Theoretical Computer Science 161 (1996) 205–233

Theoretical Computer Science

On parsing coupled-context-free languages

Günter Hotz, Gisela Pitsch *,1

FB 14 – Informatik, Universität des Saarlandes, D-66123 Saarbrücken, Germany

Received March 1994; revised January 1995
Communicated by W. Thomas

Abstract

Coupled-Context-Free Grammars are a natural generalization of context-free grammars obtained by combining nonterminals to parentheses which can only be substituted simultaneously. Refering to the generative capacity of the grammars we obtain an infinite hierarchy of languages that comprises the context-free languages as the first and all the languages generated by Tree Adjoining Grammars (TAGs) as the second element. The latter is important because today, TAGs are commonly used to model the syntax of natural languages. Here, we present an approach to analyse all language classes of the hierarchy uniformly. The distinct generative capacity of the subclasses is reflected in the time complexity of the algorithm which grows by the factor of the squared input length from one subclass to the next powerful one. For all our grammars, this complexity is only linear in its size which is dominating in case of the large grammars for natural languages, where the sentences are usually short. In addition, we show how to generate the normal form required by our algorithm and discuss subclasses which can be analysed faster than the general case.

1. Introduction

In order to process natural languages, the first problem to be solved is to model the syntax formally. Many investigations, as for example [11], show that this cannot be done by context-free grammars. For context-sensitive grammars which are powerful enough it is known that they are PSPACE-complete. Thus, there is a trade-off between the power of the formalism and its analysis complexity. To solve this dilemma, much work has been done to characterize language classes completely in between context-free and context-sensitive languages being powerful enough to model the syntax of natural languages but endowed with a polynomial time analysis. Coupled-Context-Free Grammars represent such a formalism generalizing context-free grammars. Their suitability

* Corresponding author. E-mail: hotz|pitsch@cs.uni-sb.de.

[1] This research has been supported by a Graduiertenkolleg-fellowship of the Deutsche Forschungsgebreak meinschaft and by the ESPRIT Basic Research Action No. 6317 (ASMICS2).

to model syntactical phenomena follows from the fact that they include the languages generated by the Tree Adjoining Grammars (TAGs) of [14] as a subclass. The linguistic significance of TAGs is well explored (cf. e.g. [1, 13, 16, 17, 21]. Among other properties, both formalisms are able to model the linguistic phenomenon of cross-serial dependencies, which is not context-free but frequently appears in natural languages (cf. [2, 24]).

The formalism of Coupled-Context-Free Grammars has been introduced in [7, 9], and further investigated in [19, 20]. It belongs to the family of regulated string rewriting systems investigated in [5]. The increased generative capacity compared to context-free grammars is obtained by allowing to rewrite a certain number of elements simultaneously. Other regulated string rewriting systems as, e.g., the Scattered Context Grammars of [6] generalize context-free grammars by allowing simultaneous rewriting of arbitrary combinations of elements. In [5], it is shown that this results in languages which are not semilinear. But semilinearity is important since it formalizes the "constant-growth property" of natural languages. In contrast to these formalisms, all languages defined by Coupled-Context-Free Grammars are semilinear because of two restrictions on the rewriting steps. First, only those elements which were produced by the same rewriting step can be rewritten simultaneously. Second, the Coupled-Context-Free Grammars consider elements rewritten simultaneously as components of a parenthesis. Those can only be substituted if they form a parenthesis and they can only be substituted by sequences of parentheses which are correctly nested.

When characterizing Coupled-Context-Free Grammars by the maximal number of elements rewritten simultaneously – which we call the *rank* of the grammar – we get an infinite hierarchy. The generative capacity of Coupled-Context-Free Grammars grows with the rank. This permits to extend the context-free languages by continuously growing semilinear language classes. The smallest element of the hierarchy – the one of rank 1 – is represented by the context-free grammars. The next element, namely Coupled-Context-Free Grammars of rank 2, generates the same class of languages as the Tree Adjoining Grammars (TAGs) of [15] and [14]. Hence, all notions and algorithms designed for Coupled-Context-Free Grammars of rank 2 can easily be translated onto TAGs using the construction of the correspondence proof in [7].

Since Coupled-Context-Free Grammars are a natural generalization of context-free grammars, all efficient context-free parsing procedures represent possible starting points to develop an analysing algorithm for them. Here, the algorithm of Younger is one of the most attractive candidates because of its elegance. To use it, we need at first an appropriate generalization of the Chomsky normal form used by this algorithm. In [25], this approach was used to present the first polynomial time algorithm for the analysis of TAGs and therefore of Coupled-Context-Free Grammars of rank 2, too. The way via a generalization of this context-free algorithm was also used in [10] in order to develop an algorithm analysing TAGs. In contrast to the procedure in [25], the algorithm presented there is devided into an only context-free analysis and a

TAG-specific part. This partition also applies to the algorithm presented here. None of both algorithms for TAGs has been proven to be faster than $O(n^6)$, where n denotes the length of the input. In contrast to the two results just mentioned, we generalize the normal form as well as the algorithm for Coupled-Context-Free Grammars of arbitrary rank. Altogether, this is actually the first algorithm dealing with the analysis of grammars of arbitrary rank. The time complexity of the algorithm presented here amounts to $O(|P| \cdot n^{3l})$, where $|P|$ denotes the number of rewriting rules, n the length of the input, and l the rank of the grammar. For $l = 2$, this is asymptotically as fast as the algorithms for TAGs are. For $l > 2$, this time bound can still be improved to $O(|P| \cdot n^{2l+2})$ by constructing a more restrictive normal form which nevertheless exists for all grammars of rank $l > 2$. Thus, when processing a parenthesis of rank $l > 2$, the first two "partial parentheses" cost the factor n^3, while all the others only cost n^2. The necessary construction only increases the size of the grammar by the factor l.

When analysing natural languages one often has to deal with rather short sentences while the underlying grammar is of an enormous size. Therefore, the time complexity's dependency on the size of the grammar is important. Even for large ranks, here, the time complexity depends only linearly on the size of the grammar. In contrast to our algorithm, the time complexity of all parsing procedures for TAGs and Coupled-Context-Free Grammars of rank 2, respectively, known so far including the Earley-based ones in [22] and in [7], respectively, depends at least quadratically on the size of the grammar.

In addition, we characterize subclasses of all Coupled-Context-Free Grammars of some fixed rank $l \geqslant 2$ with a reduced analysis complexity. This is in contrast to the context-free case where the algorithm of Younger always needs the worst-case complexity. This fact is usually used as an argument *not* to use this algorithm for any generalization. In particular, this adaptive behaviour is a consequence of our new second step specifically designed for Coupled-Context-Free Grammars, which obviously reflects special properties of the grammar, e.g., a limited direct ambiguity of its productions.

The paper starts with the definition of Coupled-Context-Free Grammars. Then, the normal form and the algorithm for context-free grammars are shortly recalled. Here, the algorithm of Younger is slightly modified such that it determines graphs instead of matrices. The generalized procedure is essentially based on this graph. To be able to construct it for Coupled-Context-Free Grammars of any rank, the normal form of Chomsky is first generalized. The aptitude of our generalization for constructing an efficient algorithm follows from the parsing procedure itself and its complexity. To ease the understanding, we first present the generalized procedure for Coupled-Context-Free Grammars of rank 2. Subsequently, we prove by construction that our generalized Chomsky normal form can be generated for any Coupled-Context-Free Grammar. Finally, some subclasses of the whole hierarchy of languages are discussed which can be parsed faster than the general case. From here, the improvement of the general complexity follows.

2. Coupled-Context-Free Grammars

In this section, we present the formalism of Coupled-Context-Free Grammars in detail. They generalize the idea of a "coupled" substitution in [12] and were first presented in [8]. An extensive characterization of Coupled-Context-Free Grammars can be found in [7] and [9].

The Coupled-Context-Free Grammars are defined over the extended semi-Dyck sets which are a generalization of semi-Dyck sets. Elements of these sets can be regarded as sequences of parentheses that are correctly nested. Semi-Dyck sets play an important role in the theory of formal languages. Chomsky and Schützenberger [4] show that they are generators of context-free languages. In order to extend the family of context-free languages by using semi-Dyck sets we consider parentheses of arbitrary finite order defined as follows:

Definition 1 (*Parentheses set*). A finite set $\mathscr{K} := \{(k_{i,1}, \ldots, k_{i,m_i}) \mid i, m_i \in \mathbb{N}\}$ is a *parentheses set* if and only if it satisfies

$$k_{i,j} \neq k_{l,m} \quad \text{for } i \neq l \quad \text{or } j \neq m.$$

The elements of $\mathscr{K}$ are called *parentheses*. All parentheses of a fixed length $r \geqslant 1$ are contained in

$$\mathscr{K}[r] := \{(k_{i,1}, \ldots, k_{i,m_i}) \mid (k_{i,1}, \ldots, k_{i,m_i}) \in \mathscr{K}, m_i = r\}$$

where $\mathscr{K}[0] := \{\varepsilon\}$. ($\varepsilon$ is the empty word.) The set of all components of parentheses in $\mathscr{K}$ is denoted by

$$comp(\mathscr{K}) := \{k_{i,j} \mid (k_{i,1}, \ldots, k_{i,j}, \ldots, k_{i,m_i}) \in \mathscr{K}\}.$$

Straightforward from this, we get the generalization of the usual semi-Dyck sets as follows.

Definition 2 (*Extended semi-Dyck set*). Let $\mathscr{K}$ be a parentheses set and T an arbitrary set fulfilling $T \cap comp(\mathscr{K}) = T \cap \mathscr{K} = \emptyset$. $ED(\mathscr{K}, T)$, the *extended semi-Dyck set over $\mathscr{K}$ and T*, is a proper subset of $(comp(\mathscr{K}) \cup T)^*$ inductively defined by

(E1) $T^* \subseteq ED(\mathscr{K}, T)$.

