On the Complexities of Linear LL(1) and LR(1) Grammars*

Markus Holzer Klaus-Jörn Lange

Fakultät für Informatik, Technische Universität München, Arcisstr. 21, D–80290 München, Germany

Abstract

Several notions of deterministic linear languages are considered and compared with respect to their complexities and to the families of formal languages they generate. We exhibit close relationships between *simple* linear languages and the *deterministic linear languages* both according to Nasu and Honda and to Ibarra, Jiang, and Ravikumar. Deterministic linear languages turn out to be special cases of languages generated by linear grammars restricted to LL(1) conditions, which have a membership problem solvable in $\mathcal{NC}^1$. In contrast to that, deterministic linear languages defined via automata models turn out to have a $DSPACE(\log n)$-complete membership problem. Moreover, they coincide with languages generated by linear grammars subject to LR(1) conditions.

1 Introduction

There are many close connections between complexity theory and the area of formal languages and automata. Typical relations of this kind are completeness results for families of formal languages. Well known examples are:

1. $\mathcal{CFL}$, the family of context-free languages, is $NAuxPDA\text{-}TIMESPACE(\mathrm{pol}, \log n)$-complete [17].
2. $\mathcal{DCFL}$, the family of deterministic context-free languages, is complete for the class $DAuxPDA\text{-}TIMESPACE(\mathrm{pol}, \log n)$ [17].
3. $\mathcal{LIN}$, the family of linear context-free languages, is $NSPACE(\log n)$-complete [16].
4. $\mathcal{REG}$, the family of regular languages, is $\mathcal{NC}^1$-complete [2].

Here we have to use very "fine" notions of reducibility in order to treat low-level complexity classes like $\mathcal{NC}^1$ or $\mathcal{AC}^0$. With respect to the usual logspace reductions the regular languages, for instance, would trivially be $DSPACE(\log n)$-complete.

This brings us to the aim of this paper: to extend this list with an entry for the class $DSPACE(\log n)$. One natural idea would be to conclude the commuting diagram of Figure 1.

But Ibarra et al. introduced in [10] a family of *deterministic linear languages* by imposing LL(1)-like conditions on linear context-free grammars and, unfortunately, showed

*This work was supported by DFG-La 618/1-1.

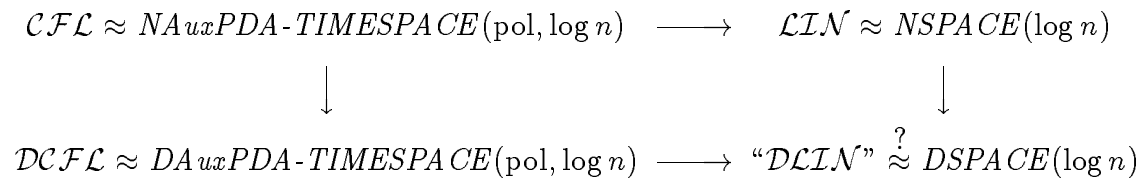

Figure 1: A commuting diagram?

containment in $\mathcal{NC}^1$. Since deterministic linear languages contain $\mathcal{REG}$, this implies the $\mathcal{NC}^1$-completeness. But if we look again at Figure 1, we see that for (arbitrary) context-free languages determinism is defined via the corresponding automaton model and not via a grammar type. This makes no difference for context-free languages; context-free LR(1) grammars generate exactly the deterministic context-free languages and context-free LL(1)-languages, although being a proper subfamily, nevertheless are complete for the class $DAuxPDA\text{-}TIMESPACE(\mathrm{pol}, \log n)$ (see [17]).

In the following we will show that in the linear case both the automata and the LR(1) approach yield the validity of Figure 1. This will be done by considering *one-turn* pushdown automata [6] which execute no push move after the first pop operation occurred in a computation. It is well known that these automata characterize the linear context-free languages: $\mathcal{NPDA}_{1\text{-}turn} = \mathcal{LIN}$ [6]. First of all, we prove that $\mathcal{DPDA}_{1\text{-}turn}$, the deterministic counterpart of $\mathcal{NPDA}_{1\text{-}turn}$, coincides with the family of languages generated by linear context-free grammars restricted by an LR(1) condition. More important, we will show the $DSPACE(\log n)$-completeness of $\mathcal{DPDA}_{1\text{-}turn}$. These results demonstrate that the complexity theoretical equivalence of LR(1) and LL(1) conditions of [17] are no longer valid in the presence of linear grammars, since here their equivalence would imply $\mathcal{NC}^1 = DSPACE(\log n)$. Altogether, we think that this demonstrates that the definition of Ibarra et al. in [10] does not give an adequate notion of a *deterministic linear language*. Thus, throughout this paper we will use the term $\mathcal{DLIN}$ for languages generated by linear grammars subjected to an LR(1) condition. This will be justified by the coincidence of this class with $\mathcal{DPDA}_{1\text{-}turn}$, the class of languages accepted by deterministic one-turn pushdown automata.

