Linear languages recognition in deterministic polynomial time and sub-linear space.

Aleksander Mądry*

Abstract: *This paper addresses the problem of context-free language's recognition. Namely, recognition of CFL in the minimum possible space but mantaining still polynomial time. The best known result is the well-known CYK [2] [3] [4] algorithm which by usage of dynammic programming paradigm, works in $O(n^2)$ space. On the other hand, as first shown by Cook [5], deterministic context-free languages can be recognized in polynomial time and polylogarithmical space. Therefore, a question arises whether this huge gap between CFL and DCFL can be tightened. As an attempt to approach it, we consider this problem with respect to a strict subset of CFL which is still nondeterministic - the linear languages. In this case just a straight-forward adjustment of CYK algorithm yields an algorithm which uses only $O(n)$ space. Thus we are interested in investigation whether this linear space bound can be broken while still using only polynomial-time. We present an algorithm which solves the problem using $O(\frac{n^{2+\delta}}{\log n})$ time and $O(\frac{n}{\log n})$ space, where δ can be any positive constant. Thus it is possible to break the space bound imposed by, the only known so far, dynamic programming approach.*

1 Introduction

There has been much attention in computer science in the problem of context-free language recognition. And one of the fundamental questions in this context are the bounds on resources needed in order to solve it. Lewis et al. [6] described a deterministic algorithm which performs this task using only $\log^2 n$ space, but super-polynomial time. His algorithm is based on divide&conquer method which allows finding derivation by cutting off majority of possibilities thus reducing the space needed to keep actual configuration. On the other hand, there is a result of Valiant [7] which using sophisticated algebraic approach yields an algorithm solving this problem in time needed to multiply two $n \times n$ Boolean matrices. However, when we are interested in algorithm that minimizes the space used and is still polynomial in time, then the best result known is an dynamic programming algorithm by Cocke-Younger-Kasami [2] [3] [4] using $O(n^2)$ space.

But when we restrict ourselves to the deterministic context-free languages then, as first shown by Cook [5], there is an algorithm which runs in polynomial time and $O(\log^2 n)$ space simultaneously, putting DCFL in the class SC of languages recognizable simultaneously in polynomial time and polynomial in $\log n$ space. This result was

*Institute of Computer Science, University of Wrocław. E-mail address: a.madry@psz.pl.

extended by showing an optimal time-space tradeoff [1] for DCFL's recognition. Therefore, a natural question arises whether this huge gap between CFL and DCFL can be tightened. Unfortunately, at this time no such result is known. So, in an attempt to approach it, we consider this problem with respect to a strict subset of CFL which is still nondeterministic, the linear languages. This are those context-free languages which have at most one non-terminal at the right side of each production, which makes them much easier to recognize. As a result, just a straight-forward adjustment of CYK [2] [3] [4] algorithm yields an algorithm which uses only $O(n)$ space and $O(n^2)$ time. Thus we are interested in investigation whether this linear space bound can be broken while still using only polynomial-time. We present an algorithm which solves the problem in $O(\frac{n^{2+\delta}}{\log n})$ time and $O(\frac{n}{\log n})$ space, where δ can be any positive constant. This result can be seen as the first step towards showing that either linear languages (or even general CFLs) are in SC or posing a lower bound on the space needed in such polynomial deterministic calculations.

2 Preliminaries

We first define the linear languages.

Definition 1. Linear language *is a context-free language over an alphabet $\mathcal{A}$, which can be recognized by a* **linear grammar** *i.e. a grammar $G = \langle \mathcal{A}, N, R, S\rangle$ where N is a set of nonterminals, S is a starting nonterminal and R is the set of productions such that each production $\gamma \in R$ has at most one nonterminal on the right side.*

However, from our point of view it is more convenient to utilize the following straightforward observation.

Observation 2. *If $G = \langle \mathcal{A}, N, R, S\rangle$ is a linear grammar then we can assume that R is the set of productions of the form:*

$X \longrightarrow aY$ *(***left** *production)*

$X \longrightarrow Ya$ *(***right** *production)*

$X \longrightarrow a$ *(***ending** *production)*

where $X, Y \in N$ and $a \in \mathcal{A}$.