(E2) $\mathscr{K}[1] \subseteq ED(\mathscr{K}, T)$.

(E3) $u_1, \ldots, u_r \in ED(\mathscr{K}, T), (k_1, \ldots, k_{r+1}) \in \mathscr{K}[r+1] \Longrightarrow$
$k_1 u_1 \cdots k_r u_r k_{r+1} \in ED(\mathscr{K}, T)$.

(E4) $u, v \in ED(\mathscr{K}, T) \Longrightarrow u \cdot v \in ED(\mathscr{K}, T)$.

(E5) $ED(\mathscr{K}, T)$ is the smallest set fulfilling the conditions (E1)–(E4).

Now, we define a procedure to generate new elements in $ED(\mathscr{K}, T)$ starting from given ones.

Definition 3 (*Parenthesis Rewriting System*). A *Parenthesis Rewriting System* over $ED(\mathscr{K},T)$ is a finite, nonempty set P of productions of the form

$$\{(k_1,\dots,k_r) \to (\alpha_1,\dots,\alpha_r) \mid (k_1,\dots,k_r) \in \mathscr{K}, \alpha_1 \cdots \alpha_r \in ED(\mathscr{K},T)\}.$$

The right-hand side of p for $p := (X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$ is denoted by $rhs(p) := (\alpha_1,\dots,\alpha_r)$.

Now, we are able to define our grammars. The term "coupled" denotes the fact that here, a certain number of context-free rewritings is executed in parallel and controlled by the definition of $\mathscr{K}$.

Definition 4 (*Coupled-Context-Free Grammar*). A *Coupled-Context-Free Grammar* over $ED(\mathscr{K},T)$ is an ordered 4-tuple $G = (\mathscr{K},T,P,S)$ where P is a Parenthesis Rewriting System over $ED(\mathscr{K},T)$ and $S \in \mathscr{K}[1]$. Therefore, $\mathscr{K}$ can be regarded as a set of coupled nonterminals. The set of all these grammars is denoted by *CCFG*.

At last, we still need an appropriate definition of derivation in *CCFG*. Let $G = (\mathscr{K},T,P,S) \in CCFG$ and $V := comp(\mathscr{K}) \cup T$. We define the relation $\Rightarrow_G$ as a subset of $V^* \times V^*$ consisting of all *derivation steps of rank* r for G with $r \geqslant 1$. $\varphi \Rightarrow_G \psi$ holds for $\varphi,\psi \in V^*$ if and only if

$$\text{there exist } u_1,u_{r+1} \in V^*,\ u_2,\dots,u_r \in ED(\mathscr{K},T),\ (k_1,\dots,k_r) \to (\alpha_1,\dots,\alpha_r) \in P$$

$$\text{such that } \varphi = u_1k_1u_2k_2 \cdots u_rk_ru_{r+1} \text{ and } \psi = u_1\alpha_1u_2\alpha_2 \cdots u_r\alpha_ru_{r+1}\,.$$

$\overset{*}{\Rightarrow}_G$ denotes the reflexive, transitive closure of $\Rightarrow_G$. Obviously, for the above φ and ψ, $u_1 \cdot u_{r+1} \in ED(\mathscr{K},T)$ follows from $S \overset{*}{\Rightarrow}_G \varphi$ since the result of the substitution is a sequence of parentheses correctly nested if and only if the original word was. A sequence $\varphi_1,\dots,\varphi_n$ with $\varphi_i \Rightarrow_G \varphi_{i+1}$ for all $1 \leqslant i < n$ and $\varphi_1 = \varphi$, $\varphi_n = \psi$ is called a *derivation of ψ from φ in G*. The language generated by G is

$$L(G) := \{w \in T^* \mid S \overset{*}{\Rightarrow}_G w\}.$$

Notation 1. Let $\alpha_p := \alpha_1 \cdot \ldots \cdot \alpha_r$ and $\vartheta_p := (\alpha_1,\dots,\alpha_r)$ for any fixed $p := (X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$. If we need to denote precisely the nonterminals appearing in $rhs(p)$, this is done by the jth nonterminal occuring from the left on in $(\alpha_1,\dots,\alpha_r)$, $\alpha_1 \cdots \alpha_r \in ED(\mathscr{K},T)$, $r \geqslant 1$, which is defined as

$$NT_j(\vartheta_p) := \begin{cases} (Y_1,\dots,Y_m) \in \mathscr{K} & \text{if } \alpha_p = \gamma_1A^{(1)}\gamma_2 \dots A^{(j-1)}\gamma_jY_1\gamma_{j+1} \text{ while } A^{(i)} = X_1 \\ & \text{for some } (X_1,\dots,X_t) \in \mathscr{K} \text{ and } \gamma_i \neq \gamma_{i,1}B\gamma_{i,2} \\ & \text{for all } B = Z_1, (Z_1,\dots,Z_s) \in \mathscr{K} \text{ and } 1 \leqslant i \leqslant j, \\ \varepsilon & \text{else.} \end{cases}$$

In order to be able to describe the generative capacity of Coupled-Context-Free Grammars of different ranks exactly, we need the following notions:

Definition 5 (*Rank, CCFG(l)*). For any $G = (\mathscr{K}, T, P, S) \in CCFG$, let the *rank* of G be defined as $rank(G) := \max\{r \mid (k_1, \ldots, k_r) \in \mathscr{K}\}$. Then, we define for all $l \geqslant 1$

$$CCFG(l) := \{G \in CCFG \mid rank(G) \leqslant l\}.$$

Example 1. $G := (\{S, (X_1, X_2)\}, \{a, b, c, d\}, P, S)$ is in $CCFG(2)$ and generates the language $\{a^n b^n c^n d^n \mid n \geqslant 0\}$, if we define $P := \{S \to aX_1bcX_2d, (X_1, X_2) \to (aX_1b, cX_2d) \mid (\varepsilon, \varepsilon)\}$, e.g.

$$S \Rightarrow_G aX_1bcX_2d \Rightarrow_G aaX_1bbccX_2dd \Rightarrow_G aaaX_1bbbcccX_2ddd \Rightarrow_G aaabbbcccddd.$$

In general, $G := (\{S, (X_1, X_2, \ldots, X_i)\}, \{a_1, a_2, \ldots, a_{2i}\}, P, S)$ is in $CCFG(i)$, if we define $P := \{S \to a_1X_1a_2a_3X_2a_4 \ldots a_{2i-1}X_ia_{2i}, (X_1, X_2, \ldots, X_i) \to (a_1X_1a_2, a_3X_2a_4, \ldots, a_{2i-1}X_ia_{2i}) \mid (\varepsilon, \varepsilon, \ldots, \varepsilon)\}$. This one generates the language $\{a_1^n a_2^n \ldots a_{2i-1}^n a_{2i}^n \mid n \geqslant 0\}$.

Example 2. The language $\{ww \mid w \in \{a, b\}^*\}$ modelling cross-serial dependencies is generated by the grammar $G := (\{S, (X_1, X_2)\}, \{a, b\}, P, S) \in CCFG(2)$, if we define $P := \{S \to X_1X_2, (X_1, X_2) \to (aX_1, aX_2) \mid (bX_1, bX_2) \mid (\varepsilon, \varepsilon)\}$.

The following theorem proven in [7] shows that *CCFG* builds up an infinite hierarchy of languages and, at the same time, represents a proper extension of context-free grammars not exceeding the power of context-sensitive grammars.

Theorem 1 (Hierarchy). *Let CFL be the family of all context-free, CSL the family of all context-sensitive languages, TAL the family of languages generated by TAGs and CCFL(l) the one generated by CCFG(l). It holds*:

(1) $CFL = CCFL(1)$, $TAL = CCFL(2)$.
(2) $CCFL(l) \subsetneq CCFL(l+1)$ *for all* $l \geqslant 1$.
(3) $CCFL(l) \subsetneq CSL$ *for all* $l \geqslant 1$.

Sometimes, it is useful to "neglect" the relations between the components of a parenthesis defined by $G = (\mathscr{K}, T, P, S) \in CCFG$. Then, we investigate the grammar $G' := (comp(\mathscr{K}), T, P', S)$ for

$$P' := \bigcup_{(k_1, \ldots, k_r) \to (\alpha_1, \ldots, \alpha_r) \in P} \{k_i \to \alpha_i \mid 1 \leqslant i \leqslant r\}.$$

Since G' is certainly a context-free grammar, we denote G' (resp. P') by $CF(G)$ (resp. $CF(P)$) in the sequel. Obviously, G' satisfies $L(G) \subseteq L(G')$.

3. The context-free case

To be able to generalize the Chomsky normal form and the algorithm of Younger consistently, we shortly recall the context-free steps. The section finishes by presenting the modification of the original algorithm which renders it producing a graph instead of a matrix. This graph represents all possible derivation trees for the input in an overlapped manner.

3.1. *The normal form of Chomsky*

Definition 6 (*Chomsky Normal Form* [3]). A context-free grammar $G = (N,T,P,S)$ is in *Chomsky normal form* if and only if it fulfills the conditions (1) and (2) given as

(1) $\varepsilon \in L(G) \implies S \to \varepsilon \in P$ and $P \setminus \{S \to \varepsilon\} \subseteq N \times ((N \setminus \{S\})^2 \cup T)$.

(2) $\varepsilon \notin L(G) \implies P \subseteq N \times (N^2 \cup T)$.

Based on this definition, it holds

Theorem 2 (Chomsky [3]). *For any context-free grammar $G=(N,T,P,S)$, there exists a context-free grammar $G' = (N',T,P',S)$ in Chomsky normal form fulfilling $L(G) = L(G')$.*

Let $p_{\max}$ be the maximal length of $rhs(p)$ for any $p \in P$. Given G, the grammar G' can be determined in time $O(p_{\max} \cdot |P| \cdot |N|)$ extending the size of P' to $O(p_{\max} \cdot |P| \cdot |N|)$. In [26], this normal form is used to develop the efficient parsing procedure for context-free grammars described below.