It is remarkable to find these relations again when considering one-counter instead of one-turn automata. It is well known that $\mathcal{ROCL}$, the family of languages recognized by nondeterministic one-counter automata [7], i.e.: pushdown automata with a singleton pushdown alphabet, is $NSPACE(\log n)$-complete (see e.g. [18]). It is quite easy to modify our construction concerning the $DSPACE(\log n)$-completeness of $\mathcal{DPDA}_{1\text{-}turn}$ into a deterministic one-counter automaton accepting a $DSPACE(\log n)$-complete set, which implies the $DSPACE(\log n)$-completeness of $\mathcal{D}\text{-}\mathcal{ROCL}$. In this context it is interesting that Chang et al. showed in [4] that the the simultaneous restriction of a deterministic pushdown automaton to be both one-turn and one-counter makes the resulting word problem $\mathcal{NC}^1$-recognizable.

In the next section we introduce the necessary notions. In Section 3 we prove a close relationship between simple linear and deterministic linear languages defined by Nasu and Honda [15] and Ibarra, Jiang, and Ravikumar [10]. In Section 4 we show that linear

LL(1) languages and deterministic one-turn counter languages are $\mathcal{NC}^1$-complete. The main result is given in Section 5. There we show that $\mathcal{DLIN}$, the class of languages accepted by deterministic one-turn pushdown automata, is $DSPACE(\log n)$-complete. Moreover this class coincides with the class of languages generated by linear LR(1) grammars. In addition we get that $\mathcal{D}$-$\mathcal{ROCL}$, the class of deterministic counter languages, is $DSPACE(\log n)$-complete, too. Open questions are discussed in the last section.

2 Preliminaries

The reader is assumed to be familiar with basic notions of formal language (see e.g. [9]) and complexity theory (see e.g. [1, 18]). In addition we use the following notations:

Definition 1 A context-free grammar $G = (N, T, P, S)$ is *linear* if and only if all productions in P are of the form:

$$A \to u_1 B v_1 \mid u_2$$

for $A, B \in N$, and $u_1, u_2, v_1 \in T^*$.

In the following languages generated by linear grammars are referred to linear languages, and the class of linear languages to $\mathcal{LIN}$.

Let λ denote the string of length zero. We say that G is λ-*free* if and only if it has no productions of the form $A \to \lambda$. A string of $\alpha \in (N \cup T)*$ is called a *sentential form* if $S \overset{*}{\Rightarrow} \alpha$, where $\overset{*}{\Rightarrow}$ denotes the usually derivation relation.

To characterize the complexity of linear languages, we deal with the complexity classes $\mathcal{DLOGTIME}$, $\mathcal{NC}^1$, and $DSPACE(\log n)$. For a definition of this classes see, e.g. [1].

To study the relative complexity of problems it is helpful to use very weak reductions. Completeness are always meant with respect to $\mathcal{DLOGTIME}$ reducibilities, like they were introduced by Buss in [3], unless otherwise stated.

3 Deterministic Linear Languages

In this section we consider the definitions of deterministic linear languages of Nasu and Honda and Ibarra, Jiang, and Ravikumar and show a close relation to simple linear languages.

Definition 2 [15] A linear grammar $G = (N, T, P, S)$ is *NH-deterministic linear* if and only if all productions in P are of the form:

$$A \to aBu \mid a$$

for $A, B \in N$, $a \in T$, $u \in T^*$, and furthermore for any $A \in N$ and $a \in T$, there is at most one production $A \to a\alpha$ for $\alpha \in NT^* \cup \{\lambda\}$ in P.

In the following languages generated by NH-deterministic linear grammars are referred to NH-deterministic linear languages, and the class of NH-deterministic linear languages to $\mathcal{DLIN}_{NH}$.