Throughout the whole paper G, $\mathcal{A}$, N, R, S will always have the above meaning and by $w = w_1 \dots w_n$ we will mean the word over $\mathcal{A}$ being the input of the algorithm.

We use a convention that for a word $v = v_1 \dots v_p$, $v_p \dots v_q = \varepsilon$ if $p > q$ and $v_0 = \varepsilon$.

We should note that any stage of some derivation $S \Rightarrow^* w$ of the word w in G can be represented by a **phrase** $\alpha = w_1 \dots w_k X w_l \dots w_n$, where $k < l$ and $X \in N \cup \{\varepsilon\}$. Having such phrase α in mind, we call the word $w_1 \dots w_k$ **the left side of w (with respect to α)**, the word $w_l \dots w_n$ - **the right side of w (with respect to α)** and if $X = \varepsilon$ then w_l is the **middle of w (with respect to α)**. Moreover, we say that a production γ from R **extends** the phrase $\alpha = w_1 \dots w_k X w_l \dots w_n$ iff either

- $\gamma = X \to w_{k+1}Y$ or $\gamma = X \to Y w_{l-1}$ for some $Y \in N$ and $k+1 < l$; or
- $\gamma = X \to w_{k+1}$ and $k+2 = l$.

In other words, a production γ extends a phrase α iff γ can be applied to α and the result is still a phrase.

Accordingly, we say that a phrase β **extends** the phrase α iff $\alpha \Rightarrow^* \beta$, i.e., there exists some (possibly empty) sequence of production whose consecutive application derives β from α.

3 The algorithm

We can think of finding the derivation of a word w in a grammar G as of a simple nondeterministic algorithm, which starts with the phrase $\alpha_0 = S$ and in i-th step, it nondeterministically chooses a production from R extending already constructed phrase α_{i-1} to a phrase α_i if such production exists and rejecting otherwise. When this algorithm reaches the phrase $w_1 \dots w_n$ then it accepts. It can be easily seen that this algorithm uses linear[1] time and logarithmic space. Unfortunately, its performance depends strongly on advantages of nondeterministic computation.

So, the idea to develop a deterministic polynomial time and sub-linear space algorithm is the following: since simulating nondeterminism is expensive in both time and space, we divide the word w into blocks and restrict our 'guessing' only to the order in which we extend whole blocks, using dynamic programming techniques to manage the ambiguity introduced by this abstracting. The precise description of this approach is developed in the two following subsections.

3.1 Outline of a derivation of w

Let $B_1 \dots B_m$ be a division of w into m blocks, each of length $\lceil \frac{n}{m} \rceil$ at most. Assume for the sake of simplicity that $n \ mod \ m = 0$ and let $o = \frac{n}{m}$. That is, $B_i = w_{(i-1)\cdot o+1} \dots w_{i \cdot o} = w_{i.1} \dots w_{i.o}$, where we employed a convention that $t.u = (t-1) \cdot o + u$, i.e., $t.u$ is the index of u-th symbol in t-th block.

By an **outline** $\mathcal{O} = \langle i_1, \dots, i_m \rangle$ we mean some ordering of blocks B_i.

Definition 3. *We say that a derivation $\rho = (S \Rightarrow^* \alpha)$ of a phrase $\alpha = w_1 \dots w_k X w_l \dots w_n$ is* **compatible with** $\mathcal{O}$*, where $\mathcal{O} = \langle i_1, \dots, i_m \rangle$ is an outline iff, for every two blocks B_{i_j}, $B_{i_{j'}}$ derived*[2] *in α, if $j < j'$ then B_{i_j} was derived before $B_{i_{j'}}$ in ρ.*

Intuitively, compatibility of a derivation ρ with an outline $\mathcal{O}$ means that the order of blocks' deriving in ρ obeys the order indicated by $\mathcal{O}$.