3.2. *The original Younger algorithm*

Its input is a context-free grammar $G = (N,T,P,S)$ in Chomsky normal form and some $w \in T^*$, $n := |w|$. To analyse w, the $n \times n$-matrix $\mathcal{Y} = (y_{i,j})$ is determined such that $y_{i,j}$ consists of subsets of N fulfilling

$$X \in y_{i,j} \iff X \overset{*}{\Rightarrow}_G w_i \ldots w_{i+j-1}.$$

We get a triangular matrix since for $i+j-1 > n$, the entries $y_{i,j}$ consist of the empty set. Obviously, it holds $w \in L(G) \iff S \in y_{1,n}$. This procedure can be formulated as the following algorithm:

```
procedure Younger($G = (N,T,P,S), w$)
begin
  for $i := 1$ to $n$ do $y_{i,1} := \{X \mid X \to w_i \in P\}$ od;
  for $j := 2$ to $n$ do
    for $i := 1$ to $n-j+1$ do
      $y_{i,j} := \emptyset$;
      for $k := 1$ to $j-1$ do
(*)       $y_{i,j} := y_{i,j} \cup \{X \mid \exists A \in y_{i,k}\ \exists B \in y_{i+k,j-k}: X \to AB \in P\}$
      od;
    od;
  od;
end;
```

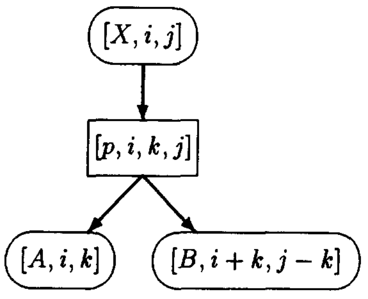

Fig. 1. Context-free graph substructure.

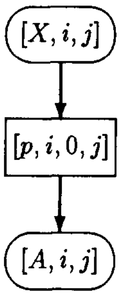

Fig. 2. New graph substructures.

If we realize the sets $y_{i,j}$ as one-dimensional boolean arrays of length $|N|$, step $(*)$ only costs $O(|P|)$ steps for any fixed i, j, and k. Thus, the total complexity of the algorithm is dominated by the three for-loops and amounts to $O(|P| \cdot n^3)$.

3.3. The Derivation Graph

To be able to perform the coupled analysis efficiently, we modify the algorithm such that it determines a graph storing all possible derivation trees for the input instead of the matrix $\mathcal{Y}$. At first, we produce a "nonterminal" node marked by $[X,i,j]$ for any $X \in y_{i,j}$. Then, we could insert directed edges $([X,i,j],[A,i,k])$ and $([X,i,j],[B,i+k,j-k])$ for any $k < j$ and $X \to AB \in P$, if $A \in y_{i,k}$ and $B \in y_{i+k,j-k}$ holds. But thus, it is not obvious which pair of edges realizes one fixed production. Therefore, the edges are constructed via a "production" node for $p = X \to AB \in P$ and all suitable k as shown in Fig. 1. Consequently, the whole graph consists of a superposition of such substructures. This construction can be formalized as follows.

Definition 7 (*Derivation Graph*). Let $G = (N,T,P,S)$ a context-free grammar. For any $w \in T^*$, $n := |w|$, the directed *Derivation Graph of w in G*, is defined as *Graph*(G,w) $= (V_{Graph(G,w)}, E_{Graph(G,w)})$. Here, the node set is defined as

$$V_{Graph(G,w)} := NN \,\dot{\cup}\, PN \,\dot{\cup}\, TN$$

consisting of the set of all *nonterminal* nodes

$$NN := \{[X,i,j] \mid 1 \leqslant i,j \leqslant n,\ i+j-1 \leqslant n,\ X \overset{*}{\Rightarrow}_G w_i \ldots w_{i+j-1}\},$$

the set of all *expanding production* nodes

$$PN := \{[p,i,k,j] \mid p = X \to AB \in P,\ 1 \leqslant i,j,k \leqslant n,\ i+j-1 \leqslant n,$$
$$k < j,\ A \overset{*}{\Rightarrow}_G w_i \ldots w_{i+k-1},\ B \overset{*}{\Rightarrow}_G w_{i+k} \ldots w_{i+j-1}\},$$

and the set of all *terminating production* nodes $TN := \{[p,i] \mid X \to w_i \in P\}$. At the same time, it holds

$$E_{Graph(G,w)} := \bigcup_{\substack{[p,i]\in TN,\\ p=X\to w_i}} \{([X,i,1],[p,i])\}$$
$$\cup \bigcup_{\substack{[p,i,k,j]\in PN,\\ p=X\to AB}} \{([X,i,j],[p,i,k,j]),([p,i,k,j],[A,i,k]), ([p,i,k,j],[B,i+k,j-k])\}.$$

For any $G = (N,T,P,S)$ and $w \in T^*$, $Graph(G,w)$ can be constructed using at most $O(|P| \cdot n^3)$ operations by slightly modifying the original Younger algorithm. Obviously, this does not influence its correctness and complexity. It holds $w \in L(G) \iff [S,1,n] \in V_{Graph(G,w)}$.

4. The normal form for $G \in CCFG(l)$, $l \geqslant 1$

The condition $P \subseteq N \times N^2$ in the definition of the Chomsky normal form combines the demands that

1. the productions must be as short as possible to be analysed efficiently, and that
2. the right-hand sides of the productions have to be at least of length 2 if the normal form shall guarantee that all derivation trees for w are at most of height $|w|$. Rules as $X \to Y$ where X, $Y \in N$ would destroy this property.

In order to keep this intuition when generalizing this normal form, we have to ask what could prevent derivation trees for $w \in T^*$ relative to some Coupled-Context-Free Grammar $G = (\mathcal{K},T,P,S)$ from being heigher than $|w|$. In any case, the prohibition of partial productions producing ε and the prohibition of productions $(X_1,\dots,X_r) \to (Y_1,\dots,Y_r)$ where $(Y_1,\dots,Y_r) \in \mathcal{K}[r]$ are certainly necessary. To allow an efficient analysis, we additionally need that any element of $CF(P)$ does not produce more than two nonterminal components. It can be shown that these conditions suffice (cf.[18]). Thus, we get

Definition 8 (*Generalized Chomsky normal form*). A grammar $G = (\mathcal{K},T,P,S) \in CCFG$ is in *generalized Chomsky normal form* if and only if it fulfills the conditions (1) and (2) given as

(1) $\varepsilon \in L(G) \implies S \to \varepsilon \in P$ and $p \in P \implies S \neq NT_j(rhs(p)) \forall j$.

(2) $(X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P \setminus \{S \to \varepsilon\} \implies$ for all $1 \leqslant i \leqslant r$ holds:

(a) $1 \leqslant |\alpha_i| \leqslant 2$.

(b) $(\alpha_1,\dots,\alpha_r) \notin \mathcal{K}[r]$.

(c) $|\alpha_i| = 2 \implies \alpha_i \in (comp(\mathcal{K}))^2$.

For $l = 1$, this equals the definition of [3]. In Section 6, it is shown that this normal form can be constructed for any $G \in CCFG$. Now, we present how to exploit the generalized normal form to construct an efficient parsing procedure for $CCFG$.

5. The Algorithm

The complete analysis consists of two steps. At first, the context-free graph corresponding to the Coupled-Context-Free Grammar is constructed. The definition of the generalized normal form possibly causes cycles in this graph not existing in the context-free case because we allow partial productions $X_i \to Y_j$ for $(X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$ if $(\alpha_1,\dots,\alpha_r) \notin \mathscr{K}[r]$ holds. The second step of the procedure recursively traverses the graph controlled by the parentheses structures as defined by the grammar. Therefore, the cycles are always traversed coupled to noncyclic structures. This prevents the algorithm from running into an infinite loop.

5.1. The modified Derivation Graph

Let $G = (\mathscr{K}, T, P, S) \in CCFG$ be in generalized Chomsky normal form and $w \in T^*$. In principle, *Graph*$(CF(G), w)$ is constructed as before. Only productions $X \to Y \in CF(P)$ have to be treated with a special procedure. As to the procedure *Younger*, it suffices to insert the line

$$y_{i,j} := y_{i,j} \cup \{X \mid \exists A \in y_{i,j} \text{ such that } X \overset{*}{\Rightarrow}_{CF(P)} A\};$$

directly after the **for**-loop surrounding step $(*)$. This corresponds to a compression of all productions $X \to Y$ as transitive closure. For any $A \in comp(\mathscr{K})$, the set of all the elements of $comp(\mathscr{K})$ reaching A via the relation $\overset{*}{\Rightarrow}_{CF(G)}$ is defined by $CF(G)$. Thus, for any subset of $comp(\mathscr{K})$, the relevant set can be predetermined once for all inputs. This does not influence the costs of the analysis.