Remark: A context-free grammar is a *simple* grammar [12] if and only if all productions are of the form

$$A \to a_1\alpha_1 \mid \ldots \mid a_n\alpha_n$$

for $a_i \in T$, $\alpha_i \in (N \cup T)^*$ with $1 \leq i \leq n$ then $a_i \neq a_j$ for $i \neq j$. Obviously, a grammar G is NH-deterministic linear if and only if G is a linear and a simple grammar.

Note that in [10] Ibarra, Jiang, and Ravikumar defined a notion of linear determinism as well. A language L is *IJR-deterministic linear* if and only if it is generated by some context-free grammar $G = (N, T, P, S)$ with the property, that any of the three productions

$$A \to u_1B_1v_1 \mid u_2B_2v_2 \mid u_3$$

for $A, B_1, B_2 \in N$ and $u_1, u_2, u_3, v_1, v_2 \in T^*$ are in P, then u_i is not a prefix of u_j for $i \neq j$. The class of IJR-deterministic linear languages is referred to $\mathcal{DLIN}_{IJR}$.

Theorem 3 $\mathcal{DLIN}_{NH} = \mathcal{DLIN}_{IJR} \setminus \{\{\lambda\}\}$.

Sketch of Proof: "$\subseteq$" Obviously. "$\supseteq$" First we use the fact, that for a IJR-deterministic linear language L the following hold: $\lambda \in L$ if and only if $L = \{\lambda\}$. Therefore $S \to \lambda$ does not occur in the production set of the grammar. Now with standard techniques one can construct a λ-free IJR-deterministic linear grammar and then out of this a simple linear one. Then the claim obviously follows. □

4 The Complexity of Linear LL(1) Languages

In this section we show that NH-deterministic linear languages coincide with λ-free linear LL(1) languages, and that there exists a $\mathcal{NC}^1$-complete linear LL(1) language. The notion of LL(1) grammars was introduced by Lewis and Stearns [14].

Definition 4 [14] A linear grammar $G = (N, T, P, S)$ is *linear LL(1)* if and only if for every nonterminal $A \in N$ and all sentential forms $S \stackrel{*}{\Rightarrow} uAv$, with $u, v \in T^*$, and all distinct productions $A \to \alpha_1$ and $A \to \alpha_2$ in P the condition

$$FIRST_G(\alpha_1 v) \cap FIRST_G(\alpha_2 v) = \emptyset$$

holds, where $FIRST_G(\alpha)$ is defined as

$$FIRST_G(\alpha) := \{\, a \in T \mid \alpha \stackrel{*}{\Rightarrow} a\beta \text{ for some } \beta \in (N \cup T)^* \,\} \cup \{\, \lambda \mid \alpha \stackrel{*}{\Rightarrow} \lambda \,\}$$

In the following languages generated by (λ-free) linear LL(1) grammars are referred to (λ-free) linear LL(1) languages, and the class of (λ-free) linear LL(1) languages to (λ-free) $\mathcal{LINLL}(1)$.

Using a result of Kurki-Suonio [13], we obtain:

Theorem 5 $\mathcal{DLIN}_{NH} = \lambda$-free $\mathcal{LINLL}(1)$.

Proof: "$\subseteq$" By definition it is clear that every NH-deterministic linear language is λ-free linear LL(1). "$\supseteq$" Using a result of Kurki-Suonio [13] we know that every λ-free LL(1) language generates a simple language. Since this result holds true for linear languages, and simple linear languages are obviously NH-deterministic linear, the claim follows. $\Box$

Remark: Obviously $\mathcal{DLIN}_{NH} \subseteq \mathcal{DLIN}_{IJR} \subseteq \mathcal{LINLL}(1)$ and each inclusion is a proper one, which is shown by the following languages: $\{\lambda\} \in \mathcal{DLIN}_{IJR} \setminus \mathcal{DLIN}_{NH}$ and $\{w \mid w \in \{0,1\}^*\} \in \mathcal{LINLL}(1) \setminus \mathcal{DLIN}_{IJR}$.

The aim of this paper is to characterize complexity classes in terms of linear languages, i.e. we are interested in completeness results. In [10] it was shown that $\mathcal{DLIN}_{IJR} \subseteq \mathcal{NC}^1$, but a carefully analysis shows that this inclusion holds for linear LL(1) languages as well.