Going further, we say that a phrase α is **compatible with** $\mathcal{O}$ iff there exists a derivation $S \Rightarrow^* \alpha$ compatible with $\mathcal{O}$. Finally, it can be seen that any derivation $\rho = (S \Rightarrow^* w)$ defines an outline $\mathcal{O}_\rho$ corresponding to the order in which blocks are derived in ρ. By definition, ρ is compatible with $\mathcal{O}_\rho$. We say that an outline $\mathcal{O}$ is **valid** iff $\mathcal{O} = \mathcal{O}_\rho$ for some derivation ρ of w in G.

We define a k-**prefix** $P^k(\mathcal{O})$ of an outline $\mathcal{O} = \langle i_1, \dots, i_m \rangle$, where $0 \leq k \leq m$, as a partial ordering corresponding to k first elements from $\mathcal{O}$ i.e. $\langle i_1, \dots i_k \rangle$.

[1]We measure the complexity with respect to the length of the given word.

[2]By derivation of a block we mean deriving all its letters, i.e., the block B_i is derived in $\alpha = w_1 \dots w_k X w_l \dots w_n$ iff B_i is a subword of $w_1 \dots w_k$ or $w_l \dots w_n$.

We should note now the following straight-forward observation.

Observation 4. *If $\mathcal{O} = \langle i_1, \dots, i_m \rangle$ is a valid outline then for every $0 \le k \le m$ and some* **boundary indices** $0 \le b^k_l(\mathcal{O}) \le m+1$ *($l = 0, 1$) it holds that:*

a block B_i is among the first k blocks derived, i.e., $i \in \{i_1, \dots, i_k\}$ iff

$$1 \le i \le b^k_0(\mathcal{O}) \le m \vee 1 \le b^k_1(\mathcal{O}) \le i \le m$$

Moreover, for each $k \ge 1$, $i_k = b^k_{l^k(\mathcal{O})}(\mathcal{O})$ for some $l^k(\mathcal{O}) \in \{0, 1\}$, we call $l^k(\mathcal{O})$ k-**parity** *of $\mathcal{O}$.*

The proof of the observation is based on noticing that valid outline $\mathcal{O}$ has to correspond to some derivation ρ of w (i.e. $\mathcal{O} = \mathcal{O}_\rho$). So, since G is linear then in each stage of ρ (except the last one) the corresponding phrases α have one nonterminal with $B_1 \dots B_{b^k_0(\mathcal{O})}$ derived[3] on the left and $B_{b^k_1(\mathcal{O})} \dots B_m$ — on the right, where k denotes the number of blocks derived in α. In other words, every α corresponding to ρ is of the form $\alpha = B_1 \dots B_{b^k_0(\mathcal{O})} w_{b^k_0(\mathcal{O})+1.1} \dots w_l X w_s \dots w_{b^k_1(\mathcal{O})-1.o} B_{b^k_1(\mathcal{O})} \dots B_m$ and $l^k(\mathcal{O})$ indicates whether $b^k_0(\mathcal{O})$ or $b^k_1(\mathcal{O})$ was derived as the k-th one.

Rephrased in the above terms, our desired algorithm consists of generating all outlines which can be valid, and checking, for each such $\mathcal{O}$, whether it is valid indeed i.e. whether there exists a derivation of w compatible with $\mathcal{O}$. The first issue in the context of the space and time resources needed by such approach, is determining how much outlines we do have to check to ensure that we have not omitted any valid one. The question is answered in the following lemma

Lemma 5. *There are at most 2^m valid outlines and they can be easily encoded in space $O(m)$.*

Proof.

Let $\mathcal{O} = \mathcal{O}_\rho$ for some derivation ρ of w. In order to settle the lemma, we construct a string $v^\rho_0 \dots v^\rho_m$ of $m+1$ bits for each different outline $\mathcal{O}_\rho$.

Let $\mathcal{O}_\rho = \langle i_1, \dots, i_m \rangle$, we should observe that only one of border blocks i.e. B_1 or B_m can be derived in ρ as the first one. So, we set v^ρ_0 to encode this.