Instead of these sets, we can also predetermine the subgraphs equivalent. For all $X \in comp(\mathscr{K})$, they are defined as $help(X) := (V^{NN}_{help}(X) \cup V^{PN}_{help}(X), E_{help}(X))$, where

$$V^{NN}_{help}(X) := \{A \in comp(\mathscr{K}) \mid A \overset{*}{\Rightarrow}_{CF(G)} X\},$$

$$V^{PN}_{help}(X) := \{p \mid p \in CF(P), rhs(p) \in V^{NN}_{help}(X)\},$$

$$E_{help}(X) := \{(A, p), (p, B) \mid A, B \in V^{NN}_{help},\ p = A \to B \in CF(P)\}.$$

Here, the indices i and j of the nodes have to remain unset as long as they are undefined for the corresponding nonterminal but we know that they all are identical inside a subgraph. The insertion of a subgraph consists in filling the indices in the nodes and in identifying identical nodes appearing in the inserted as well as in the original graph. Its realization inserted at the same place as before is:

$$NN := NN \cup \{[A, i, j] \mid A \in V^{NN}_{help}(X) \text{ for } [X, i, j] \in NN\};$$

$$PN := PN \cup \{[p, i, 0, j] \mid p \in V^{PN}_{help}(X) \text{ for } [X, i, j] \in NN\};$$

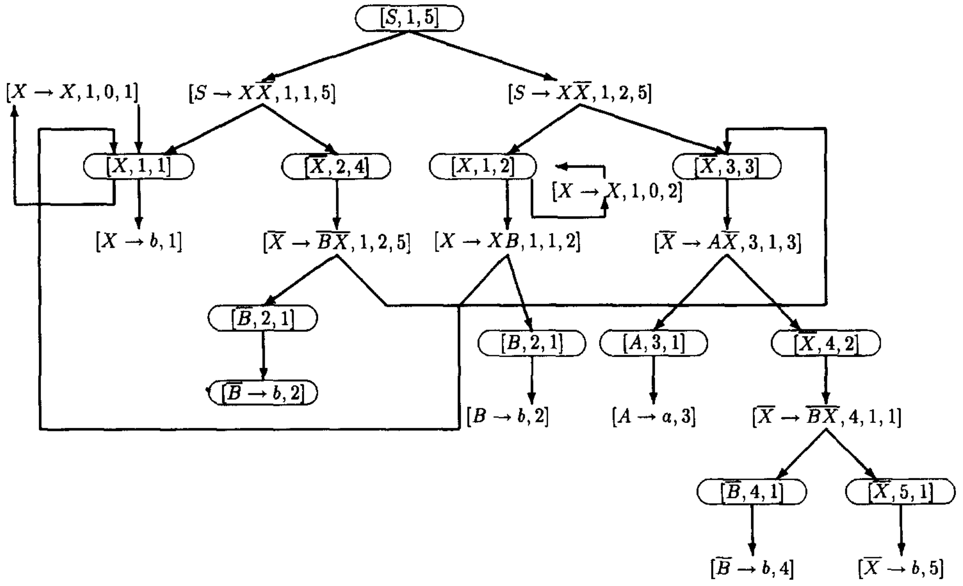

Fig. 3. The derivation graph resulting from Example 3.

$$
\begin{aligned}
E_{Graph(CF(G),w)} :=\ & \\
E_{Graph(CF(G),w)} & \cup \{([A,i,j],[p,i,0,j]) \mid (A,p) \in E_{help}(X) \text{ and } [X,i,j] \in NN\} \\
& \cup \{([p,i,0,j],[B,i,j]) \mid (p,B) \in E_{help}(X) \text{ and } [X,i,j] \in NN\};
\end{aligned}
$$

The procedure now resulting from the procedure *Younger* is called *modified_Younger_Graph*. As in the context-free case, the graph stores all derivation trees for w relative to $CF(G)$. It remains to show how to single out all those trees which are also correct in the sense of the coupling.

Example 3. Let $G = (\mathscr{K}, T, P, S) \in CCFG(2)$ be defined via

$$
\begin{aligned}
P := \{& S \to X\overline{X}, \\
& (X,\overline{X}) \to (X, A\overline{X}) \mid (XB, \overline{BX}) \mid (b,b), \\
& (B,\overline{B}) \to (b,b), A \to a\}.
\end{aligned}
$$

Obviously, G is in generalized Chomsky normal form. We use G to explain the steps of the algorithm. At first, Fig. 3 shows the graph generated by *modified_Younger_Graph* $(CF(G),w)$ where $w := bbabb$. To simplify the situation, we omit all nodes in the figure which cannot be reached from $[S,1,5]$. Anyway, they are never used in a derivation $S \overset{*}{\Rightarrow}_{CF(G)} bbabb$.

5.2. $G \in CCFG(2)$

Let $Graph(CF(G),w)$ for $G = (\mathscr{K}, T, P, S) \in CCFG(2)$ in generalized Chomsky normal form and $w \in T^*$, $n := |w|$, be determined. The idea of the second step consists in traversing the derivation graph in a depth-first manner. We start at $[S,1,n]$.

All edges starting here are investigated in turn by asking "*Is there any derivation tree for $w_1 \dots w_n$ below this edge which is correct relative to the coupling*?". This is done as long as no edge answers with "*There is.*", respectively *true*, or as all edges were investigated resulting in a negative answer. In the first case, $[S, 1, n]$ is marked by *true* denoting that $S \overset{*}{\Rightarrow} w_1 \dots w_n$ holds, otherwise by *false* expressing the contrary.

In the sequel, we discuss some important cases concerning the transmission of information about the coupling inside the sub-derivation trees. We distinguish the inquiries following the number of nodes inquired in parallel. Thus, unary inquiries investigate nodes $[X,i,j]$ where $X \in \mathscr{K}[1]$ holds while two-ary ones look at pairs of nodes $[X,i,j]$, $[Y,l,r]$ for $(X,Y) \in \mathscr{K}[2]$. Terminating productions are ignored since their handling is obvious.

The whole graph consists of substructures as shown in Figs. 1 and 2. Unary inquiries only investigate structures of the first kind since productions $X \to Y$ where $X, Y \in \mathscr{K}[1]$ are forbidden. Let us look at Fig. 1. The answer of the question "$X \overset{*}{\Rightarrow}_G w_i \dots w_{i+j-1}$?" denoting the unary inquiry for $[X,i,j]$ and determining the mark at the node $[X,i,j]$ is the logical *or* of all answers given by edges starting at $[X,i,j]$. Therefore, we explain the procedure for production nodes reached by those edges. It is iterated for all those $[p,i,k,j]$ as often as there remains an edge disregarded while the answer of all the others is *false*.

If $A,B \in \mathscr{K}[1]$ holds, the question "$X \overset{*}{\Rightarrow} w_i \dots w_{i+j-1}$?" is forwarded via $[p,i,k,j]$ in the form of the questions "$A \overset{*}{\Rightarrow} w_i \dots w_{i+k-1}$?" and "$B \overset{*}{\Rightarrow} w_{i+k} \dots w_{i+j-1}$?". When they both were recursively investigated, the nodes $[A,i,k]$ and $[B,i+k,j-k]$ are marked by the answer. The answer for $[X,i,j]$ via the node $[p,i,k,j]$ is the logical *and* of these marks. This corresponds to the fact that if there is a derivation tree relative to G from A to $w_i \dots w_{i+k-1}$ and from B to $w_{i+k} \dots w_{i+j-1}$, there is a derivation tree from X to $w_i \dots w_{i+j-1}$ because we have $X \to AB \in P$.

If $(A,B) \in \mathscr{K}[2]$ holds, the nodes $[A,i,k]$ and $[B,i+k,j-k]$ have to be treated in mutual dependence. Therefrom, the first two-ary inquiries arise. Thus, let the algorithm have to decide for any pair of nodes $[X_1,i,j]$ and $[X_2,l,r]$, $i+j \leqslant l$ and $(X_1,X_2) \in \mathscr{K}[2]$, whether it holds $X_1X_2 \overset{*}{\Rightarrow}_G w_i \dots w_{i+j-1} w_l \dots w_{l+r-1}$. Again, the edges starting at these nodes are investigated in turn. Now, we always have to investigate pairs of edges. At first, we have to test whether the two drain nodes $[p,i,k,j]$ and $[q,l,s,r]$ fulfill the property $(p,q) \in P$. If they do, let the situation to investigate be as shown in Fig. 4. the inquiry here depends on A, B, C, and D. If, for instance, $(A,D),(B,C) \in \mathscr{K}[2]$ holds, the question "$X_1X_2 \overset{*}{\Rightarrow}_G w_i \dots w_{i+j-1} w_l \dots w_{l+r-1}$?" is split via this edge pair into the questions "$AD \overset{*}{\Rightarrow}_G w_i \dots w_{i+k-1} w_{l+s} \dots w_{l+r-1}$?" respectively "$BC \overset{*}{\Rightarrow}_G w_{i+k} \dots w_{i+j-1} w_l \dots w_{l+s-1}$?". If we have $(A,C) \in \mathscr{K}[2]$ and $B,D \in \mathscr{K}[1]$, the original question is split into a two-ary inquiry for $([A,i,k],[C,l,s])$ and two unary inquiries for $[B,i+k,j-k]$ and $[D,l+s,r-s]$. The answer for the original pair $([X_1,i,j],[X_2,l,r])$ consists in the logical *and* of these answers found recursively.

Now, we look at inquiries which investigate substructures as shown in Fig. 2. At first, we deal with a two-ary inquiry as shown in Fig. 5. Its splitting results from the

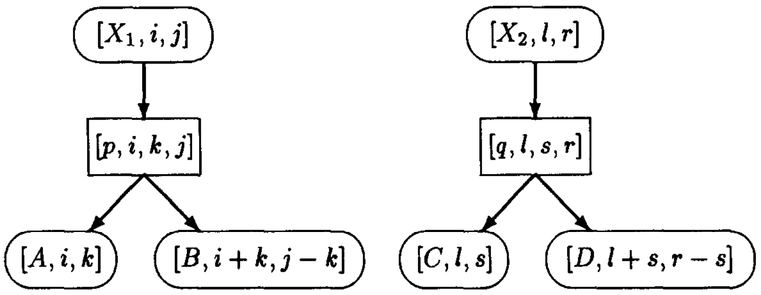

Fig. 4. Simple kind of a two-ary inquiry.