Theorem 6 [10] $\mathcal{LINLL}(1) \subseteq \mathcal{NC}^1$.

Now for $\mathcal{NC}^1$-completeness it suffices to use the result of Barrington, Immerman and Straubing.

Theorem 7 [2] There is a regular language which is $\mathcal{NC}^1$-complete[1].

By Theorem 6, 7 and the obvious fact that every regular language is linear LL(1) we obtain

Theorem 8 There is a linear LL(1) language which is $\mathcal{NC}^1$-complete.

Corollary 9 $\mathcal{LINLL}(1)$ is $\mathcal{NC}^1$-complete.

Remark: The result of Theorem 8 and 9 holds for both NH-deterministic linear languages and for IJR-deterministic linear languages, since the regular language which is $\mathcal{NC}^1$-complete is contained in $\mathcal{DLIN}_{NH}$. Note that regular and NH-deterministic linear languages are incomparable.

Remark: In [4] it was shown that $\mathcal{D}\text{-}\mathcal{ROCL}_{1\text{-}turn}$, the class of deterministic one-turn counter languages, belongs to $\mathcal{NC}^1$. The arguments of Theorem 8 apply here as well and hence these languages are $\mathcal{NC}^1$-complete, too.

5 The Complexity of Linear LR(1) Languages

In this section we study the complexity of linear LR(1) languages. We show that this class of languages coincides with the class of languages acceptable by deterministic one-turn pushdown automata, and that there exists a $DSPACE(\log n)$-complete linear LR(1) language. The notion of LR(1) grammars was introduced by Knuth [11].

[1] Usually the result was shown for $\mathcal{AC}^0$-reductions, but it is possible to use $\mathcal{DLOGTIME}$ ones.

Definition 10 [9, 11] A linear grammar $G = (N, T, P, S)$ is *linear LR(1)* if and only if the following condition holds for every string $u\alpha a w_1$, with $u, w_1 \in T^*$, $\alpha \in (N \cup T)^*$, and $a \in T$, such that $S\$ \stackrel{*}{\Rightarrow} u\alpha a w_1$[2]. If the next to the last step of the above derivation is $uAaw_1$, so that $S\$ \stackrel{*}{\Rightarrow} uAaw_1 \Rightarrow u\alpha a w_1$ and there is some other word $u\alpha a w_2$ with $w_2 \in T^*$ such that $S\$ \stackrel{*}{\Rightarrow} vBw \Rightarrow u\alpha a w_2$ then $u = v$, $A = B$, and $w = aw_2$.

In the following languages generated by linear LR(1) grammars are referred to deterministic linear languages, and the class of deterministic linear languages to $\mathcal{DLIN}$.

The second class of linear languages investigated in this section is a restriction of the class of linear languages $\mathcal{LIN}$ resp. of the class of languages acceptable by nondeterministic one-turn pushdown automata.

Definition 11 A language L is a *deterministic one-turn pushdown language* if and only if L is accepted by some one-way deterministic one-turn pushdown automata. Throughout this paper the class of deterministic one-turn pushdown languages is referred to $\mathcal{DPDA}_{1\text{-}turn}$.

The following proposition shows that the class of deterministic linear languages and the class of deterministic one-turn pushdown languages coincide.

Proposition 12 $\mathcal{DLIN} = \mathcal{DPDA}_{1\text{-}turn}$.

Sketch of Proof: "$\mathcal{DLIN} \subseteq \mathcal{DPDA}_{1\text{-}turn}$" In [9], Theorem 12.8, it was shown, that every context-free LR(k) language, is accepted by some deterministic pushdown automata M. If we use a linear[3] instead of a context-free language, it is easy to see that the only part why M is not a one-turn pushdown is, because between two pop moves the pushdown height could increase by several push moves. Since the number of push moves between two pop moves is bounded by l, where l is the longest right hand side of a production in the linear LR(1) grammar, one can show that M can be modified in such a way, by handling a constant number of pushdown symbols in the finite control of M, that M works as an one-turn pushdown.

