Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 95 - 112, 2004.

CD Grammar Systems with $LL(k)$ Conditions*

Henning Bordihn

Institut für Informatik, Universität Potsdam
August-Bebel-Straße 89, D-14482 Potsdam, Germany
henning@cs.uni-potsdam.de

György Vaszil

Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
vaszil@sztaki.hu

Abstract

The paper extends the notion of context-free $LL(k)$ grammars to CD grammar systems working in the so-called $(=m)$ mode of derivation. The definition of the extended $LL(k)$ condition is followed by a short examination of the basic properties of $LL(k)$ CD grammar systems, and then an algorithm is presented which parses languages generated by CD grammar systems of this type in $O(n \cdot \log^2 n)$ time.

1 Introduction

The most investigated families of languages are the regular and the context-free ones. But, for many applications of formal languages, non-context-free aspects are needed. All programming languages, for example, in which variables must be declared before they can be used in the main program, involve the structure of the language $\{ ww \mid w \in \Sigma^* \}$, where Σ is some alphabet.

In [6], seven circumstances where context-free languages turn out to be insufficient are discussed, emerging in the field of programming languages, the language of logic, graph theory, development biology, economic modeling, folklore, and natural languages. The fact that there are non-context-free aspects in the syntax of spoken languages is exhaustively explored in the literature of linguistics, e.g., see [5, 9, 10]. This led to the concept of *mildly context-sensitive grammars* [9] providing some lower and upper bound conditions for families of languages which may be useful in linguistics. A grammar formalisms is said to be mildly context-sensitive if

*Research supported in part by the Intergovernmental S&T Cooperation Program of the Office of Research and Development Division of the Hungarian Ministry of Education and its German partner, the Federal Ministry of Education and Research (BMBF), under grant no. D-35/2000.

1. it contains (besides the context-free languages[1])

$$\begin{aligned} L_1 &= \{\, a^n b^n c^n \mid n \geq 1 \,\}, \\ L_2 &= \{\, wcw \mid w \in \{a,b\}^* \,\}, \\ L_3 &= \{\, a^m b^n c^m d^n \mid m,n \geq 1 \,\}, \end{aligned}$$

2. its generated languages can be parsed in polynomial time, and

3. it can generate only semilinear languages.

The next level in the Chomsky hierarchy, namely the context-sensitive grammars, are also not used in most applications since they are too powerful. For example, the fixed membership problem (i.e., the parsing problem) is **PSPACE**-complete and many other relevant decision problems are proved to be undecidable for context-sensitive grammars.

Therefore, a series of grammar formalisms has been introduced which are able to cover all the desired non-context-free aspects but maintain the nice properties of context-free grammars. Besides tree-adjoining grammars introduced in [9] and equivalent mildly context-sensitive grammars [16], the three most important sources of such language describing devices are grammars with parallel derivations (mainly Lindenmayer systems), grammars with regulated rewriting, and grammar systems; for a survey about these topics, see [14].

Unfortunately, most of those mechanisms lose too many positive properties of the context-free grammars, in particular the fixed membership problem becomes **NP**-complete in many cases. Even the known deterministic polynomial-time parsing algorithms for tree-adjoining and the other mildly context-sensitive grammars are of complexity $O(n^6)$, see, e.g., [10].

The aim of the present paper is to select one representative of those non-context-free devices and restrict it in a such a way that (1) at least the lower bound of mildly context-sensitive languages is met, that is, all the aforementioned languages L_1, L_2 and L_3 can be described, and (2) a parsing algorithm can be provided the computational complexity of which is as close to linear time as possible.

To this end, cooperating distributed grammar systems (CD grammar systems, for short) are considered here. The concept has its root in [11] aiming to generalize the notion of two-level substitution grammars to a multi-level concept, but the theory started with [3] introducing cooperating distributed grammar systems for describing, in terms of formal grammars and languages, communities of cooperating autonomous problem solving agents which use the blackboard model of problem solving. Later on, they were also considered as sequential counterparts of tabled Lindenmayer systems [1].

A cooperating distributed grammar system (CDGS, for short) consists of a finite set of (context-free) grammars, called components, performing derivation steps on a common sentential form in turns, according to some cooperation protocol. One natural cooperation protocol is the so-called $= m$-mode, for some $m \geq 1$, where a

[1] This requirement to cover all context-free languages has been weakened in more recent articles.

component, once started, has to perform exactly m derivation steps. The next component is nondeterministically chosen. In terms of distributed problem solving, the components correspond to the independent problem solving agents, the sentential form to the current state of the problem solving, and the generated language represents the set of problem solutions. For more on CD grammar systems see [4, 7]. It is known that this type of CD grammar systems generates all context-free languages and that it can describe matrix languages (without appearance checking).

One natural approach to fast parsers is to restrict the CD grammar systems to unambiguity. The present paper aims to do so by means of an $LL(k)$ condition for CD grammar systems. This yields a class of grammars which describe all context-free $LL(k)$-languages, the languages L_1, L_2, and L_3 above, and their languages can be parsed in time $O(n \cdot \log^2 n)$ time. Without further restriction, however, they do *not* define a new class of mildly context-sensitive languages since also non-semilinear languages can be generated.

It is worth mentioning here, that the logarithm in the time bound of the algorithm is squared only because we carefully count bit operations in which reading and writing a search index n takes $O(\log n)$ time. A uniform measurement of our algorithm would yield a time complexity of $O(n \cdot \log n)$.

2 Definitions

We assume the reader to be familiar with the basic notions of formal languages, as contained in [15]. In general, we have the following conventions: $\subseteq$ denotes inclusion, while $\subset$ denotes strict inclusion. The set of positive integers is denoted by $\mathbb{N}$ and the cardinality of a set M is denoted by $\#M$. By V^+ we denote the set of nonempty words over alphabet V; if the empty word ε is included, the we use the notation V^*. Set union and substraction is denoted by $\cup$ and $-$, respectively.