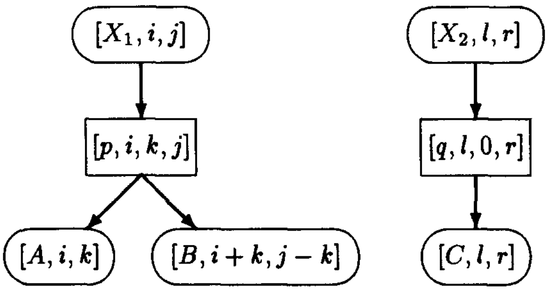

Fig. 5. More difficult kind of a two-ary inquiry.

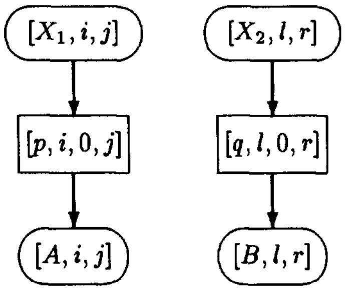

Fig. 6. Most difficult two-ary inquiry.

properties of A, B, and C, too. The new point is that because of the right part, some of the cycles are possibly taken into consideration here. Can this cause the algorithm to run into an infinite loop? Obviously, the splitting of the inquiry as performed here causes a new inquiry for at most one nonterminal node showing the same indices as before. In Fig. 5, these are l and r. Thus, we cannot produce an inquiry for $([X_1,i,j],[X_2,l,r])$ again, while it is possible to produce inquiries of the kind $([X_1,i,t],[X_2,l,r])$ where $t<j$.

The last case to investigate is shown in Fig. 6. At first glance, it seems to be a problem that no pair of indices is modified when splitting the inquiry. This seems to produce cycles inside the algorithm in any case. But because of the definition of the normal form, $A,B \in \mathscr{K}[1]$ holds. It directly follows that no infinite loop is possible, since on the one hand, $[A,i,j]$ or $[B,l,r]$, respectively, cannot cause a two-ary inquiry of $([X_1,i,j],[X_2,l,r])$ all by itself. On the other hand, they cannot cause such an inquiry together since $(X_1,X_2) \notin \mathscr{K}[2]$ holds. Thus, our algorithm terminates since cycles inside $Graph(G,w)$ are only considered coupled to structures demanding a splitting.

The complete second step of our algorithm consists in the combination of all the partial steps just explained to the recursive functions *test_1* resp. *test_2* as given below treating unary resp. two-ary inquiries. They become efficient since any node (pair of nodes) is marked such that an inquiry of this node (pair of nodes) has to be executed only once. If it is inquired a second time, the mark can be used instead of a new inquiry of the subtrees. In addition, the mark can be accessed in constant time. To simplify the algorithm, we assume

- that there exists a global matrix $K[|NN|,|NN|]$ which is indexed by elements of NN. Its entries lie in $\{-1,0,1\}$, where -1 denotes the situation where no inquiry has been performed so far. 1 stands for *true* as result of the inquiry, 0 for *false*. The information about nonterminal nodes marked by $X \in \mathscr{K}[1]$ is stored in $K[[X,i,j],[X,i,j]]$.
- that for all $[X,i,j] \in NN$, $\mathscr{P}([X,i,j])$ denotes the list of all production nodes $[p,i,k,j]$ directly reached by $([X,i,j],[p,i,k,j]) \in E_{Graph(G,w)}$. The list operations *head* and *next* are defined as usual. In addition, *prod* outputs the production belonging to a fixed element of the list.

```
function test_1([X,i,j]):boolean
begin
  if $j = 1$
  then /* $[X,i,j] \in E_{Graph(CF(G),w)} \Longrightarrow X \to w_i \in CF(P)$ */
       $K[[X,i,j],[X,i,j]] := 1$ fi;
  if  $K[[X,i,j],[X,i,j]] = -1$
  then /* it holds $j \neq 1$ */
       $K[[X,i,j],[X,i,j]] := 0$;
       $q := head(\mathscr{P}([X,i,j]))$;
       while  $q \neq nil$  and  $K[[X,i,j],[X,i,j]] \neq 1$
       do /* let $[prod(q),i,k,j]$ the node under consideration */
          case $prod(q)$  of
               $X \to A\bar{A}$, $(A,\bar{A}) \in \mathscr{K}[2]$: $answer := test\_2([A,i,k],[\bar{A},i+k,j-k])$;
               $X \to N_1N_2$, $N_1,N_2 \in \mathscr{K}[1]$: $answer := test\_1([N_1,i,k])$ and
                                                                  $test\_1([N_2,i+k,j-k])$;
          esac;
          $K[[X,i,j],[X,i,j]] := answer$;
          $q := q \to next$;
       od;
  fi;
  return($K[[X,i,j],[X,i,j]]$);
end
```

```
function $test\_2([X,i,j],[Y,l,r])$:boolean
begin
  if  $K[[X,i,j],[Y,l,r]] = -1$
  then
      $K[[X,i,j],[Y,l,r]] := 0$;
      $p_1 := head(\mathscr{P}([X,i,j]))$;
      while  $p_1 \neq nil$  and  $K[[X,i,j],[Y,l,r]] \neq 1$
      do
          $p_2 := head(\mathscr{P}([Y,l,r]))$;
          while $(prod(p_1),prod(p_2)) \notin \mathscr{K}[2]$  and  $p_2 \neq nil$
          do $p_2 := p_2 \rightarrow next$ od;
          if  $p_2 \neq nil$
          then /* $(prod(p_1),prod(p_2)) \in \mathscr{K}[2]$ holds */
               /* split the original inquiry following the structure of $rhs(p)$; */
               /* details can be found below in structure_2                     */
               $K[[X,i,j],[Y,l,r]] := structure\_2(p_1,p_2)$;
          fi;
          $p_1 := p_1 \rightarrow next$;
      od;
  fi;
  return($K[[X,i,j],[Y,l,r]]$);
end
```

The splitting of the original inquiry into partial inquiries can be done by a case-construct. For the sake of shortness, we present only the most important fragments of the function *structure_2* performing this job. All the omitted other possibilities can be treated completely analogous, only the indices of the recursive procedure calls change. W.l.o.g., we assume that for $j \neq 1$, the production node p_1 can be represented as $p_1 = [prod(p_1),i,k,j]$. The same holds for p_2, which is represented as $p_2 = [prod(p_2),l,s,r]$ if $r \neq 1$ holds. The indices just given are used in the presentation below.

```
function $structure\_2(p_1,p_2)$
begin
  case  $(prod(p_1),prod(p_2))$  of
  $(X,Y) \rightarrow (AB,\overline{B}\,\overline{A})$, $(A,\overline{A}),(B,\overline{B}) \in \mathscr{K}[2]$:
          answer := $test\_2([A,i,k],[\overline{A},l+s,r-s])$
                    and  $test\_2([B,i+k,j-k],[\overline{B},l,s])$;
  $(X,Y) \rightarrow (A\overline{A},B\overline{B})$, $(A,\overline{A}),(B,\overline{B}) \in \mathscr{K}[2]$:
          answer := $test\_2([A,i,k],[\overline{A},i+k,j-k])$
                    and  $test\_2([B,l,s],[\overline{B},l+s,r-s])$;
  $(X,Y) \rightarrow (AN_1,\overline{A}N_2)$, $(A,\overline{A}) \in \mathscr{K}[2]$, $N_1,N_2 \in \mathscr{K}[1]$:
          answer := $test\_2([A,i,k],[\overline{A},l,s])$
```

```
                    and  $test\_1([N_1, i+k, j-k])$
                    and  $test\_1([N_2, l+s, r-s])$;
                     ⋮
/* all possible combinations of $A, \bar{A}, N_1, N_2$ on the right hand side
                                                    of the production */
     ⋮
$(X,Y) \to (AN_1, N_2\bar{A})$, $(A,\bar{A}) \in \mathscr{K}[2]$, $N_1, N_2 \in \mathscr{K}[1]$:
     answer := $test\_2([A,i,k],[\bar{A}, l+s, r-s])$
                    and  $test\_1([N_1, i+k, j-k])$
                    and  $test\_1([N_2, l, s])$;
$(X,Y) \to (N_1N_2, N_3N_4)$, $N_1, N_2, N_3, N_4 \in \mathscr{K}[1]$:
     answer:= $test\_1([N_1,i,k])$ and $test\_1([N_2, i+k, j-k])$
                    and $test\_1([N_3,l,s])$ and $test\_1([N_4, l+s, r-s])$;
$(X,Y) \to (A, \bar{A}N)$, $(A,\bar{A}) \in \mathscr{K}[2]$, $N \in \mathscr{K}[1]$:
     answer := $test\_2([A,i,j],[\bar{A},l,s])$ and $test\_1([N, l+s, r-s])$;
                     ⋮
/* all possible combinations of $A, \bar{A}, N$ on the right hand side of the production */
                     ⋮
$(X,Y) \to (NA, \bar{A})$, $(A,\bar{A}) \in \mathscr{K}[2]$, $N \in \mathscr{K}[1]$:
     answer := $test\_1([N,i,k])$ and $test\_2([A, i+k, j-k],[\bar{A},l,r])$;
$(X,Y) \to (N_1N_2, N_3)$, $N_1, N_2, N_3 \in \mathscr{K}[1]$:
     answer := $test\_1([N_1,i,k])$ and $test\_1([N_2, i+k, j-k])$
                              and $test\_1([N_3,l,r])$;
$(X,Y) \to (N_1, N_2N_3)$, $N_1, N_2, N_3 \in \mathscr{K}[1]$:
     answer := $test\_1([N,i,j])$ and $test\_1([N_2,l,s])$
                              and $test\_1[N_3, l+s, r-s]$);
$(X,Y) \to (A\bar{A}, w_l)$, $(A,\bar{A}) \in \mathscr{K}[2]$:
     answer := $test\_2([A,i,j],[\bar{A}, i+k, j-k])$;
$(X,Y) \to (w_i, A\bar{A})$, $(A,\bar{A}) \in \mathscr{K}[2]$:
     answer := $test\_2([A,l,s],[\bar{A}, l+s, r-s])$;
$(X,Y) \to (N_1N_2, w_l)$, $N_1, N_2 \in \mathscr{K}[1]$:
     answer := $test\_1([N_1,i,j])$  and  $test\_1([N_2, i+k, j-k])$;
$(X,Y) \to (w_i, N_1N_2)$, $N_1, N_2 \in \mathscr{K}[1]$:
     answer := $test\_1([N_1,l,s])$  and  $test\_1([N_2, l+s, r-s])$;
$(X,Y) \to (N_1, N_2)$, $N_1, N_2 \in \mathscr{K}[1]$:
     answer := $test\_1([N_1,i,j])$  and  $test\_1([N_2,l,r])$;
$(X,Y) \to (N, w_l)$, $N \in \mathscr{K}[1]$:
     answer := $test\_1([N,i,j])$;
$(X,Y) \to (w_i, N)$, $N \in \mathscr{K}[1]$:
     answer := $test\_1([N,l,r])$;
$(X,Y) \to (w_i, w_l)$:
```

```
            answer := true;
    esac;
    return(answer);
end;
```