"$\mathcal{DLIN} \supseteq \mathcal{DPDA}_{1\text{-}turn}$" In [9], Theorem 12.9, it was shown that every language L which is accepted by a deterministic pushdown automaton, is generated by a LR(1) grammar. This grammar is constructed according to the standard method for constructing a context-free grammar from a pushdown automaton. If we use a one-turn pushdown instead of a pushdown automaton, the standard method leads us to a context-free grammar instead of a linear one. Therefore we construct a linear grammar G from the one-turn pushdown according to the method used in e.g. [8], Theorem 5.7.1. Now using the same arguments as in [9], Theorem 12.9, one can show that G is linear LR(1). □

The following two propositions establish the $DSPACE(\log n)$-completeness of $\mathcal{DLIN}$.

Proposition 13 $\mathcal{DLIN} \subseteq DSPACE(\log n)$.

[2] Let \$ be a new symbol which belongs to T.

[3] Note that at each step of M there is at most one symbol on the pushdown which corresponds to a nonterminal in the LR(1) grammar. This fact is essential for the proof.

Proof: Let L be a deterministic linear language. Then by Proposition 12 there exists a one-way deterministic one-turn pushdown automata A with $L = L(A)$[4]. The Turing machine M that simulates A works on input w as follows:

It starts the simulation in the initial configuration of A. Then M can directly simulate A only by knowing the surface configuration[5] and the pushdown height of A, as long as A makes no pop move. After A has made a pop move, M computes the new top symbol of the pushdown by a resimulation of A up to the current pushdown height. The pushdown symbol in the reached surface configuration will become the new top symbol of the pushdown and the simulation of A can be continued. If during the whole simulation A accepts w, M halts and accepts.

It is easy to see that $L(M) = L(A)$ and that M is a deterministic logspace bounded Turing machine. □

Before we come to the next proposition, we consider a restricted version of a well known $DSPACE(\log n)$-complete problem.

Definition 14 The graph accessibility problem for ordered graphs with out-degree one, for short 1GAP_o, is defined as the set of all codings $\langle G \rangle$ of graphs $G = (V, E)$ with the properties:

1. There exists a $k \in \mathbb{N}_0$ such that $V = \{1, 2, \ldots, 2^k\}$.
2. The graph has out-degree one; This means that if $(i, j) \in E$ and $(i, l) \in E$ then $j = l$.
3. The graph is ordered; This means that if $(i, j) \in E$ then $i < j$ for $1 \leq i \leq 2^k - 1$ and $i = j$ for $i = 2^k$.
4. In the graph there is way from the node 1 to the node 2^k; This means that there exists $i_1, i_2, \ldots, i_t \in V$ such that $i_1 = 1$, $i_t = 2^k$, and $(i_j, i_{j+1}) \in E$ for all $1 \leq j \leq t - 1$.

By Cook and McKenzie [5] we obtain:

Theorem 15 [5] 1GAP_o is $DSPACE(\log n)$-complete[6].

In the definition of 1GAP_o, we left open the coding $\langle G \rangle$ of a graph G. Since every vertex has at most one successor, one can code the graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, 2^k\}$ for some $k \in \mathbb{N}_0$, by the string

$$\langle G \rangle := block(1)\$^{2^k} block(2)\$^{2^k} \ldots \$^{2^k} block(2^k)\#^{2^k}$$

where $block(i)$ for $i \in V$ is defined as $block(i) := 0^{2^k - j}1^j$ if $(i, j) \in E$, and $block(i) := 0^{2^k}$ otherwise.

Now we can show that there is a deterministic linear language which is hard for $DSPACE(\log n)$.

---

[4] We can assume that A changes the pushdown height during one step at most by one symbol.

[5] A *surface configuration* of a pushdown automata consists of the position of the input head, the internal state, and the top symbol of the pushdown.

[6] Usually the result was shown for $\mathcal{NC}^1$-reducibility and for graphs without the restriction that there exists a $k \in \mathbb{N}_0$ such that $V = \{1, 2, \ldots, 2^k\}$. But a careful analysis shows that 1GAP_o is still $DSPACE(\log n)$-complete, and it is possible to use $\mathcal{DLOGTIME}$ instead of $\mathcal{NC}^1$-reducibility.

Proposition 16 $1\mathrm{GAP}_o$ is $\mathcal{DLOGTIME}$-reducible to a deterministic linear language.