A context-free grammar is a four tuple $G = (N, T, P, S)$, where N and T are disjoint alphabets of nonterminals and terminals, respectively, $S \in N$ is the axiom, and P is a finite set of productions of the form $A \to u$, where $A \in N$ and $u \in (N \cup T)^*$.

Let V be some alphabet and $w \in V^*$. By $First_k(w)$ we denote the prefix of length k of w if $|w| \geq k$ or, otherwise, the string w itself.

Now we repeat the definition of a cooperating distributed grammar system, where we restrict ourselves (without further mentioning) to the case of context-free components.

A cooperating distributed grammar system (CDGS for short) of degree n is an $n+3$-tuple

$$G = (N, T, S, P_1, P_2, \ldots, P_n)\,,$$

where N and T are two disjoint alphabets, the alphabet of nonterminals and the alphabet of terminals, respectively, $S \in N$ is the axiom, and for $1 \leq i \leq n$, P_i is a finite set of context-free productions, i.e., productions of the form $A \to \alpha$, $A \in N$, $\alpha \in (N \cup T)^*$. The sets P_i are called components of the system G. For $1 \leq i \leq n$, we set $N_i = \{A \in N \mid A \to \alpha \in P_i\}$ and $T_i = (N \cup T) \setminus N_i$.

In the following, let x and y be sentential forms over $N \cup T$. A *direct derivation* according to some component P_i, $1 \leq i \leq n$, is defined as usual by $x \Longrightarrow_i y$ if and

only if $x = zAz'$, $y = z\alpha z'$, for some $z, z' \in (N \cup T)^*$ and $A \to \alpha \in P_i$. This relation is extended to an *m-step derivation* according to P_i, in symbols $x \overset{m}{\Longrightarrow}_i y$, in the natural way: $x \overset{m}{\Longrightarrow}_i y$ if and only if there are strings $x_0, x_1, \ldots, x_m$ over $N \cup T$ such that

$$x = x_0,\ y = x_m,\ \text{and } x_{j-1} \Longrightarrow_i x_j \text{ for } 1 \leq j \leq m\,.$$

Let $prod(x \overset{m}{\Longrightarrow}_i y)$ denote the set of production sequences which can be used in the m-step derivation $x \overset{m}{\Longrightarrow}_i y$. More precisely,

$$(A_0 \to \alpha_0, A_1 \to \alpha_1, \ldots, A_{m-1} \to \alpha_{m-1}) \in prod(x \overset{m}{\Longrightarrow}_i y)$$

if and only if there are strings $x_0, x_1, \ldots, x_m$ with $x_{j-1} = z_{j-1}A_{j-1}z'_{j-1}$ and $x_j = z_{j-1}\alpha_{j-1}z'_{j-1}$, and $A_{j-1} \to \alpha_{j-1} \in P_i$, for $1 \leq j \leq m$.

We write $x \overset{m}{\Longrightarrow}^* y$ if and only if $x = y$, or there are strings $x_0, x_1, \ldots, x_r$ over $N \cup T$, $r \geq 1$, such that

$$x = x_0,\ y = x_r,\ \text{and } x_{j-1} \overset{m}{\Longrightarrow}_{i_j} x_j\,,$$

for $1 \leq i_j \leq n$ for $1 \leq j \leq r$.

The language generated by G in the $(=m)$-mode of derivation is defined to be the set

$$L_{=m}(G) = \{\, w \in T^* \mid S \overset{m}{\Longrightarrow}^* w \,\}\,.$$

An m-step derivation $x \overset{m}{\Longrightarrow}_i y$ is referred to as *leftmost*, in symbols $x \overset{m}{\underset{l}{\Longrightarrow}}_i y$, if and only if the following condition is satisfied: If $x \overset{m}{\Longrightarrow}_i y$ is the derivation

$$x = x_0 \Longrightarrow_i x_1 \Longrightarrow_i x_2 \Longrightarrow_i \ldots \Longrightarrow_i x_m = y$$

for strings $x_0, x_1, x_2, \ldots, x_m \in (N \cup T)^*$, then we have

$$x_0 = u_0A_0z_0,\ x_1 = u_0\alpha_0z_0,$$
$$u_0 \in T^*,\ z_0 \in (N \cup T)^*,\ A_0 \to \alpha_0 \in P_i,$$

and, for $2 \leq j \leq m$,

$$x_{j-1} = u_{j-1}A_{j-1}z_{j-1},\ x_j = u_{j-1}\alpha_{j-1}z_{j-1},$$
$$u_{j-1} \in T_i^*,\ z_{j-1} \in (N \cup T)^*,\ A_{j-1} \to \alpha_{j-1} \in P_i.$$

That is, in the first step the leftmost occurrence of a symbol in N is replaced and in each of the following steps the leftmost occurrence of a symbol in N_i is replaced, each by some production in component P_i. This notion of leftmost derivation is the same as in [12] or as the notion of a *weakly leftmost* derivation in [8].

The relation $\overset{m}{\underset{l}{\Longrightarrow}}_i$ over $(N \cup T)^*$ is extended to $\overset{m}{\underset{l}{\Longrightarrow}}^*$ analogously to the extension of $\overset{m}{\Longrightarrow}_i$ to $\overset{m}{\Longrightarrow}^*$. The language generated by a CD grammar system G in the $(=m)$-mode of derivation working in this leftmost manner is defined by

$$L_{=m}(G\text{-left}) = \{\, w \in T^* \mid S \overset{m}{\underset{l}{\Longrightarrow}}^* w \,\}\,.$$

Next, we present the definition of an $LL(k)$ condition appropriate for CD grammar systems (in m-mode of derivation). It is adopted from the context-free case, for a definition see, e.g., [13, 15].