For any $G = (\mathscr{N}, T, P, S) \in CCFG(2)$ in generalized Chomsky normal form and any $w \in T^*$, $n := |w|$, the complete generalized Younger algorithm is:

```
procedure generalized_Younger(G, w)
begin
    modified_Younger_Graph(CF(G), w);
    if test_1([S, 1, n])
    then "accept"
    else "reject"
    fi;
end;
```

Example 4 (*Example* 3 *continued*). When applying *generalized_Younger* onto the grammar of Example 3, we get the following (cf. Fig. 3 to trace the corresponding paths in the graph):

- At first, we call *test*_1($[S,1,n]$). Here, $\mathscr{P}([S,1,n])$ consists of the elements $[S \to X\overline{X},1,1,5]$ and $[S \to X\overline{X},1,2,5]$. We go on with $[S \to X\overline{X},1,1,5]$. Here, the first case applies. Therefore, we have to call *test*_2($[X,1,1],[\overline{X},2,4]$).
  - Now, $\mathscr{P}([X,1,1])$ contains $[X \to X,1,0,1]$ and $[X \to b,1]$,
    while $\mathscr{P}([\overline{X},2,4])$ only contains $[\overline{X} \to \overline{BX},2,1,4]$.
    Obviously, neither $(X,\overline{X}) \to (X,\overline{BX}) \in P$ nor $(X,\overline{X}) \to (b,\overline{BX}) \in P$ holds.
    Therefore, $K[[X,1,1],[\overline{X},2,4]] = 0$ holds forever.
- Since the first derivation tree for *bbabb* was not correctly coupled, we investigate the next one, i.e. we call *test*_2($[X,1,2],[\overline{X},3,3]$).
  - Here, $\mathscr{P}([X,1,2])$ contains $[X \to XB,1,1,2]$ and $[X \to X,1,0,2]$,
    while $\mathscr{P}([\overline{X},3,3])$ only contains $[\overline{X} \to A\overline{X},3,1,3]$.
    We have $(X,\overline{X}) \to (X,A\overline{X}) \in P$. Therefore, we take the corresponding case in *structure_2* (i.e. $(X,Y) \to (A,N\overline{X})$) and get:
    answer := *test*_2 ($[X,1,2],[\overline{X},4,2]$) **and** *test*_1($[A,3,1]$)
    * *test*_1($[A,3,1]$) obviously answers true.
    * $\mathscr{P}([X,1,2])$ contains $[X \to XB,1,1,2]$ and $[X \to X,1,0,2]$,
      $\mathscr{P}([\overline{X},4,2])$ contains $[\overline{X} \to \overline{BX},4,1,1]$.
      Here, we have $(X,\overline{X}) \to (XB,\overline{BX}) \in P$. Therefore, we have to take the first case in *structure_2* (i.e. $(X,Y) \to (AB,\overline{BA})$) and get
      :*answer* := *test*_2($[X,1,1],[\overline{X},5,1]$) **and** *test*_2($[B,2,1],[\overline{B},4,1]$).
      Since we have $(X,\overline{X}) \to (b,b) \in P$ and $(B,\overline{B}) \to (b,b) \in P$, we get
      : *answer* := *true*.

Thus, at the end of the process, we found $bbabb \in L(G)$.

The algorithm presented solves the word problem by extracting one possible derivation tree for w. If we want to get all of them, the iteration over all production nodes reached by a certain nonterminal node has to be performed always for all possibilities. Obviously, this does not affect the worst-case complexity of the algorithm.

The complexity

During any analysis, the processing of nonterminal nodes for an inquiry consists in investigating all combinations of production nodes reachable via them. Therefore, the time complexity of the algorithm is proportional to the maximal number of production nodes (pairs of production nodes) visited in $\mathit{Graph}(CF(G), w)$. Any inquiry concerning a production node can only be initiated via the uniquely determined incoming edge. Thus, any production node (pair of production nodes) is only inquired twice if the nonterminal node (pair of nonterminal nodes) attached is inquired twice. But since we mark it by the result of its first inquiry, each further inquiry can use the entry in K. Therefore, each production node (pair of production nodes) in $\mathit{Graph}(CF(G), w)$ is itself inquired only once during an analysis.

The handling of production nodes costs a constant amount of operations to test the structure of the right-hand side of the production and to call *test_1* resp. *test_2* recursively. If we can find the answer in K, such a recursive call only costs a constant amount of time. Hence, all the other recursive calls cost proportional to the number of production nodes (pairs of production nodes) investigated furthermore. Since all of them are inquired at most once, the total complexity of the algorithm is proportional to the maximal number of pairs of production nodes in $\mathit{Graph}(CF(G), w)$. This amounts to $O((|PN| + |TN|)^2)$. Thus, we can show

Theorem 3. *Let $G = (\mathscr{K}, T, P, S) \in CCFG(2)$ in generalized Chomsky normal form and $w \in T^*$, $n := |w|$. The procedure generalized_Younger analyses w relative to G in time* $O(|P| \cdot n^6)$.

Proof. In $\mathit{Graph}(CF(G), w)$, there are $O(|CF(P)| \cdot n^3)$ production nodes. Actually, this would result in a complexity of $O(|CF(P)|^2 \cdot n^6)$. But since it is possible to have only short inputs to be analysed relative to large grammars, it is worthwhile reducing the factor $|CF(P)|^2$. This can be done as follows:

1. When constructing the graph, we sort the edges starting at a fixed nonterminal node following the production in the node reached. At the same time, an array of length $|P|$ is attached to each nonterminal node. All elements of P are enumerated by a bijective function $v : P \to [1, |P|]$ such that the array can be directly addressed by productions. For any $[X, i, j] \in NN$, the entry at position $v(X \to \alpha)$, $X \to \alpha \in CF(P)$, is a pointer to the first edge $([X, i, j], [X \to \alpha, i, k, j])$. Here, $|P|$ instead of $|CF(P)|$ entries suffice because for any $X \in \mathit{comp}(\mathscr{K})$, we know whether X is a first or a second component of a coupled nonterminal.

Obviously, these preparations can be performed during the construction of the graph without affecting the dimension of the complexity.

2. When traversing this modified derivation graph, for any $[p_1, i, k, j]$, we can find the first appropriate $[p_2, l, s, r]$ within a constant amount of time by using the pointer in the entry $v(p)$. The last appropriate one was visited when we encounter the first not appropriate one. Thus, we investigate only those partial productions in a coupled way which are really coupled. □

5.3. $G \in CCFG(l)$, $l \geqslant 1$

Graph$(CF(G), w)$ is again traversed starting at $[S, 1, n]$. The difference consists in the fact that now, each inquiry has to investigate r-tuples, $1 \leqslant r \leqslant l$. Each such r-tuple corresponds to a nonterminal $(X_1, \dots, X_r) \in \mathscr{N}[r]$ and the inquiry tests whether the components and the subtrees rooted by these components are correctly coupled. To model the l different kinds of possible inquiries, we have to deal with l distinct functions modeling one of the unary up to the l-ary inquiry. In addition, the array K has to be l-dimensional. Since the more complex case-construct whose size only depends on l, not on $|w|$, can be implemented such that the additional code costs only a constant overhead, it is considered to be constant as to the time complexity. Thus, we can show

Theorem 4. *Let $G = (\mathscr{N}, T, P, S) \in CCFG(l)$, $l \geqslant 1$, in generalized Chomsky normal form and $w \in T^*$, $n := |w|$. The procedure generalized_Younger analyses w relative to G within $O(|P| \cdot n^{3l})$ operations.*

Proof. The procedure is analogous to the case $l = 2$. In addition, we still have at most $O(|CF(P)| \cdot n^3)$ production nodes here, which also dominates the number of nonterminal nodes.

During any analysis, the processing of a tuple of nonterminal nodes for an inquiry consists in investigating all combinations of production nodes reachable via them. Therefore, the time complexity of the algorithm is proportional to the maximal number of tuples of production nodes visited in *Graph*$(CF(G), w)$. Any inquiry concerning a single production node can only be initiated via the uniquely determined incoming edge. Thus, any tuple of production nodes is only inquired twice if the tuple of nonterminal nodes attached is inquired twice. But since we mark it by the result of its first inquiry, each further inquiry can use the entry in K. Therefore, each fixed tuple of production nodes in *Graph*$(CF(G), w)$ is itself inquired only once during an analysis.