Proof: On input $w = block(1)\$^{2^k} \ldots \$^{2^k} block(2^k)\#^{2^k}$ let f be the mapping[7]:

$$f(w) := ((\overline{block}(1)\$^{2^k} \ldots \$^{2^k}\overline{block}(2^k)\#^{2^k})^R)^{2^k}\&^{2^k}0^{2^k}\$^{2^k}0^{2^k-1}1\$^{2^k} \ldots \$^{2^k}0^2 1^{2^k-2}\#^{2^k},$$

where $\overline{block}(i)$ for $i \in V$ is defined as $\overline{block}(i) := 0^{2^k-j+i}1^{j-i}$ if $(i,j) \in E$, and $\overline{block}(i) := 0^{2^k}$ otherwise. To show that $1\mathrm{GAP}_o$ is reducible to a deterministic linear language via f, we construct a deterministic one-turn pushdown M such that $w \in 1\mathrm{GAP}_o$ if and only if $f(w) \in L(M)$ holds.

On an input $f(w)$, M starts by pushing the symbols, which are scanned by the input head, until the first & symbol is reached. Then the & symbols are over read, and from then on, it goes into a loop where it operates in stages. It (1) uses the information in the words $\overline{block}(\cdot)$ in the pushdown to position the input head. More precisely, for every 1 symbol in the topmost $\overline{block}(\cdot)$ the input head reads the word $\{0,1\}^*\$^{2^k}$. If during this stage the word $\{0,1\}^*\#^{2^k}$ is read with the last 1 symbol from the topmost $\overline{block}(\cdot)$ the machine M halts and accepts. Otherwise M continues by (2) popping the word $\{0,1,\$\}^*\#^{2^k}$ and (3) using the information in the word $\{0\}^*\{1\}^*$ in the rest of the input to position the pushdown head. More precisely, for every 1 symbol in the currently by the input head scanned word $\{0\}^*\{1\}^*$, it pops the word $\{0,1\}^*\$^{2^k}$. After (4) the last 1 symbol was read by the input head, it scans the afterward word $\$^{2^k}$. If during this computation the pushdown gets empty, or the input head moves to the right of the last # symbol, or M discovers any syntactic inconsistency in the input, it halts and rejects.

It is evident that M accepts $f(w)$ if and only if $w \in 1\mathrm{GAP}_o$ since the topmost words $\overline{block}(\cdot)$ in stage (1) can be used to describe the ordered way through the coded graph w from the node 1 to the node 2^k.

Furthermore, $f(w)$ is polynomially bounded and and $\mathcal{DLOGTIME}$-computable since computing the length of the input in binary representation (see [3]), addition, subtraction of *small* numbers, i.e. polynomially bounded numbers, and multiplication, division of small numbers where the multiplicand or the divisor is a power of two, can be done by a deterministic Turing machine in time $O(\log n)$. □

Remark: By Proposition 12 it is possible to construct a linear LR(1) grammar instead of a deterministic one-turn pushdown automaton. The linear LR(1) grammar $H = (\{A, B, \ldots, L\}, \{0, 1, \$, \#, \&\}, P, A)$ which can be constructed instead of the one-turn pushdown has the productions[8]:

$A \to 0A \mid 1A \mid \$A \mid \#A \mid \$B\#$	$B \to \$B\# \mid \$1C\#$	$C \to C0 \mid C1 \mid D0 \mid D1$
$D \to 0D \mid 1E\$ \mid \$F\$ \mid K\&$	$E \to E\$ \mid C\$$	$F \to \$F\$ \mid \$G1\$$
$G \to 0G \mid 1G \mid 0H \mid 1H$	$H \to H0 \mid \$I \mid \#L$	$I \to \$I \mid \$G1$
$J \to 0J \mid 1J \mid \$J \mid \$D$	$K \to K\& \mid \&$	$L \to \#L \mid \#J0$

Now the following theorem can be directly obtained from Proposition 13, and 16:

[7] Let w^R denote the *mirror* image of w.

[8] We did not check H to have the LR(1) condition. Instead we used the UNIX[9] tools *yacc* and *bison*, which stated that H in fact fulfills the stonger LALR(1) property.

[9] UNIX is a registrated trademark of AT&T.

Theorem 17 There is a deterministic linear language which is $DSPACE(\log n)$-complete, moreover the class $\mathcal{DLIN}$ is $DSPACE(\log n)$-complete.

