number of alternations on both sides. Due to the lack of space we omit the details.

Theorem 3. $\bigcup_c$ **AuxX-ASpaceTime**$(s(n), 2^{c\cdot s(n)}) = \bigcup_c$ **ATime**$(2^{c\cdot s(n)})$ *if* X *is a stack storage and* $s(n) \geq \log n$. $\square$

Since **PSpace** = **ATime**$(\mathrm{pol}\, n)$ we conclude:

Corollary 4. $\bigcup_c$ **AuxX-ASpaceTime**$(\log n, \mathrm{pol}\, n)$ = **PSpace** *if* X *is a stack storage.* $\square$

Since the simulation for the inclusion from right to left in the previous proof uses at least a nondeterministic auxiliary stack automaton we can state:

Corollary 5. $\bigcup_c \mathbf{Aux}\mathrm{X}\text{-}\mathbf{\Sigma_k SpaceTime}(s(n), 2^{c\cdot s(n)}) = \bigcup_c \Sigma_k \mathbf{Time}(2^{c\cdot s(n)})$ *if* X *is a stack storage,* $k \geq 1$*, and* $s(n) \geq \log n$. □

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 6. *Let* $s(n) \geq \log n$*. We have:*

1. $\bigcup_c \mathbf{Aux}\mathrm{X}\text{-}\mathbf{DSpaceTime}(s(n), 2^{c\cdot s(n)}) = \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})$ *if* X *is a stack or nonerasing stack.*
2. $\bigcup_c \mathbf{AuxCS\text{-}DSpaceTime}(s(n), 2^{c\cdot s(n)}) = \mathbf{DSpace}(s(n))$. □

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 [5, page 123, Theorem 3.4] in terms of auxiliary pushdown automata, where $\mathbf{AuxPD}\text{-}\Sigma_{k+1}\mathbf{SpaceTime}(\log n, \mathrm{pol}\, n) = \Sigma_k\mathbf{P}$ for $k \geq 1$ was shown. Note that [5] claims the case $k \geq 1$ of part 2 of the following corollary.

Corollary 7. *1.* $\mathbf{Aux}\mathrm{X}\text{-}\Sigma_k\mathbf{SpaceTime}(\log n, \mathrm{pol}\, n) = \Sigma_k\mathbf{P}$ *if* X *is a stack or nonerasing stack and* $k \geq 0$.
2. $\mathbf{AuxCS}\text{-}\Sigma_0\mathbf{SpaceTime}(\log n, \mathrm{pol}\, n) = \mathbf{DSpace}(\log n)$ *and the checking stack class* $\mathbf{AuxCS}\text{-}\Sigma_k\mathbf{SpaceTime}(\log n, \mathrm{pol}\, n) = \Sigma_k\mathbf{P}$ *if* $k \geq 1$. □

4 Empty Alternating Auxiliary Stack Automata

Lange and Reinhardt [9] exhibited 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 [9, page 501, Theorem 10].

Lemma 8. *Let* X *be a stack storage and* $s(n) \geq \log n$*. If* $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{NSpace}(s(n)) \subseteq \mathbf{C}_1$ *and* $\bigcup_c \mathbf{Aux}\mathrm{X}\text{-}\mathbf{NSpaceTime}(s(n), 2^{c\cdot s(n)}) \subseteq \mathbf{C}_2$*, for some* $\mathbf{C}_1$ *and* $\mathbf{C}_2$*, then*

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

4.1 Automata Without Runtime Restrictions

With the help of the above lemma we can now extend the argument in [9, Page 498, Theorem 5] and show that the empty alternation hierarchy for auxiliary stack automata collapses to its nondeterministic level. Again we have to omit the proof.

Theorem 9. $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{EASpace}(s(n)) = \mathbf{Aux}\mathrm{X}\text{-}\mathbf{NSpace}(s(n))$ *if* X *is a stack storage and* $s(n) \geq \log n$. □

For $s(n) = \log n$ we obtain from Theorem 9 the following corollary.