The handling of an inquiry costs a constant amount of operations to test the structure of the right-hand side of the production and to call the *test*-functions recursively. If we can find the answer in K, such a recursive call only costs a constant amount of time. Hence, all the other recursive calls cost proportional to the number of tuples of production nodes investigated furthermore. Since all of them are inquired at most once, the total complexity of the algorithm is proportional to the maximal number of different tuples of production nodes in $Graph(CF(G), w)$. This amounts to $O((|PN| + |TN|)^l)$.

In general, at most $O(|CF(P)| \cdot n^3)$ production nodes are contained in $Graph(CF(G), w)$. Besides, we use the strategy presented to prevent the algorithm from inquiring production nodes not coupled in P. Therefrom, the theorem results. □

6. Constructing the normal form for $G \in CCFG(l)$, $l \geq 1$

The steps necessary for this construction equal the context-free case with regard to their content and their order. But their procedure is much more complex because they always have to respect the dependencies between different components of coupled nonterminals and/or coupled productions.

To simplify the notation, long productions are represented as

$$\begin{pmatrix} X_1 \\ \vdots \\ X_r \end{pmatrix} \rightarrow \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_r \end{pmatrix}$$

instead of $(X_1, \ldots, X_r) \rightarrow (\alpha_1, \ldots, \alpha_r)$.

Let $G = (\mathscr{X}, T, P, S) \in CCFG(l)$. We transform G by using the following steps:

Step 1: Elimination of $t \in T$ in α_i where $X_i \rightarrow \alpha_i \in CF(P)$ and $|\alpha_i| > 1$.

Here, each such $t \in T$ is substituted by a new nonterminal $\chi_t \in \mathscr{X}[1]$ (χ_t is created once for any fixed $t \in T$), while $\chi_t \rightarrow t$ is added once to P.

```
$\mathscr{X} := \mathscr{X} \cup \{\chi_t \mid t \in T, \chi_t \notin comp(\mathscr{X})\}$;
$P := P \cup \{\chi_t \rightarrow t \mid t \in T, \chi_t \notin comp(\mathscr{X})\}$;
**while exists** $X_i \rightarrow \alpha_i = \alpha_i^{(1)} t \alpha_i^{(2)}$ where $t \in T$ and $\alpha_i^{(1)}\alpha_i^{(2)} \neq \varepsilon$ for some
                    $p := (X_1, \ldots, X_i, \ldots, X_r) \rightarrow (\alpha_1, \ldots, \alpha_i^{(1)} t \alpha_i^{(2)}, \ldots, \alpha_r) \in P$
**do**
    $$P := P \setminus \{p\} \cup \left\{ \begin{pmatrix} X_1 \\ \vdots \\ X_{i-1} \\ X_i \\ X_{i+1} \\ \vdots \\ X_r \end{pmatrix} \rightarrow \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{i-1} \\ \alpha_i^{(1)} \chi_t \alpha_i^{(2)} \\ \alpha_{i+1} \\ \vdots \\ \alpha_r \end{pmatrix} \right\};$$
**od**;
```

Step 2: Reduction of $|\alpha_i|$ where $X_i \to \alpha_i \in CF(P)$.

The *idea* is to produce several nonterminal components successively instead of simultaneously. Hereby, we have to guarantee that each of the new intermediate steps produces sequences of parentheses correctly nested. For example, we transform $(X_1,X_2,X_3) \to (A_1B_1,B_2NA_2,A_3) \in P$ where (A_1,A_2,A_3), (B_1,B_2), $N \in \mathscr{K}$ hold into the following sequence of productions: $(X_1,X_2,X_3) \to (A'_1B_1,B_2A'_2,A'_3)$, $(A'_1,A'_2,A'_3) \to (A_1,NA_2,A_3)$, where (A'_1,A'_2,A'_3) becomes a new element in $\mathscr{K}[3]$. $X \to Y_1Y_2Y_3Y_4 \in P$ for $(Y_1,Y_2,Y_3,Y_4) \in \mathscr{K}[4]$ is transformed into $X \to A_1A_2$, $(A_1,A_2) \to (Y_1Y_2,Y_3Y_4)$, where (A_1,A_2) is a new element in $\mathscr{K}[2]$. This special kind of a factorization generates the original sequence by always treating the innermost parenthesis first. The whole procedure is divided into four partial steps:

The *first* partial step replaces all $\delta \in ED(\mathscr{K},T)$ produced by one partial production by a new element in $\mathscr{K}[1]$, which itself only produces δ. This is done analogously as it was done for $t \in T$ in Step 1. In addition, we shorten the productions whose left-hand side are in $\mathscr{K}[1]$.

The *second* partial step treats subsequences NY_j resp. Y_jN, where $N \in \mathscr{K}[1]$ and $(Y_1,\dots,Y_r) \in \mathscr{K}[r]$, $1 \leqslant j \leqslant r$, hold inside partial productions. They are removed by substituting $(Y_1,\dots,Y_{j-1},NY_j,Y_{j+1},\dots,Y_r)$ resp. $(Y_1,\dots,Y_{j-1},Y_jN,Y_{j+1},\dots,Y_r)$ in the complete production by a new $(\xi_1,\dots,\xi_r) \in \mathscr{K}[r]$, which itself only generates the sequence substituted. For subsequences Y_jY_{j+1}, the analogue is done by the *third* partial step. Here, the new element introduced in $\mathscr{K}$ is of rank $(r-1)$.

To be able to guarantee for the last partial step that the new nonterminal introduced there does not transgress the rank of the grammar, we perform the first three partial steps as long as there exists any partial production of length greater than two even if this is not the one actually shortened. This is expressed before the execution of the step itself by the condition

$$(|\alpha_i| > 2 \text{ or } \exists l \neq i \text{ where } |\alpha_l| > 1).$$

Subsequent to these three steps, there remain productions showing the normal form or showing partial productions of length greater than two where the structure is of a special kind, namely

$$\alpha_i = A_{last(A)}B_{last(B)}\dots C_{last(C)}Y_jU_1V_1\dots W_1,$$

i.e., a sequence of final parenthesis components possibly followed by a single component of some arbitrary arity and finished by a sequence of first components. These partial productions are removed by substituting two parentheses at a time by one parenthesis of a higher rank which produces only these two parentheses. Here, the two parentheses have to belong to consecutive signs inside the partial production of length > 2. For example, $(X_1,X_2,X_3,X_4) \to (A_1,B_1,C_1,C_2B_2A_2) \in P$ where (A_1,A_2), (B_1,B_2), $(C_1,C_2) \in \mathscr{K}[2]$ holds is transformed into the sequence $(X_1,X_2,X_3,X_4) \to (A_1,D_1,D_2,D_3A_2)$, $(D_1,D_2,D_3) \to (B_1,C_1,C_2B_2)$ where (D_1,D_2,D_3) becomes a new element in $\mathscr{K}[3]$.

In the sequel, we often use the notion of the jth nonterminal occuring in $rhs(p)$ for any $p \in P$. This was defined as $NT_j(rhs(p))$ in Notation 1. To simplify the presentation, we additionally denote by $sp_i\left((\alpha_1,\ldots,\alpha_r),(\beta_1,\ldots,\beta_j)\right)$ ("*substitute parenthesis i in* $(\alpha_1,\ldots,\alpha_r)$ *by* $(\beta_1,\ldots,\beta_j)$") the operation which substitutes the ith nonterminal in $(\alpha_1,\ldots,\alpha_r)$, $\alpha_1\cdots\alpha_r \in ED(\mathscr{K},T)$, by an arbitrary tuple $(\beta_1,\ldots,\beta_j)$ of the same arity as $NT_i(\alpha_1,\ldots,\alpha_r)$. Now, the complete second step is partitioned into

(*a*) Removal of $\delta \in ED(\mathscr{K},T)$ inside partial productions and treatment of uncoupled productions

```
while exists $X_i \to \alpha_i = \alpha_i^{(1)}\delta\alpha_i^{(2)}$ in $CF(P)$ where $\delta \in ED(\mathscr{K},T)$ and $|\delta| \geqslant 2$
            for some $p := (X_1,\ldots,X_r) \to (\alpha_1,\ldots,\alpha_r) \in P$ and
            $(|\alpha_i| > 2$ or $\exists l \neq i$ where $|\alpha_l| > 1)$
do /* let $\xi \notin comp(\mathscr{K})$ */
    $\mathscr{K} := \mathscr{K} \cup \{\xi\}$;   /* $\xi \in \mathscr{K}[1]$ */
    $P := P \setminus \{p\} \cup \{\xi \to \delta\} \cup \left\{\begin{pmatrix} X_1 \\ \vdots \\ X_{i-1} \\ X_i \\ X_{i+1} \\ \vdots \\ X_r \end{pmatrix} \to \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{i-1} \\ \alpha_i^{(1)}\xi\alpha_i^{(2)} \\ \alpha_{i+1} \\ \vdots \\ \alpha_r \end{pmatrix}\right\}$;
    while exists $X \to \alpha \in P$ where $X \in \mathscr{K}[1]$ and $|\alpha| > 2$, $\alpha = \alpha_1\alpha_2\ldots\alpha_{|\alpha|}$
    do
        if $\alpha = \alpha^{(1)}\alpha^{(2)}$ where $\alpha^{(1)},\alpha^{(2)} \in ED(\mathscr{K},T)$
        then /* let $N_1,N_2 \notin comp(\mathscr{K})$ */
            $\mathscr{K} := \mathscr{K} \cup \{N_1,N_2\}$;
            $P := P \setminus \{p\} \cup \{X \to N_1N_2, N_1 \to \alpha^{(1)}, N_2 \to \alpha^{(2)}\}$;
        else /* let $N_1$, $N_2 \notin comp(\mathscr{K})$ */
            $\mathscr{K} := \mathscr{K} \cup \{(N_1,N_2)\}$;
            $P := P \setminus \{p\} \cup \{X \to N_1N_2,\ (N_1,N_2) \to (\alpha_1\alpha_2, \alpha_3\ldots\alpha_{|\alpha|})\}$;
        fi;
    od;
od;
```