Definition 1 Let $G = (N, T, S, P_1, P_2, \dots P_n)$, $n \geq 1$, be a CDGS and let $k \geq 1$ and $m \geq 1$. We say G *satisfies the* $LL(k)$ *condition* in the $(=m)$-mode of derivation if the following holds: Let

$$S \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uv \text{ and}$$

$$S \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uXy' \overset{m}{\underset{l}{\Longrightarrow}}{}_{i'} uz' \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uv'$$

be two derivations in G, where $u, v, v' \in T^*$, $X \in N$, $y, y', z, z' \in (N \cup T)^*$, and $First_k(v) = First_k(v')$. Then $i = i'$ and $prod(uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz) = prod(uXy' \overset{m}{\underset{l}{\Longrightarrow}}{}_{i'} uz')$ is a singleton set.

The idea behind this concept is the following: Given a terminal word uv and a sentential form uXy, $X \in N$ and $y \in (N \cup T)^*$, which has been obtained from S by iterated leftmost m-step derivations, the first k letters of v allow to determine the next component and the sequence of rules of that component which is to be applied to uXy in order to derive uv by leftmost m-step derivations.

By $CD_n(=m)$ with $m, n \geq 1$ the family of languages which can be generated by a CD grammar system of degree n working in the m-mode of derivation is denoted. If we restrict the CD grammar systems of degree n to satisfy the $LL(k)$-condition in the $(=m)$-mode of derivation, then the families of languages obtained by leftmost derivations are denoted by $CD_nLL(k)(=m)$ with $k, m \geq 1$. If the number of components is not restricted, we write $CD_*(=m)$ and $CD_*LL(k)(=m)$, respectively. Finally, let $LL(k)$ denote the family of all context-free $LL(k)$ languages [15].

Example 1 Consider the CD grammar system

$$\Gamma = (\{S, S', S'', A, B, C, A', B', C'\}, \{a, b, c\}, S, P_1, P_2, P_3, P_4)$$

with

$$\begin{aligned} P_1 &= \{S \to S', S' \to S'', S'' \to ABC\}, \\ P_2 &= \{A \to aA', B \to bB', C \to cC'\}, \\ P_3 &= \{A' \to A, B' \to B, C' \to C\}, \\ P_4 &= \{A \to a, B \to b, C \to c\}. \end{aligned}$$

This system generates (by leftmost derivations) in the $(= 3)$ mode the language

$$L_1 = \{\, a^n b^n c^n \mid n \geq 1 \,\},$$

and as seen below, satisfies the $LL(2)$ condition, thus, $\{\, a^n b^n c^n \mid n \geq 1 \,\} \in CD_4LL(2)(= 3)$.

Considering a derivation

$$S \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uv,$$

the pair $X \in N$ and $First_2(v)$ determines the component P_i and the unique production sequence in $prod(uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz)$ which are used, as indicated in the following table.

	aa	ab
S	P_1: $(S \to S', S' \to S'', S'' \to ABC)$	P_1: $(S \to S', S' \to S'', S'' \to ABC)$
A	P_2: $(A \to aA', B \to bB', C \to cC')$	P_4: $(A \to a, B \to b, C \to c)$
A'	P_3: $(A' \to A, B' \to B, C' \to C)$	P_3: $(A' \to A, B' \to B, C' \to C)$

It is an easy exercise to prove that $\{ a^n b^n c^n \mid n \geq 1 \} \in CD_4LL(2)(= 2)$ also holds. (Let A produce an a and b simultaneously.)

Example 2 Analogously, one can see that the CD grammar systems

$$\Gamma = (\{S, S', A, B, A', B'\}, \{a, b, c\}, S, P_1, P_2, \ldots, P_7)$$

with

$$\begin{aligned} P_1 &= \{S \to S', S' \to c\}, \\ P_2 &= \{S \to S', S' \to AcB\}, \\ P_3 &= \{A \to aA', B \to aB'\}, \\ P_4 &= \{A \to bA', B \to bB'\}, \\ P_5 &= \{A' \to A, B' \to B\}, \\ P_6 &= \{A \to a, B \to a\}, \\ P_7 &= \{A \to b, B \to b\}. \end{aligned}$$

and

$$\Gamma = (\{S, S', A, B, A', B'\}, \{a, b, c, d\}, S, P_1, P_2, \ldots, P_5)$$

with

$$\begin{aligned} P_1 &= \{S \to S', S' \to AB\}, \\ P_2 &= \{A \to aA', B \to cB'\}, \\ P_3 &= \{A \to A'b, B \to B'd\}, \\ P_4 &= \{A' \to A, B' \to B\}, \\ P_5 &= \{A \to ab, B \to cd\}. \end{aligned}$$

obey the $LL(2)$ condition in the =2-mode, generating (by leftmost derivations) the languages

$$L_2 = \{ wcw \mid w \in \{a, b\}^* \}$$

and

$$L_3 = \{ a^m b^n c^m d^n \mid m, n \geq 1 \},$$

respectively.

Hence, L_1, L_2, and L_3 from the definition of mild context-sensitivity are contained in $CD_*LL(2)(=2)$.

3 Properties of $CD_*LL(k)(=m)$

First, we present the following trivial hierarchies.

Lemma 1 *For any integers $k \geq 1$, $m \geq 1$, and $n \geq 1$, we have*

1. $CD_nLL(k)(= m) \subseteq CD_nLL(k+1)(= m)$,
2. $CD_*LL(k)(= m) \subseteq CD_*LL(k+1)(= m)$,
3. $CD_nLL(k)(= m) \subseteq CD_{n+1}LL(k)(= m)$.

Concerning the parameter m, one only knows from the non-restricted case that there exists a prime lattice structure-like hierarchy [4].

Next, we show that all context-free $LL(k)$ languages are in $CD_nLL(k)(=m)$ for some $n, m \geq 1$.