Corollary 10. *1.* $\mathbf{AuxS\text{-}EASpace}(\log n) = \bigcup_k \mathbf{DTime}(2^{n^k})$ *and*
2. $\mathbf{Aux}\mathrm{X}\mathbf{\text{-}EASpace}(\log n) = \mathbf{PSpace}$ *if* X *is a nonerasing or checking stack.* □

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 5 and Theorem 6. Moreover, for an unbounded number of alternations the upper bound was already settled in Lemma 8. For the lower bound, we find the following situation, where we have to distinguish between stack and nonerasing (checking) stack automata. We start with auxiliary stack automata and adapt an argument from [9, page 501, Theorem 10].

Lemma 11. *Let* $s(n) \geq \log n$ *and* $\mathbf{C} \subseteq \bigcup_c \mathbf{AuxS\text{-}NSpaceTime}(s(n), 2^{c\cdot s(n)})$, *then* $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]} \subseteq \bigcup_c \mathbf{AuxS\text{-}E\Sigma_2SpaceTime}(s(n), 2^{c\cdot s(n)})$. □

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 12. $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]} \subseteq \bigcup_c \mathbf{Aux}\mathrm{X}\mathbf{\text{-}E\Sigma_3SpaceTime}(s(n), 2^{c\cdot s(n)})$ *if* X *is a nonerasing or checking stack,* $s(n) \geq \log n$, *and* $\mathbf{C}$ *is contained in* $\bigcup_c \mathbf{Aux}\mathrm{X}\mathbf{\text{-}NSpaceTime}(s(n), 2^{c\cdot s(n)})$. □

The above two lemmata together with Lemma 8 show that for $s(n) \geq \log n$ we have sandwiched both $\bigcup_c \mathbf{AuxS\text{-}E\Sigma_2SpaceTime}(s(n), 2^{c\cdot s(n)})$ and the class $\bigcup_c \mathbf{Aux}\mathrm{X}\mathbf{\text{-}E\Sigma_3SpaceTime}(s(n), 2^{c\cdot s(n)})$, where X is a nonerasing or checking stack, in between the classes $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]}$ and $\bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})^{\mathbf{C}}_{||}$, with $\mathbf{C} = \bigcup_c \mathbf{NTime}(2^{c\cdot s(n)})$ because of Corollary 5. In case $s(n) = \log n$, these bounded query classes are known to be equal due to Wagner [14, page 838, Corollary 3.7.1]. Fortunately, this generalizes to arbitrary space bounds greater or equal than $\log n$, as a careful analysis of Wagner's proof reveals.

Theorem 13. $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]} = \bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})^{\mathbf{C}}_{||}$ *if* $s(n) \geq \log n$ *and* $\mathbf{C} = \bigcup_c \mathbf{NTime}(2^{c\cdot s(n)})$. □

As an immediate corollary we obtain:

Corollary 14. *Let* X *be a stack storage. If* $s(n) \geq \log n$ *and* $\mathbf{C}$ *is equal to* $\bigcup_c \mathbf{NTime}(2^{c\cdot s(n)})$, *then* $\bigcup_c \mathbf{Aux}\mathrm{X}\mathbf{\text{-}E\Sigma_2SpaceTime}(s(n), 2^{c\cdot s(n)})$ *and the class* $\bigcup_c \mathbf{Aux}\mathrm{X}\mathbf{\text{-}EASpaceTime}(s(n), 2^{c\cdot s(n)})$, *where* $k = 2$ *if* X *is a stack and* $k = 3$ *if* X *is a nonerasing or checking stack, coincide with* $\bigcup_c \mathbf{DTime}(2^{c\cdot s(n)})^{\mathbf{C}}_{||}$. □

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 [14, page 844, Theorem 8.1]. A similar result was obtained by Lange and Reinhardt [9, page 501, Theorem 10] for empty alternating polytime machines.