Remark: Obviously deterministic counter machines can be simulated by deterministic logspace bounded Turing machines (see also the remark after Corollary 9). With the same idea as in the proof of Proposition 16 one can show that the class of deterministic counter languages, for short $\mathcal{D}$-$\mathcal{ROCL}$, is $DSPACE(\log n)$-complete, too. Note that $\mathcal{ROCL}$, the class of nondeterministic counter languages, is $NSPACE(\log n)$-complete (see e.g. [18]), but for nondeterministic one-turn counter languages a completeness result, like in the deterministic case, remains open.

6 Conclusions

We introduced the notion of a *deterministic linear language* via LR(1) restrictions of linear grammars. The adequacy of this concept is reflected by its equivalence with $\mathcal{DPDA}_{1\text{-}turn}$. This approach leads to family of deterministic linear languages with a $DSPACE(\log n)$-complete word problem, which fits in nicely into the framework of completeness-results like [16, 17] (see Figure 1). In contrast to that both the deterministic linear languages according to Ibarra, Jiang, and Ravikumar [10] and those according to Nasu and Honda [15] are based on linear grammars restricted by an LL(1) condition. The word problem corresponding to this approach leads to apparently simpler complexities, unless we believe in $\mathcal{NC}^1 = DSPACE(\log n)$. Thus contrasting the case of general context-free grammars, LL conditions seem to be too restrictive in connection with linear grammars, where they spoil the "sequential" character of linear languages as it is expressed by the $NSPACE(\log n)$-completeness of $\mathcal{LIN}$ [16] or the $DSPACE(\log n)$-completeness of $\mathcal{DLIN}$.

One-counter languages are similar to linear languages with respect to complexity. This is expressed by the well-known $NSPACE(\log n)$-completeness of $\mathcal{ROCL}$ (see e.g. [18]) and the $DSPACE(\log n)$-completeness of $\mathcal{D}$-$\mathcal{ROCL}$ which follows along the lines of Proposition 16. We leave it as an open question to find grammatical representations of this fact using LR (or LL) conditions. In this connection it should be remarked that Chang, Ibarra, Jiang, and Ravikumar [4] put the membership problem of $\mathcal{D}$-$\mathcal{ROCL}_{1\text{-}turn}$ into $\mathcal{NC}^1$. This leaves open the complexity of $\mathcal{ROCL}_{1\text{-}turn}$.

7 Acknowledgments

The authors would like to thank U. Vollath for helpful comments on the UNIX[9] tools *yacc* and *bison*.

References

[1] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[2] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[3] S. R. Buss. The boolean formula value problem is in ALOGTIME. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 123–131, 1987.

[4] J. H. Chang, O. H. Ibarra, T. Jiang, and B. Ravikumar. Some classes of languages in $\mathcal{NC}^1$. *Information and Computation*, 90:86–106, 1991.

[5] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987.

[6] S. Ginsburg and E. H. Spanier. Finite-turn pushdown automata. *SIAM Journal on Computing*, 4(3):429–453, 1966.

[7] S. A. Greibach. An infinite hierarchy of context-free languages. *Journal of the Association for Computing Machinery*, 16:91–106, 1969.

[8] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

[9] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata.* Addison-Wesley, 1968.

[10] O. H. Ibarra, T. Jiang, and B. Ravikumar. Some subclasses of context-free languages in $\mathcal{NC}^1$. *Information Processing Letters*, 29:111–117, October 1988.

[11] D. E. Knuth. On the translation of languages from left to right. *Information and Control*, 8:607–639, 1965.

[12] A. Korenjak and J. Hopcroft. Simple deterministic languages. In *IEEE Conference Record 7th Annual Symposium Switching Automata Theory*, pages 36–46, 1966.

[13] R. Kurki-Suonio. Notes on top-down languages. *Bit*, 9:225–238, 1969.

[14] P. Lewis and R. Stearns. Syntax-directed transduction. *Journal of the Association for Computing Machinery*, 15:465–488, 1968.

[15] M. Nasu and N. Honda. Mappings induced by PGSM-mappings and some recursively unsolvable problems of finite probabilistic automata. *Information and Control*, 15:250–273, 1969.

[16] I. H. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *Journal of the Association for Computing Machinery*, 22(4):499–500, October 1975.

[17] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the Association for Computing Machinery*, 25:405–414, 1978.

[18] K. Wagner and G. Wechsung. *Computational Complexity.* Mathematics and its applications (East Europeans series). VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.