Theorem 2 *For all $k \geq 1$ and $m \geq 1$, $LL(k) \subseteq CD_*LL(k)(=m)$.*

Proof. Let $L \in LL(k)$ for some $k \geq 1$, and let $G = (N, T, P, S)$ be a context-free $LL(k)$ grammar with $L = L(G)$. Let the rules $r \in P$ be labelled by $1 \leq lab(r) \leq |P|$. For any integer $m \geq 1$, we construct a CD grammar system $\Gamma = (N', T, S, P_1, P_2, \ldots, P_n)$ satisfying the $LL(k)$ condition and generating L in the $=m$ mode of derivation as follows. The number of components of Γ is going to be $n = |P|$. Let

$$N' \;=\; N \cup \{\, X^i \mid 1 \leq i \leq m-1,\ X \in N \,\},$$

and for $1 \leq i \leq |P|$, let

$$P_i \;=\; \{\, X \to X^1, X^1 \to X^2, \ldots, X^{m-1} \to \alpha \mid i = lab(X \to \alpha) \,\}.$$

It is easy to see that a rewriting step $x \overset{m}{\underset{l}{\Longrightarrow}}_i y$ in Γ is possible if and only if $x \Rightarrow_i^l y$ is possible in G, where $\Rightarrow_i^l$ denotes a rewriting step on the leftmost nonterminal using rule r with $lab(r) = i$. □

Lemma 3 *There are non-semilinear languages in $CD_*LL(1)(= 2)$.*

Proof. Consider the CD grammar system

$$G = (N,\ \{a, b, c, d, e, f\},\ P_1, P_2, \ldots, P_{13},\ S)$$

with $N = \{S, S', A, A', B, B', C, C', D, E, F, T, X\}$ and

$$\begin{array}{rclcrcl}
P_1 &=& \{S \to S',\ S' \to AET\}, & & & & \\
P_2 &=& \{A \to dA',\ E \to DD\}, & & P_7 &=& \{B \to dB',\ D \to EE\}, \\
P_3 &=& \{A' \to X,\ X \to A\}, & & P_8 &=& \{B' \to X,\ X \to B\}, \\
P_4 &=& \{A \to aX,\ X \to B\}, & & P_9 &=& \{B \to aX,\ X \to A\}, \\
P_5 &=& \{A \to X,\ X \to C\}, & & P_{10} &=& \{B \to X,\ X \to C\}, \\
P_6 &=& \{C \to eC',\ D \to b\}, & & P_{11} &=& \{C \to fC',\ E \to b\}, \\
P_{12} &=& \{C' \to X,\ X \to C\}, & & & & \\
P_{13} &=& \{C' \to c,\ D \to F,\ E \to F,\ T \to c\}\,. & & & &
\end{array}$$

We are going to determine the language $L_{=2}(G\text{-left})$. Any leftmost derivation of G in the $= 2$-mode starting off with the axiom must initially use component P_1 leading to AET. Then, the components P_2, P_3, P_4, P_7, P_8, and P_9 can be used in turns. In this phase, the number of occurrences of D's and E's can be increased. Note that this number can at most be doubled until a new symbol a will be introduced before a further increase is possible. Moreover, whenever one more D or E is introduced, then simultaneously a terminal d must emerge. This phase is finished by one application of either P_5 or P_{10} turning the leftmost nonterminal (A or B) to C. Now, all occurrences of nonterminals D and E can be terminated with the help of P_6, P_{11}, and P_{12}. Finally, the leftmost and the rightmost nonterminals, that is C' and T at this stage of the derivation, can be terminated by using P_{13}. Since also this terminating component must be applied in the leftmost way and F is a trap symbol, it is guaranteed that all occurrences of D and E have vanished before P_{13} can successfully be applied in the $= 2$-mode. Therefore, in every non-terminal sentential form, either A, B, C (or its primed versions or X) is the leftmost occurring nonterminal, steering the selection of the components. Thus, the different phases of the derivation cannot be mixed.

Consequently, the non-semilinear language L is generated, where

$$L \subseteq K = \{fcbc\} \cup \{\, d^{i_1}ad^{i_2}a \dots d^{i_n}vcb^m c \mid n \geq 1,\, 0 \leq i_j \leq 2^j, \\ \text{for } 1 \leq j \leq n,\, m = 1 + \textstyle\sum_{j=1}^{n} i_j,\, v \in \{e, f\}^m \,\}\,.$$

Here, L is not equal to K only because the portion v has to obey some additional combinatorial properties which do not affect the non-semilinearity of the language. Since writing down these properties would decrease readability, they are omitted. On the other hand, the e's and f's are needed in order to make sure that G is $LL(k)$ in the $= 2$-mode.

In fact, one can readily prove that G satisfies the $LL(1)$ condition. □

According to Lemma 1, we know that there exist non semilinear languages in $CD_nLL(k)(= 2)$ for all $k \geq 1$ and all $n \geq 13$. By a simple prolongation technique, this is also true for any $= m$-mode of derivation, $m \geq 2$. One has to split the productions replacing the leftmost nonterminal to $m-1$ productions via new nonterminal symbols in each component. The restriction to leftmost derivations guarantees that these productions are consequently used such that the other production can be applied only once in any $= m$-step.

Corollary 4 *For any integers $k \geq 1$, $m \geq 2$, and $n \geq 13$, there are non-semilinear languages in $CD_nLL(k)(= m)$.*

4 Syntactic analysis of $CD_*LL(k)(=m)$

In this section we present a parser that is able to parse languages generated by CD grammar systems satisfying the $LL(k)$ condition in an efficient way. As we shall later see, its running time is $O(n \cdot \log^2 n)$ where n is the length of the input word.

Let G be a CD grammar system given as $G = (N, T, S, P_1, \dots, P_n)$ satisfying the $LL(k)$ condition in the $(=m)$-mode of derivation for some $m, k \geq 1$. First we present the notions we will use.