Let $\alpha_f = w_1 \dots w_l X w_{l+2} \dots w_n$ be the phrase corresponding to the stage of ρ immediately before using the ending production $X \to w_{l+1}$. Now, for a block B_i we set $v^\rho_i = 0$ iff the block B_j being derived immediately after B_i in ρ has its first letter $w_{j \cdot o}$ *not* on the right side of w (with respect to α_f). For the last block B_{i_m} derived in ρ, $v^\rho_{i_m}$ is arbitrary. We see that in fact $v_{i_k} = l^{k+1}(\mathcal{O}_\rho)$ for $k > 0$.

To decode the outline $\mathcal{O} = \langle i_1, \dots, i_m \rangle$ from a string $v_0 \dots v_m$ we employ the following procedure. We obtain from v_0 the value of i_1. Now, let us assume that we have found the value of i_j for all $j \le k$ i.e. we have found $P^k(\mathcal{O})$. We look at v_{i_k} and set $i_{k+1} = b^k_0(\mathcal{O}) + 1$ if $v_{i_k} = 0$; and $i_{k+1} = b^k_1(\mathcal{O}) - 1$ otherwise (cf. Observation 4). Thus the lemma follows. ■

[3]We employ a convention that $B_0 = B_{m+1} = \epsilon$.

3.2 Checking validity of an outline

Now, to follow the idea of our approach we have to develop the way of finding (if any) derivation of w compatible with some outline $\mathcal{O}$ i.e. checking whether $\mathcal{O} = \langle i_1, \ldots, i_m \rangle$ is valid. Our method of fulfilling this task consists of tracing, with the help of dynamic-programming paradigm, all phrases α compatible with $\mathcal{O}$ for increasing length of α. More precisely, we employ the following definition.

Definition 6. *Let $\mathcal{O} = \langle i_1, \ldots, i_m \rangle$ be an outline, $0 \le k < m$, let*

$$D^k_{s,j}(\mathcal{O}) = \{B_1 \ldots B_{b^k_0(\mathcal{O})} w_{b^k_0(\mathcal{O})+1.1} \cdots w_{b^k_0(\mathcal{O})+1.s} X w_{b^k_1(\mathcal{O}).(1-j)} \cdots w_{b^k_1(\mathcal{O}).0} B_{b^k_1(\mathcal{O})} \ldots B_m | X \in N\},$$

for some $0 \le s, j \le o$ (we remind that $t.u = (t-1) \cdot o + u$). Finally, let for $0 \le j \le o$

$$D^k_j(\mathcal{O}) = \begin{cases} \bigcup_{s=0}^{o-1} D^k_{s,j}(\mathcal{O}) & \text{if } l^{k+1}(\mathcal{O}) = 1 \\ \bigcup_{s=0}^{o-1} D^k_{j,s}(\mathcal{O}) & \text{otherwise} \end{cases}$$

*We say that a set C is a (k, j)-***configuration with respect to** *$\mathcal{O}$ iff $C \subseteq D^k_j(\mathcal{O})$ and for every $\alpha \in C$ there is a derivation of α in G.*

We say that such C is **left** *(resp.* **right***) iff $l^{k+1}(\mathcal{O}) = 1$ ($l^{k+1}(\mathcal{O}) = 0$).*

We call C **full** *iff it holds that for every $\alpha' \in D^k_j(\mathcal{O})$ extending some phrase $\alpha \in C$, $\alpha' \in C$.*

We call C **complete** *iff every derivation of w compatible with $\mathcal{O}$, extends some phrase in C.*

Clearly, any (k, j)-configuration C can be easily implemented as an array consisting of $O(|N| \cdot o + |k| + |j|) = O(\frac{n}{m} + \log n)$ bits corresponding to all elements of $D^k_j(\mathcal{O})$.