Corollary 15. *$\Theta_2\mathbf{P}$ is equal to $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{E}\Sigma_k\mathbf{SpaceTime}(\log n, \mathrm{pol}\, n)$ and the class $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{EASpaceTime}(\log n, \mathrm{pol}\, n)$, where X is a stack storage and $k = 2$ 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 16. $\bigcup_c \mathbf{Aux}\mathrm{X}\text{-}\mathbf{E}\Sigma_2\mathbf{SpaceTime}(s(n), 2^{c\cdot s(n)}) = \mathbf{NSpace}(s(n))^{\langle \mathbf{C}[1]\rangle}$ *if X is a nonerasing or checking stack storage, $s(n) \geq \log n$, and $\mathbf{C}$ is equal to $\mathbf{Aux}\mathrm{X}\text{-}\mathbf{NSpaceTime}(s(n), 2^{c\cdot s(n)})$.*

Proof. First we prove the inclusion from right to left. Let M be a nondeterministic $s(n)$ space bounded Turing machine with an oracle $A \in \mathbf{C}$ making exactly *one* oracle question. Further, let M_A and $M_{\bar{A}}$ be the auxiliary automaton 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 deterministically writing 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 universally alternating—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.

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

$$A_{\exists,M} = \{\, ID \in C(x) \mid M \text{ starting in } ID \text{ accepts with existential moves only} \,\}$$

and

$$A_{\forall,M} = \{\, ID \in C(x) \mid M \text{ starting in } ID \text{ accepts with universal moves only} \,\}.$$

For the simulation we consider two cases. Either M writes the first time to the stack in the existential or universal phase. Note that in the former case M can not alternate anymore. 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 "$ID_0 \in A_{\exists,M}$," 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 on question "$ID \in$ **co-**$A_{\forall,M}$" the answer is "yes," then M' rejects. Otherwise, it accepts. Since $A_{\exists,M}$ and complement of $A_{\forall,M}$ both belong to **C** our claim follows. $\square$

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

Corollary 17. $\mathbf{NL}^{\langle \mathbf{NP}[1] \rangle} = \mathbf{Aux}\mathrm{X}\text{-}\mathbf{E}\Sigma_2\mathbf{SpaceTime}(\log n, \mathrm{pol}\, n)$ *if* X *is a non-erasing or checking stack.* $\square$

5 Conclusions

Tables 1 and 2 summarize the known complexity class characterizations based on **AuxS-**, **AuxNES-**, and **AuxCS-** alternating and empty-alternating automata. In addition to the short-hand notations introduced in Section 2 we use **EDTime** (**EEDTime**, respectively) to denote the class $\bigcup_k \mathbf{DTime}(2^{n^k})$ ($\bigcup_k \mathbf{DTime}(2^{2^{n^k}})$, respectively) and also **ENSpace** as an abbreviation for the class $\bigcup_k \mathbf{NSpace}(2^{n^k})$.

Alternations	Auxiliary alternating Turing machine with storage X								
	none	PD-		S-		NES-		CS-	
		time bound		time bound		time bound		time bound	
		poly	without	poly	without	poly	without	poly	without
determ.	**L**	LOG(**DCFL**) [11]	**P** [2]	**P**	**EDTime** [3]	**P**	**PSpace** [3]	**L**	**L** [3]
Σ_1	**NL**	LOG(**CFL**) [11]		**NP**		**NP**		**NP**	**PSpace** [3]
Σ_2		**NP** [5]	**PSpace** [8]	Σ_2**P**	**EDTime** $\subseteq \cdot \subseteq$ **ENSpace**	Σ_2**P**	**PSpace** $\subseteq \cdot \subseteq$ **ENSpace**	Σ_2**P**	**PSpace** $\subseteq \cdot \subseteq$ **ENSpace**
Σ_3		Σ_2**P** [5]		Σ_3**P**		Σ_3**P**		Σ_3**P**	
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	
Σ_k	[4, 12]	Σ_{k-1}**P** [5]		Σ_k**P**		Σ_k**P**		Σ_k**P**	
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	
A	**P** [1]	**PSpace** [5]	**EDTime** [7]	**PSpace**	**EEDTime** [7]	**PSpace**	**EEDTime** [7]	**PSpace**	**EEDTime**

Table 1. Complexity classes of auxiliary $\log n$ space bounded alternating automata (shading represents results obtained in this paper).