- A *production* is $p = X \to \alpha \in N \times (N \cup T)^*$ with $left(p) = X, right(p) = \alpha$.
- A *stack over* $\mathbb{N}$ is $st = x_j]x_{j-1}] \ldots]x_1]$, $x_i \in \mathbb{N}$, $1 \leq i \leq j$, with $top(st) = x_j$, $pop(st) = x_{j-1}] \ldots]x_1]$, and for some $y \in \mathbb{N}$, $push(y, st) = y]x_j]x_{j-1}] \ldots]x_1]$. The empty stack, $pop(x])$ for some $x \in \mathbb{N}$, is denoted by $\varepsilon]$.
- A *stack over* $N \cup T$ is $st = x_j]x_{j-1}] \ldots]x_1]$, $x_i \in N \cup T$, $1 \leq i \leq j$, with $top(st) = x_j$, $pop(st) = x_{j-1}] \ldots]x_1]$ and for some $y = y_1 \ldots y_m \in (N \cup T)^*$, $y_i \in N \cup T$, $1 \leq i \leq m$, $push(y, st) = y_1] \ldots]y_m]x_j]x_{j-1}] \ldots]x_1]$. The empty stack, $pop(x])$ for some $x \in N \cup T$, is denoted by $\varepsilon]$.
- A *production queue* is $pq = (p_1, p_2, \ldots, p_j)$, $p_i \in P_l$, $1 \leq l \leq n$, $1 \leq i \leq j$, with $first(pq) = p_1$, $butfirst(pq) = (p_2, \ldots, p_j)$.

The lookup table for the $LL(k)$ CD grammar system is given as $lookupTable \subseteq N \times T^k \times PQ$ where PQ denotes the set of all production queues consisting of m productions; it is a function which for a nonterminal $X \in N$ and a terminal word of length k, $y \in T^k$, returns a production queue $pq = lookupTable(X, y)$.

The parsing algorithm is given in Figure 1. It uses the variables

- $step$, $stepOfTopmost \in \mathbb{N}$, natural numbers,
- $mainStack$, a stack over $N \cup T$, the "main" stack of the parser,
- $stacksForN$, an l-tuple of stacks for natural numbers where $l = |N|$; it provides a stack over $\mathbb{N}$ for each nonterminal of the grammar system,
- $input \in T^*$, the string to be analyzed,
- $topmost \in N$, a nonterminal symbol,
- $pQueue$, a production queue as above,
- $pQueuesLeft \subseteq \mathbb{N} \times PQ$, where PQ denotes the set of all production queues of length at most m, that is, $pQueuesLeft$ is a set of pairs of the form $(i; pq)$ where i is an integer and pq is a production queue as above,
- $pToUse \in N \times (N \cup T)^*$, a production as above.

Example 3 In the following we demonstrate the work of the algorithm through an example.

Consider the CD grammar system

$$\Gamma = (\{S, A_1, A_2, A_3, A_4, A_5\}, \{a, b\}, S, P_1, P_2, P_3)$$

with

$$\begin{aligned} P_1 &= \{S \to A_1A_2A_1A_3, S \to bA_1A_2A_1A_3, A_2 \to b, A_3 \to A_4\}, \\ P_2 &= \{A_1 \to aA_2, A_4 \to A_5\}, \\ P_3 &= \{A_2 \to a, A_5 \to b\}, \end{aligned}$$

```
1 step ← 0
2 mainStack ← push(mainSt, S)
3 stacksForN(S) ← push(stacksForN(S), 0)
4 while mainStack is not empty and there is no ERROR do
5   if top(mainStack) is a terminal symbol then
6       if top(mainStack) coincides with the first symbol of input then
7           mainStack ← pop(mainStack)
8           input ← input without its first symbol
9       else ERROR
10  else topmost ← top(mainStack)
11      stepOfTopmost← top(stacksForN(topmost))
12      stacksForN(topmost) ← pop(stacksForN(topmost))
13      if there exist (i; pQueue) ∈ pQueuesLeft such that
        i ≥ stepOfTopmost, left(first(pQueue)) = topmost,
        and furthermore, if (i'; pQueue') ∈ pQueuesLeft
        with left(first(pQueue')) = topmost, then i < i', then
14          pQueuesLeft ← pQueuesLeft − {(i; pQueue)}
15          pToUse ← first(pQueue)
16          pQueue ← butfirst(pQueue)
17          if pQueue is not empty then
18              pQueuesLeft ← pQueuesLeft ∪ {(i; pQueue)}
19          mainStack ← pop(mainStack)
20          mainStack ← push(mainStack, right(pToUse))
21          for each symbol X from right(pToUse) do
22              if X ∈ N then
23                  stacksForN(X) ← push(stacksForN(X), step)
24      else step ← step + 1
25          lookahead ← the next k symbols of input
26          pQueue ← lookupTable(topmost, lookahead)
27          if pQueue is empty then
28              ERROR
29          else pToUSe ← first(pQueue)
30              pQueue ← butfirst(pQueue)
31              if pQueue is not empty then
32                  pQueuesLeft ← pQueuesLeft ∪ {(step, pQueue)}
33              mainStack ← pop(mainStack)
34              mainStack ← push(mainStack, right(pToUse)
35              for each symbol X from right(pToUse) do
36                  if X ∈ N then
37                      stacksForN(X) ← push(stacksForN(X), step)
38 if there is no ERROR then successful termination
```

Figure 1: The parsing algorithm.

	a	b
S	P_1: $(S \to A_1A_2A_1A_3, A_2 \to b, A_3 \to A_4)$	P_1: $(S \to bA_1A_2A_1A_3, A_2 \to b, A_3 \to A_4)$
A_1	P_2: $(A_1 \to aA_2, A_1 \to aA_2, A_4 \to A_5)$	—
A_2	P_3: $(A_2 \to a, A_2 \to a, A_5 \to b)$	—

Figure 2: The lookup table for the grammar system of Example 3.