Intuitively, a left (resp. right) (k, j)-configuration C with respect to $\mathcal{O}$ corresponds to some set of phrases derivable in G that have already derived all blocks from $P^k(\mathcal{O})$ and contain exactly j first (resp. last) letters of $B_{i_{k+1}}$. Furthermore, fullness of C means that we cannot increase the number of elements of C using only left (resp. right) productions. Finally, completeness ensures us that we did not 'lose track' of a possible derivation of w compatible with $\mathcal{O}$. We should also be aware that there can be more than one full and complete (k, j)-configuration with respect to $\mathcal{O}$, but all of them will contain a unique minimal one (in the set-theoretic sense). It is also worth noting that (k, o)-configuration for $k < m-1$ is $(k+1, 0)$-configuration as long as $l^{k+1}(\mathcal{O}) = l^{k+2}(\mathcal{O})$.

Now, having some full and complete (k, j)-configuration C with respect to $\mathcal{O}$ for $0 \le k < m - 1$, $0 \le j < o$ by **extending** C we mean finding full and complete $(k + 1, j')$-configuration, for some $j' < o$ such that $j' = 0$ if $k = m - 2$.[4]

Lemma 7. *Let, for an outline $\mathcal{O}$, $0 \le k < m - 1$ and $0 \le j < o$, C be a full and complete (k, j)-configuration with respect to $\mathcal{O}$. We can extend C in deterministic time $O(\frac{n^2}{m^2})$ and space $O(\frac{n}{m})$.*

[4]This condition is needed in order to ensure that if, for example, C is left then $w_{b^k_0(\mathcal{O})+1.1} \cdots w_{b^k_0(\mathcal{O})+1.s}$ does not overlap with $w_{b^k_1(\mathcal{O}).(1-j')} \cdots w_{b^k_1(\mathcal{O}).0}$ for some $0 \le s \le o - 1$.

Proof.

Throughout the whole proof we assume that C is a left configuration i.e. $l^{k+1}(\mathcal{O}) = 1$, since the opposite case's treatment is completely analogous.

Obtaining full and complete (k, o)-configuration

Let us focus first onto finding a full and complete $(k, j+1)$-configuration C' with respect to $\mathcal{O}$. We start with $C' := \emptyset$. To make C' complete it is sufficient to add to C' only those phrases α' extending some phrase in C, which are from $D^k_{j+1}(\mathcal{O})$. To see this we should note that C is complete and no derivation of w compatible with $\mathcal{O}$ can derive $w_{b_0^k(\mathcal{O})+1.o}$, before $w_{b_1^k(\mathcal{O}).-j}$. The latter holds since deriving $w_{b_0^k(\mathcal{O})+1.o}$ would mean deriving $B_{b_0^k(\mathcal{O})+1}$ before $B_{b_1^k(\mathcal{O})-1} = B_{i_{k+1}} \neq B_{b_0^k(\mathcal{O})+1}$ (we remind that $j < o$ and $k < m-1$). So, restricting the form of α' only to the ones corresponding to $D^k_{j+1}(\mathcal{O})$, does not rule out from C' possible derivations of w compatible with $\mathcal{O}$.

We should note now that if $\alpha' \in D^k_{j+1}(\mathcal{O})$ has to extend some $\alpha \in C \subseteq D^k_j(\mathcal{O})$ then there is exactly one right production in the derivation $\alpha \Rightarrow^* \alpha'$. Thus, by fullness of C, to make C' complete it is sufficient to initially take C' to be the set $\mathcal{A}'$ of those $\alpha' \in D^k_{j+1}(\mathcal{O})$ which can be derived from some $\alpha \in C$ by exactly one right production. Then, by making C' full, we will ensure that all the desired $\alpha' \in D^k_{j+1}(\mathcal{O})$ extending $\alpha \in C$ are indeed in C'.

Obviously, finding $\mathcal{A}'$ can be done in time $O(|C|) = O(\frac{n}{m})$ and the space $O(\frac{n}{m})$ — needed to store only two configurations. It is left to make C' full. We can do this by simply starting with $i = 0$, looking at each of phrases $\alpha' \in C' \cap D^k_{i,j+1}(\mathcal{O})$, checking whether there exists a left production extending α' to some $\alpha'' \in D^k_{i+1,j+1}(\mathcal{O})$ and adding α'' to C' if so. After repeating this procedure with increasing i until $i = o-1$, we end up with full C'. Obviously, this operation can be done in $O(\frac{n}{m})$ time.