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?

References

1. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.

<table>
<tr><th rowspan="4">Alternations</th><th colspan="9">Auxiliary *empty* alternating Turing machine with storage X</th></tr>
<tr><th rowspan="3">none</th><th colspan="2">**PD-**</th><th colspan="2">**S-**</th><th colspan="2">**NES-**</th><th colspan="2">**CS-**</th></tr>
<tr><th colspan="2">time bound</th><th colspan="2">time bound</th><th colspan="2">time bound</th><th colspan="2">time bound</th></tr>
<tr><th>poly</th><th>without</th><th>poly</th><th>without</th><th>poly</th><th>without</th><th>poly</th><th>without</th></tr>
<tr><td>determ.</td><td>**L**</td><td>LOG(**DCFL**) [11]</td><td rowspan="2">**P** [2]</td><td>**P**</td><td rowspan="2">**EDTime** [3]</td><td>**P**</td><td rowspan="2">**PSpace** [3]</td><td>**L**</td><td>**L** [3]</td></tr>
<tr><td>$\mathbf{E}\Sigma_1$</td><td rowspan="6">**NL** [4, 12]</td><td>LOG(**CFL**) [11]</td><td>**NP**</td><td>**NP**</td><td>**NP**</td><td>**PSpace** [3]</td></tr>
<tr><td>$\mathbf{E}\Sigma_2$</td><td rowspan="5">LOG(**CFL**) [9]</td><td rowspan="6">**P** [9]</td><td rowspan="6">$\Theta_2\mathbf{P}$</td><td rowspan="6">**EDTime**</td><td>$\mathbf{NL}^{\langle \mathbf{NP}[1] \rangle}$</td><td rowspan="6">**PSpace**</td><td>$\mathbf{NL}^{\langle \mathbf{NP}[1] \rangle}$</td><td rowspan="6">**PSpace**</td></tr>
<tr><td>$\mathbf{E}\Sigma_3$</td><td rowspan="5">$\Theta_2\mathbf{P}$</td><td rowspan="5">$\Theta_2\mathbf{P}$</td></tr>
<tr><td>$\vdots$</td></tr>
<tr><td>$\mathbf{E}\Sigma_k$</td></tr>
<tr><td>$\vdots$</td></tr>
<tr><td>**EA**</td><td>**P** [1]</td><td>**P** [9]</td></tr>
</table>

Table 2. Complexity classes of auxiliary $\log n$ space bounded *empty* alternating automata (shading represents results obtained in this paper).

2. S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, January 1971.
3. O. H. Ibarra. Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata. *Journal of Computer and System Sciences*, 5(2):88–117, April 1971.
4. N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, October 1988.
5. B. Jenner and B. Kirsig. Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata. In *Proceedings of the* 5*th Annual Symposium on Theoretical Computer Science*, number 294 in LNCS, pages 118–125, Bordeaux, France, February 1988. Springer.
6. R. Ladner and N. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.
7. R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, February 1984.
8. R. E. Ladner, L. J. Stockmeyer, and R. J. Lipton. Alternation bounded auxiliary pushdown automata. *Information and Control*, 62:93–108, August/September 1984.
9. K.-J. Lange and K. Reinhardt. Empty alternation. In *Proceedings of the* 9*th Conference on Mathematical Foundations of Computer Science*, number 841 in LNCS, pages 494–503, Kosice, Slovakia, August 1994. Springer.
10. W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, April 1984.
11. I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, July 1978.
12. R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.
13. K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its applications (East Europeans series). VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.
14. K. W. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, October 1990.