This system generates, in the (= 3) mode, the finite language

$$L = \{aabaab, baabaab\},$$

and satisfies the $LL(1)$ condition. The lookup table for the parser is seen on Figure 2.

Let us see how the parser analyzes the string $aabaab \in L_{=3}(G\text{-left})$. This string is generated in three steps in the (= 3)-mode as follows.

$$S \overset{3}{\underset{l}{\Longrightarrow}}_1 A_1bA_1A_4 \overset{3}{\underset{l}{\Longrightarrow}}_2 aA_2baA_2A_5 \overset{3}{\underset{l}{\Longrightarrow}}_3 aabaab.$$

Now we will follow the work of the parser step-by-step, and describe its configuration by

$$(input, mainStack, step, stacksForN, pQueuesLeft)$$

where the variables are as described above. The value of $stacksForN$ will be denoted as $(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$ where α_0 is the contents of $stacksForN(S)$, and for $i \in \{1, 2, 3, 4, 5\}$, x_i is the contents of $stacksForN(A_i)$.

The initial configuration of the parser is

$$(aabaab, S], 0, (0], \varepsilon], \ldots, \varepsilon]), \emptyset),$$

meaning that nothing is read from the input, the initial symbol, S, is placed in the main stack, the step counter is set to *zero*, the integer *zero* is placed in the stack $stacksForN(S)$ associated to the nonterminal S which indicates that it appeared in the main stack when the counter *step* had value *zero*, and the set of production queues waiting to be applied, $pQueuesLeft$, is empty.

The main stack is not empty, so the parser starts the execution of the **while** loop of the algorithm at line 4. The symbol on the top of the main stack is a nonterminal, so it jumps to line 10. Since $pQueuesLeft$, the set of production queues waiting for execution is empty, after popping the stack $stacksForN(S)$ associated to the symbol in the main stack, the parser proceeds with the instruction on line 24 by increasing the counter *step* and identifying the production queue to be applied with the help of the lookup table. At this point

$lookahead = a$,

$topmost = S$,
$pQueue = (S \to A_1A_2A_1A_3, A_2 \to b, A_3 \to A_4)$.

The production to be used is the first production of $pQueue$,

$pToUse = S \to A_1A_2A_1A_3$.

Now the remaining part of $pQueue$ is stored in $pQueuesLeft$ indexed with *one*, the current value of the *step* counter, as a pair $(1; A_2 \to b, A_3 \to A_4)$. This indicates that the rules of this queue can be used on nonterminals that appeared in the main stack when the *step* counter had value *one* or less. Now the top of the main stack is replaced with the word on the right side of $pToUse$, the stack associated to S is emptied, and the value of *step* is placed into the stacks associated to the nonterminals appearing on the right side of the rule, $stacksForN(X)$, $X \in \{A_1, A_2, A_3\}$. The configuration of the parser is

$$(aabaab, A_1]A_2]A_1]A_3], 1, (\varepsilon], 1]1], 1], 1], \varepsilon], \varepsilon]), \{(1; A_2 \to b, A_3 \to A_4)\}).$$

Now the parser starts the execution of the **while** loop on line 4 again. Since the top of the main stack is a nonterminal, A_1, and since there is no production queue in $pQueuesLeft$ having A_1 on the left-hand side of its first rule, after popping $stacksForN(A_1)$, the stack associated with the topmost nonterminal, the parser continues with line 24 of the algorithm by increasing the counter *step*, and determining the production queue and the production to be used with the help of the lookup table, obtaining

$pQueue = (A_1 \to aA_2, A_1 \to aA_2, A_4 \to A_5)$,
$pToUse = A_1 \to aA_2$.

After the application of the production $A_1 \to aA_2$ to the topmost nonterminal of the main stack, the parser is in the configuration

$$(aabaab, a]A_2]A_2]A_1]A_3], 2, (\varepsilon], 1], 2]1], 1], \varepsilon], \varepsilon]), \{(1; A_2 \to b, A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5)\}),$$

and then the execution of the algorithm continues at line 4 again.

Since the top of the main stack is the same terminal as the first symbol of the input, the parser enters

$$(abaab, A_2]A_2]A_1]A_3], 2, (\varepsilon], 1], 2]1], 1], \varepsilon], \varepsilon]), \{(1; A_2 \to b, A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5)\})$$

by popping the main stack and reading one letter of the input, then continues with line 4, and jumps to line 10 again.

Now the topmost nonterminal is A_2, and by popping *two* from the stack $stacksForN(A_2)$, it is clear that the production queue $(A_2 \to b, A_3 \to A_4)$ from $pQueuesLeft$ can not be used since it has index *one*. This means that the parser needs to turn to the lookup table again, obtaining

$pQueue = (A_2 \to a, A_2 \to a, A_5 \to b)$,

$pToUSe = A_2 \to a.$

After the necessary replacements in the stacks and after updating the value of other variables, the parser enters

$$(abaab, a]A_2]A_1]A_3], 3, (\varepsilon], 1], 1], 1], \varepsilon], \varepsilon]),$$
$$\{(1; A_2 \to b, A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}).$$

Now after popping the main stack and reading one more symbol of the input, the parser continues at line 10 again. This time, the topmost nonterminal is A_2, and the integer popped from the corresponding stack, $stacksForN(A_2)$ is *one*, so the first production of the production queue $(A_2 \to b, A_3 \to A_4)$ stored in $pQueuesLeft$ with the same index can be used. Thus, the parser continues at line 14 of the algorithm setting

$pToUse = A_2 \to b,$
$pQueuesLeft =$
$$\{(1; A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5), (3; A_2 \to a, A_5 \to\ b)\}.$$

After replacing the topmost nonterminal of the main stack with the right side of $pToUse$, the parser enters

$$(baab, b]A_1]A_3], 3, (\varepsilon], 1], \varepsilon], 1], \varepsilon], \varepsilon]),$$
$$\{(1; A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}).$$