In the above manner we have obtained C' being a full and complete left $(k, j+1)$-configuration with respect to $\mathcal{O}$. If $j+1 < o$ then, after erasing C and renaming C' to C, we can repeat the above procedure — after at most $(o-1) = O(\frac{n}{m})$ iterations we would end up with C' being a full and complete (k, o)-configuration.

Obtaining full and complete $(k+1, \cdot)$-configuration

If $l^{k+2}(\mathcal{O}) = 1$ then C' would be also the desired full and complete $(k+1, 0)$-configuration. Thus it is left to consider the case $l^{k+2}(\mathcal{O}) = 0$ in which we will find a full and complete right $(k+1, o-1)$-configuration C'' with respect to $\mathcal{O}$.

We should note first that $b_0^m(\mathcal{O}) = b_1^m(\mathcal{O}) - 1$ and thus $l^m(\mathcal{O})$ can be arbitrary. So, if $l^{k+2}(\mathcal{O}) = 0$ then we may assume that $k+2 < m$, since otherwise we could choose $l^{k+2}(\mathcal{O}) = l^m(\mathcal{O}) = 1$. Therefore, we can be sure that no overlap of $w_{b_0^{k+1}(\mathcal{O})+1.1} \cdots w_{b_0^{k+1}(\mathcal{O})+1.(j'=o-1)}$ and $w_{b_1^{k+1}(\mathcal{O}).(1-s)} \cdots w_{b_1^{k+1}(\mathcal{O}).0}$ will occur for some $0 \leq s \leq o-1$.

We will retrieve C'' by finding $C'' \cap D^{k+1}_{o-1,s}(\mathcal{O})$ for all $0 \leq s \leq o-1$. Clearly, by completeness of C', it suffices only to ensure that C'' is full and all phrases $\alpha'' \in D^{k+1}_{o-1}(\mathcal{O})$, extending some $\alpha' \in C'$ by means of a derivation $\alpha' \Rightarrow^* \alpha''$ compatible with $\mathcal{O}$, are in C''.

Let $\mathcal{O}'$ be an outline $\mathcal{O}$ altered by setting $l^{k+2}(\mathcal{O}') = 1$ (instead of $l^{k+2}(\mathcal{O}) = 0$)[5]

[5]We recall that in proof of Lemma 7 it was shown that the outline can be described exclusively by

and let us analyze C' as a $(k+1,0)$-configuration with respect to $\mathcal{O}'$. Namely, let $C'_0 = C'$ be a full and complete $(k+1,0)$-configuration with respect to $\mathcal{O}'$; and C'_s, for $0 < s \leq o-1$, be a full and complete $(k+1,s)$-configuration with respect to $\mathcal{O}'$ obtained by application to C'_{s-1} exactly the same procedure that we used earlier to obtain full and complete left $(k,j+1)$-configuration from C.