After popping the main stack and reading one more symbol of the input, the condition on line 13 is satisfied again, so the parser sets

$pToUse = A_1 \to aA_2,$
$pQueuesLeft = \{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_2 \to a, A_5 \to\ b)\},$

and then uses them, entering

$$(aab, a]A_2]A_3], 3, (\varepsilon], \varepsilon], 3], 1], \varepsilon], \varepsilon]),$$
$$\{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}).$$

Popping and reading again, then

$pToUse = A_2 \to a,$
$pQueuesLeft = \{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_5 \to\ b)\},$

since the queue $(A_2 \to a, A_5 \to b)$ stored in $pQueuesLeft$ has index *three*, the same as the value obtained from the stack $stacksForN(A_2)$, so it can be used, producing

$$(ab, a]A_3], 3, (\varepsilon], \varepsilon], \varepsilon], 1], \varepsilon], \varepsilon]), \{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_5 \to b)\}).$$

After the main stack is popped again and one more input symbol is read, the parser sets

$pToUse = A_3 \to A_4,$
$pQueuesLeft = \{(2; A_4 \to A_5), (3; A_5 \to\ b)\},$

and enters

$$(b, A_4], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], 1], \varepsilon]), \{(2; A_4 \to A_5), (3; A_5 \to b)\}).$$

The value *one* is placed in the stack $stacksForN(A_4)$ because the queue index of the rule $A_3 \to A_4$ was *one* which means that the application of the rule happens in step *one*, that is, A_4 appears in the first (= 3)-mode step of the generation of the input string.

The next configuration is

$$(b, A_5], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon], 2]), \{(3; A_5 \to b)\}),$$

and then

$$(b, b], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon]), \emptyset),$$

after which the last input symbol is read and the main stack is once again popped, so the parser enters

$$(\varepsilon, \varepsilon], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon]), \emptyset),$$

and since the main stack is empty, finishes its work at line 38 of the algorithm.

Theorem 5 *If a parser is constructed as above, based on a given CD grammar system satisfying the $LL(k)$ condition working in the $(=m)$ derivation mode, then it halts on every input word w over the terminal alphabet of the grammar system after a running time of $O(n \cdot \log^2 n)$ where n is the length of w.*

Proof. Let $G = (N, T, S, P_1, \ldots, P_s)$ be a CD grammar system satisfying the $LL(k)$ condition in the $(=m)$-mode of derivation. First we show that the parser constructed according to G halts on every input.

Assume that the parser does not halt on an input word $w \in T^*$. This means that it loops infinitely, and it can only do that if the instructions on the lines 10 – 37 are executed infinitely many times. To see this, notice that the body of the main **while** loop contains one **if-then-else** statement. Instructions of the **then** part read an input symbol, so they can not be repeated infinitely many times. This implies that the **else** part on lines 10 – 37 is repeated infinitely many times.

This part of the algorithm contains an **if-then-else** statement starting with line 13, the execution of the instructions of this part mean either the execution of the **then** part on lines 14 – 23, or the **else** part on lines 24 – 37. If lines 10 – 37 are executed infinitely many times, then there must be infinitely many such executions when no terminal symbol is written on the top of the main stack in line 20 or in line 34, which means that there is an infinite sequence of consecutive executions of lines 10 – 37 during which no terminal symbol is ever written on the top of the main stack.

Since each execution of the instructions of lines 14 – 23 removes one production from the production queues stored in $pQueuesLeft$, the instructions on lines 24 – 37 must be executed infinitely many times, or the parser cannot loop infinitely.

Because the lookahead never changes and because the number of nonterminal symbols is finite, there must be a sequence of instructions starting with lines 24 –

37, continuing with possibly several executions of lines 10 – 23 or lines 24 – 37, and then ending with lines 24 – 37 again, in such a way that the value of *topmost*, that is, the topmost nonterminal of the main stack, is the same at the first and at the last execution of lines 24 – 37.

Since the choice of the productions to be applied is based on the lookup table (line 26), the situation outlined above can only happen if the lookup table have certain properties which we describe below.

Let $X \in N$ and $y \in T^k$ be a row and a column index of the lookup table, and let $maxchain(X, y)$ denote the production queue with the following properties:

- $maxchain(X, y)$ is a prefix $(p_1, \ldots, p_l)$ of the corresponding entry of the lookup table, $lookupTable(X, y) = (p_1, \ldots, p_l, p_{l+1}, \ldots, p_m)$.
- If $maxchain(X, y)$ and $lookupTable(X, y)$ are as above, then $X \Rightarrow_{p_1} X_1 w_1$ and $X_i w_i \Rightarrow_{p_{i+1}} X_{i+1} w_{i+1}$, $X_i \in N$, $w_i \in (N \cup T)^*$, for each $1 \leq i \leq l-1$, and each p_i rewrites the leftmost nonterminal, that is, it is of the form $p_1 = X \to \alpha_1$, and $p_i = X_{i-1} \to \alpha_i$, $2 \leq i \leq l$, and furthermore,
- $maxchain(X, y)$ contains the maximal number of productions with the properties above, that is, $p_{l+1} = Z \to w$ where $Z \neq X_l$.

The parser may enter an infinite loop, if there exist a column of the lookup table, labelled with $y \in V^k$, such that

$$X \Rightarrow_{maxchain(X,y)} X_1 w_1 \Rightarrow_{maxchain(X_1,y)} X_2 w_2 \Rightarrow_{maxchain(X_2,y)} \cdots \\ \cdots \Rightarrow_{maxchain(X_l,y)} X_{l+1} w_{l+1} = X w_{l+1}$$

where $X, X_i \in N$, $w_i \in (N \cup T)^*$, $1 \leq i \leq l+1$, and $\Rightarrow_{maxchain(X,y)}$ denotes a leftmost derivation sequence using the rules of the production queue $maxchain(X, y)$.