The crucial observation here is that taking $C'' = \bigcup_{s=0}^{o-1} C'_s \cap D^{k+1}_{o-1,s}(\mathcal{O}')$ will make C'' full and complete. To see this we should first convince ourselves that $D^{k+1}_{o-1,s}(\mathcal{O}') = D^{k+1}_{o-1,s}(\mathcal{O})$ because $P^{k+1}(\mathcal{O}) = P^{k+1}(\mathcal{O}')$ and thus $b^{k+1}_l(\mathcal{O}) = b^{k+1}_l(\mathcal{O}')$. Then we should notice that any derivation ρ of w compatible with $\mathcal{O}$ has to derive $w_{b^{k+1}_0(\mathcal{O})+1.o-1}$ before $w_{b^{k+1}_1(\mathcal{O})-1.1}$, because otherwise $B_{b^{k+1}_1(\mathcal{O})-1}$ would be derived before $B_{b^{k+1}_0(\mathcal{O})+1} = B_{i_{k+2}(\mathcal{O})}$. Therefore, if α is the phrase corresponding to the stage of ρ in which $w_{b^{k+1}_0(\mathcal{O})+1.o-1}$ was just derived, then it must be the case that $\alpha \in \bigcup_{s=0}^{o-1} D^{k+1}_{o-1,s}(\mathcal{O}')$. Now, $C'_0 = C'$ is complete with respect to $\mathcal{O}$, so it contains a phrase α' corresponding to some stage of ρ such that $(\alpha' \Rightarrow^* \alpha) = \beta$. However, β does not derive neither $w_{b^{k+1}_0(\mathcal{O})+1.o}$ nor $w_{b^{k+1}_1(\mathcal{O})-1.1}$ so it cannot be ruled out during C'_s computing. Therefore, $\alpha \in \bigcup_{s=0}^{o-1} C'_s \cap D^{k+1}_{o-1,s}(\mathcal{O}') = C''$. Thus C'' is complete with respect to $\mathcal{O}$ and it is full because all C'_s were also such[6].

So, we can compute C'_s consecutively (i.e. keeping only the latest C'_s) for increasing $s \leq o-1$ and add for each s $C'_s \cap D^{k+1}_{o-1,s}(\mathcal{O}')$ to (initially empty) C''. We convince ourselves in this way that full and complete $(k+1,o-1)$-configuration C'' with respect to $\mathcal{O}$ can be obtained in time $O(\frac{n^2}{m^2})$ and space $O(\frac{n}{m})$, which concludes the proof.

■

We are ready now to state our main theorem

Theorem 8. *Let $G = \langle \mathcal{A}, N, R, S \rangle$ be a linear grammar, $w = w_1 \dots w_n$ be a word over $\mathcal{A}$ and $\delta > 0$. One can check in deterministic time $O(\frac{n^{2+\delta}}{\log n})$ and space $O(\frac{n}{\log n})$ whether w has a derivation in G.*

Proof. Let $m = \lfloor \delta \log n \rfloor$, without loss of generality we may assume that $n \, mod \, m = 0$. We partition w into m blocks of length $\frac{n}{m}$. The algorithm is following:

i_1 and the values of $l^k(.)$. Thus $\mathcal{O}'$ is well defined.

[6]But even if C'' would be only complete, but not full we could make it full within the desired time and space bounds by the technique presented earlier.

```
1. Iterate through all valid outlines $\mathcal{O}$
2.     Decode the ordering of blocks corresponding to the current outline $\mathcal{O}$
3.     Prepare $C$ being the initial complete and full $(0,0)$-configuration with
       respect to $\mathcal{O}$.
4.     for $k = 0$ to $m-2$
5.         extend $C$ (with respect to $\mathcal{O}$) and save the result as $C$
6.     perform an ending procedure
7.     if a derivation of $w$ has been found then accept
8. reject
```

The above sketch needs two explanations: how to generate the initial full and complete $(0,0)$-configuration C and how to perform the ending procedure.

To resolve the first one, we just set $C = \{S\}$, which obviously makes C complete and then we obtain fullness of C by the procedure analogous to one used in the proof of Lemma 7.

To explain the way of treating the second one, consider, say left, complete and full $(m-1,0)$-configuration C^f with respect to some outline $\mathcal{O}$. We are looking for a letter $w_{i_m.e}$ $(1 \le e \le o)$ being the middle of some hypothetic derivation of ρ of w compatible with $P^m(\mathcal{O}) = \mathcal{O}$ whose last production is $\gamma = (X \to w_{i_m.e})$. Clearly, if $e = o$ then γ has to extend some phrase from $C^f \cap D^{m-1}_{o-1,0}(\mathcal{O})$. And in general, γ has to extend a phrase from $C^f_e \cap D^{m-1}_{e-1,o-e}(\mathcal{O})$, where C^f_e corresponds to a complete and full left $(m-1, o-e)$-configuration with respect to $\mathcal{O}$, in which we discard phrases corresponding to an overlapping fragments of $w_1 \dots w_{b_0^{m-1}(\mathcal{O}).s}$ and $w_{b_1^{m-1}(\mathcal{O}).(1-o+e)}$. Clearly, we can find all C^f_e consecutively by a procedure analogous to the one from Lemma 7 within the respective time and space bounds. If we are unable to find the desired γ for any value of e then there is no derivation of w compatible with $\mathcal{O}$.

Now, the safeness of the whole algorithm is a consequence of the fact that we accept only when we find some particular derivation of w in G. And the soundness is a result of checking every possible outline. Thus if there is no derivation of w compatible with some outline then there must be no derivation at all.

To see the time and space requirements of this algorithm, we should first note that we need at most $O(\frac{n}{m}) = O(\frac{n}{\log n})$ space since we never use more than two configurations simultaneously. In case of time bounds, we have m iteration of extending procedure and each requires $O(\frac{n^2}{m^2})$ time. So, for each of $O(2^{\delta \log n})$ iterations corresponding to different outlines we need $O(\frac{n^2}{m})$ time. Thus the total time needed is $O(\frac{n^{2+\delta}}{\log n})$.

■

As a simple consequence we get

Corollary 9. *Let $G = \langle \mathcal{A}, N, R, S \rangle$ be a linear grammar, $w = w_1 \dots w_n$ be a word over $\mathcal{A}$ and $f(n) > 0$. One can check in deterministic time $O(2^{f(n)} \cdot \frac{n^2}{f(n)})$ and space*

$O(\frac{n}{f(n)} + f(n))$ *whether w has a derivation in G.*

4 Conclusions and open problems

As it can be seen, even in case of non-deterministic CFLs recognition in deterministic polynomial time, it is possible to break the space bound imposed by dynamic programming. Of course, since the only known lower-bound is trivial $O(\log n)$ requirement, the natural question arises whether in case of linear languages the upper-bound of $\frac{n}{\log n}$ is all we can hope ? If the answer is positive then it gives us the lower-bound on space needed to recognition of general CFL in polynomial-time. If not, then the another big problem is determining whether linear languages (or even all CFLs) are in SC.

It would be even interesting to determine if the algorithm stated above can be improved, that is whether our 'guessing' of the derivation can consist of significantly fewer 'nondeterministic' steps. Answers to this question seem to have impact also on the problem of space requirements in algorithms using dynamic programming paradigm. This is so, because, informally speaking, exactly the same approach can be used to reduce the space complexity (for the cost of increased, but only polynomialy, time) in case of algorithms utilizing dynamic programming in the manner analogous to the one considered in the paper. More precisely, in case of problems which are solved by dynamic programming where dependencies are of existential nature and only the last row of constructed array containing already solved cases is needed.

References

[1] B. von Braunmuehl, S. A. Cook, K. Mehlhorn, R. Verbeek, The Recognition of Deterministic CFL's in Small Time and Space, *Information and Control 56*, 1983, p. 34-51.

[2] John Cocke, Jacob T. Schwartz, *Programming languages and their compilers: Preliminary notes.* Technical report, Courant Institute of Mathematical Sciences, New York University.

[3] T. Kasami, *An efficient recognition and syntax-analysis algorithm for context-free languages.* Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

[4] Daniel H. Younger, *Recognition and parsing of context-free languages in time n^3.* Information and Control 10(2).

[5] S. A. Cook, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, *Proceedings of the eleventh annual ACM symposium on Theory of computing*, Atlanta, 1979, p. 338-345.

[6] P. M. Lewis II, R. E. Stearns, J. Hartmanis: Memory bounds for recognition of context-free and context-sensitive languages, *Proceedings of Symposium on the Foundations of Computer Science*, 1965, p. 191-202.

[7] L. Valiant, General context free recognition in less than cubic time, *Journal of Computer and System Science, Vol. 10*, 1975, p. 308-315.