Now we show that such a column cannot exist in the lookup table. If during a leftmost $(=m)$-mode derivation we encounter the nonterminal X as the leftmost nonterminal, and the production queue identified by X and the lookahead would be the queue in $lookupTable(X, y)$, then a successful application of the rules would lead to the choice of the queue $lookupTable(X_1, y)$, $lookupTable(X_2, y)$, and so on, until we would obtain X again as the leftmost nonterminal with the same lookahead, thus, the production queues identified by the leftmost nonterminal and the lookahead would never lead to a successful derivation which is a contradiction.

Now we show that given the input word $w \in T^*$, the parser halts after $O(n \cdot \log^2 n)$ number of steps where $n = |w|$. With similar arguments as above, we can show that the number of instructions executed without reading any input symbol is $O(1)$ which means that the running time of the parser is the length of the input multiplied by the time necessary to execute an instruction. All the instructions used in the algorithm can be executed in constant time, except the evaluation of the condition on line 13 and the assignments on line 14, 18, and 32 because they require the manipulation of the data stored in the set structure $pQueuesLeft$. The evaluation of line 13 requires a search, the assignments require the addition and the deletion of an element using a set where the number of stored elements can be as many as $O(n)$.

All of these operations, however, can be executed in $O(\log^2 n)$ time if we use balanced search trees, such as red-black trees for example, to store the elements of the set $pQueuesLeft$. (For more on balanced search trees and red-black trees in particular, see [2].) The implementation of the set $pQueuesLeft$ must consist of a red-black tree for each nonterminal $X \in N$ which stores the indexed production queues $(i; pq) \in pQueuesLeft$ with $X = left(first(pq))$ ordered by the index $i \in \mathbb{N}$. Having such a structure, the evaluation of the condition on line 13 can be realized by turning to the search tree associated to the nonterminal *topmost* to obtain the pair $(i; pQueue)$ where either $i = stepOfTopmost$, or if such index is not present, then i is the smallest available index with $i \geq stepOfTopmost$. To perform this search takes $O(\log n)$ comparisons since even in the worst case when the index is not present, it is enough to explore one path of the red-black tree leading from the root to one of the leaves, and the length of these paths, that is, the height of the tree is $O(\log n)$. To execute lines 14, 18, and 32, that is, to add or remove elements from the structure first requires a search to determine the appropriate tree and the location of the element in the tree, and then a constant number of elements need to be manipulated to insert or to remove the data. Since the number of trees used are finite, the number of necessary comparisons and data manipulations are $O(\log n)$. One comparison or one data manipulating step, however, also requires $O(\log n)$ time, since the integers used to index the production queues, that is, the keys used to index the nodes of the search tree, might be as large as n, so their representation can be as long as $\log n$ which means that comparing, reading or writing them requires $O(\log n)$ elementary computation steps. This gives a total running time of $O(n \cdot \log^2 n)$ where $n = |w|$, the length of the input word. □

5 Conclusion

Cooperating distributed grammar systems working in $= m$-mode of derivation have been restricted in a way such that, on the one hand, they maintain enough power in order to generate all context-free $LL(k)$ languages, the languages L_1, L_2 and L_3 of the concept of mildly context-sensitive grammars and even some non-semilinear language, but, on the other hand, there is an efficient parsing algorithm of $O(n \cdot \log^2 n)$ time complexity. The focus in this paper was on the development of the concept of an $LL(k)$ condition which is appropriate for those systems, and of the parsing algorithm. The corresponding families of languages $(CD_nLL(k)(= m))$ need further investigations. The future research should investigate, among others, the following problems:

- Which of the inclusions given in Lemma 1 are strict?
- Is it decidable whether a given CD grammar system is $LL(k)$, for a given k or for any k?

Moreover, one could extend the research to other derivation modes. Finally, other restrictions like an appropriate $LR(k)$ condition can be taken into consideration.

References

[1] Bordihn, H., Csuhaj-Varjú, E., Dassow, J.: CD grammar systems versus L systems, in: Gh. Păun, A. Salomaa, (eds.), *Grammatical Models of Multi-Agent Systems*, Gordon and Breach, 1999, 18–32.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill Book Company, 1990.

[3] Csuhaj-Varjú, E., Dassow, J.: On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics* EIK, **26** (1990), 49–63.

[4] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[5] Ch. Culy, The complexity of the vocabulary of Bambara. *Ling. and Philosophy* **8** (1985), 345–351.

[6] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory.* Springer, 1989.

[7] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems. Chapter 4. *Handbook of Formal Languages*, in: G. Rozenberg and A. Salomaa (eds.), Springer, Berlin, 1997, 155-213.

[8] J. Dassow, V. Mitrana, On the leftmost derivation in cooperating grammar systems. *Rev. Roumaine Math. Pures Appl.*, **43** (1998), 361-374.

[9] A. K. Joshi, How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars, in: D. Dowty, L. Karttunen, and A. Zwicky (eds.), *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, Cambridge University Press, New York, 1985.

[10] A. K. Joshi, Y. Schabes, Tree-adjoining grammars, in: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, Springer, Berlin, 1997, 69–123.

[11] R. Meersman and G. Rozenberg, Cooperating grammar systems, in *Proceedings of Mathematical Foundations of Computer Science MFCS'78*, volume 64 of *LNCS*, Springer, Berlin, 1978, 364–374.

[12] V. Mitrana, Parsability approaches in CD grammar systems, in: R. Freund and A. Kelemenová (eds.), *Proceedings of the International Workshop Grammar Systems 2000*, Silesian University at Opava, 2000, 165–185.

[13] D. J. Rosenkrantz, R. E. Stearns, Properties of deterministic top-down grammars, *Information and Control*, **17** (1970), 226–256.

[14] G. Rozenberg, A. Salomaa, *Handbook of Formal Languages*, Springer, Berlin, 1997.

[15] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

[16] K. Vijay-Shanker, D. J. Weir, The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* **87** (1994), 511–546.