(*b*) Removal of connected sequences Y_jN or NY_j inside partial productions for some $N \in \mathscr{K}[1]$ and $(Y_1,\ldots,Y_r) \in \mathscr{K}[r]$

```
while exists $X_i \to \alpha_i = \alpha_i^{(1)}Y_jN\alpha_i^{(2)}$ or $X_i \to \alpha_i = \alpha_i^{(1)}NY_j\alpha_i^{(2)}$ in $CF(\{p\})$
            for $(Y_1,\ldots,Y_r) \in \mathscr{K}$, $N \in \mathscr{K}[1]$ and
            $p := (X_1,\ldots,X_s) \to (\alpha_1,\ldots,\alpha_s) \in P$
            where $(|\alpha_i| > 2$ or $\exists l \neq i$ where $|\alpha_l| > 1)$
```

```
do /* let $\xi_1,\dots,\xi_r \notin comp(\mathscr{K})$ */
    $\mathscr{K} := \mathscr{K} \cup \{(\xi_1,\dots,\xi_r)\}$;
```

$$P := P \cup \left\{ \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_{j-1} \\ \xi_j \\ \xi_{j+1} \\ \vdots \\ \xi_r \end{pmatrix} \to \begin{pmatrix} Y_1 \\ \vdots \\ Y_{j-1} \\ Y_jN \\ Y_{j+1} \\ \vdots \\ Y_r \end{pmatrix} \right\}; \text{ /* or */ } P := P \cup \left\{ \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_{j-1} \\ \xi_j \\ \xi_{j+1} \\ \vdots \\ \xi_r \end{pmatrix} \to \begin{pmatrix} Y_1 \\ \vdots \\ Y_{j-1} \\ NY_j \\ Y_{j+1} \\ \vdots \\ Y_r \end{pmatrix} \right\};$$

```
    /* let $(Y_1,\dots,Y_r) = NT_a(rhs(p))$ and $N = NT_b(rhs(p))$ */
    $P := P \setminus \{p\} \cup \{(X_1,\dots,X_{i-1},X_i,X_{i+1},\dots,X_s)$
                    $\to sp_b(sp_a(rhs(p),(\xi_1,\dots,\xi_r)),\varepsilon)\}$;
od;
```

(*c*) Removal of sequences Y_jY_{j+1} inside partial productions for some $(Y_1,\dots,Y_s) \in \mathscr{K}$

```
while exists $X_i \to \alpha_i = \alpha_i^{(1)}Y_jY_{j+1}\alpha_i^{(2)}$ in $CF(\{p\})$ for some $(Y_1,\dots,Y_r) \in \mathscr{K}$
        and $p := (X_1,\dots,X_s) \to (\alpha_1,\dots,\alpha_s) \in P$ where $(|\alpha_i| > 2$ or $\exists l \neq i$
        where $|\alpha_l| > 1)$
do /* let $\xi_1,\dots,\xi_{r-1} \notin comp(\mathscr{K})$ */
    $\mathscr{K} := \mathscr{K} \cup \{(\xi_1,\dots,\xi_{r-1})\}$;
```

$$P := P \setminus \{p\} \cup \left\{ \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_{j-1} \\ \xi_j \\ \xi_{j+1} \\ \vdots \\ \xi_{r-1} \end{pmatrix} \to \begin{pmatrix} Y_1 \\ \vdots \\ Y_{j-1} \\ Y_jY_{j+1} \\ Y_{j+2} \\ \vdots \\ Y_r \end{pmatrix} \right\};$$

```
    /* let $(Y_1,\dots,Y_r) = NT_a(rhs(p))$ */
    $P := P \cup \{(X_1,\dots,X_{i-1},X_i,X_{i+1},\dots,X_s)$
            $\to sp_a(rhs(p),\ (\xi_1,\dots,\xi_{j-1},\xi_j,\varepsilon,\xi_{j+1},\xi_{j+2},\dots,\xi_{r-1}))\}$;
od;
```

Now, in any arbitrary $p := (X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$, there remain only those α_i showing $|\alpha_i| > 2$ which additionally have the structure

$$\alpha_i = A_{last(A)}B_{last(B)}\dots C_{last(C)}Y_jU_1V_1\dots W_1$$

where $(A_1,\dots,A_{last(A)}),\dots,(W_1,\dots,W_{last(W)}) \in \mathscr{K}$. If $|\alpha_i| > 2$ holds for some i, all the other α_j fulfilling $|\alpha_j| = 2$ show this structure. All the other partial productions only generate one single symbol. Partial productions α_i of this special structure can be removed by iterating the following:

(d) Let α_i, $|\alpha_i| > 2$, the partial production in p of the above structure where $s := last(A)$ is minimal. U_1 up to W_1 in α_i can be treated symmetrically.

```
if $\alpha_i = A_s Y_j U_1 \ldots W_1$
then
    /* let $(A_1,\ldots,A_s) = NT_a(rhs(p))$ and $(Y_1,\ldots,Y_l) = NT_b(rhs(p))$ */
    if exists $\alpha_q = \alpha_q^{(1)} Y_{j-1} A_1$ in $CF(\{p\})$ for these nonterminals
    then /* $\xi_1,\ldots,\xi_{l+s-2} \notin comp(\mathscr{K})$ */
        $\mathscr{K} := \mathscr{K} \cup \{(\xi_1,\ldots,\xi_{l+s-2})\}$;
        $P := P \cup \left\{ \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_{j-2} \\ \xi_{j-1} \\ \xi_j \\ \vdots \\ \xi_{j+s-3} \\ \xi_{j+s-2} \\ \xi_{j+s-1} \\ \vdots \\ \xi_{l+s-2} \end{pmatrix} \to \begin{pmatrix} Y_1 \\ \vdots \\ Y_{j-2} \\ Y_{j-1}A_1 \\ A_2 \\ \vdots \\ A_{s-1} \\ A_s Y_j \\ Y_{j+1} \\ \vdots \\ Y_l \end{pmatrix} \right\}$;
        $P := P \setminus \{p\} \cup \{(X_1,\ldots,X_r)$
            $\to sp_b(sp_a(rhs(p),\ (\varepsilon, \xi_j, \xi_{j+1}, \ldots, \xi_{j+s-3}, \xi_{j+s-2}, \varepsilon)),$
                $(\xi_1, \ldots, \xi_{j-1}, \xi_{j+s-2}, \ldots, \xi_{l+s-2}))\}$;
    else /* $A_1$ and $Y_{j-1}$ are in distinct $\alpha_q$'s */
        /* analogue procedure with $(\xi_1,\ldots,\xi_{l+s-1})$ */
    fi;
else /* $\alpha_i = A_s B_{last(B)} \alpha_i'$ */
    /* identical procedure using $B$ instead of $Y$ and $last(B)$ instead of $l$ */
fi;
```

Altogether, the new element $(\xi_1,\ldots,\xi_{last(\xi)})$ does not transgress the rank of the grammar because all the transformations described above maintain $last(\xi) \leqslant r$. For step (d), this is true since before performing it, the other steps ensure that all partial productions involved and not showing the special structure only produce one single symbol. Therefore, all components of the nonterminals $(A_1,\ldots,A_s)$ and $(Y_1,\ldots,Y_l)$ substituted by this step are distributed on different α_i's except the ones substituted in connection. Therefore, $l + s - 2$ resp. $l + s - 1$ cannot be greater than r. In addition, the new productions $(X_1,\ldots,X_r) \to (\alpha_1',\ldots,\alpha_r')$ obey the conditions $|\alpha_i'| = |\alpha_i| - 1$ and $|\alpha_j'| \leqslant |\alpha_j|$ for $j \neq i$, while the productions for $(\xi_1,\ldots,\xi_{last(\xi)})$ are always in normal form.

Step 3: Removal of partial productions $X_i \to \varepsilon$ in $(X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$

(a) To each $(X_1,\dots,X_r) \in \mathscr{K}[r]$, we attach 2^r new elements in $\mathscr{K}$ each one encoding one fixed combination of partial productions producing ε. Since there are only 2^r possibilities which combination of partial productions for $(X_1,\dots,X_r)$ can simultaneously generate ε, we can define a one-to-one mapping between the new nonterminals and the combinations of positions producing ε. If ε is produced at $i < r$ positions, the corresponding new nonterminal is of the arity $(r-i)$. Only the nonterminal for $(X_1,\dots,X_r) \to (\varepsilon,\dots,\varepsilon)$ gets the arity 1 instead of 0.

For any $(X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$, we generate all variants resulting if we substitute one or several nonterminals in $(\alpha_1,\dots,\alpha_r)$ by the new nonterminals just attached. Hereby, we substitute those components by ε which represent ε in the onto-to-one image. If we investigate the case of a complete ε-production, all but the first component are substituted by ε. The new P consists in the old P united with all these variants.

```
(b) while exists $p = (X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P$ where
            $\exists 1 \leqslant j_1 < j_2 < \dots < j_s \leqslant r : \alpha_{j_1} = \dots = \alpha_{j_s} = \varepsilon$
    do /* let $N^{E(j_1,\dots,j_s)}_{(X_1,\dots,X_r)}$ the new nonterminal attached to $(X_1,\dots,X_r)$ whose
       one-to-one image produces $\varepsilon$ exactly in the components $j_1 < j_2 < \dots < j_s$ */
       if $s = r$ and $(N^{E(1,\dots,r)}_{(X_1,\dots,X_r)} \to \varepsilon) \notin P$
       then
           for all $(Y_1,\dots,Y_l) \to (\gamma_1,\dots,\gamma_l) \in P$
           do
               for $k := 1$ to $s$
               do
                   if $\gamma_k = N^{E(1,\dots,r)}_{(X_1,\dots,X_r)}\gamma'_k$ or $\gamma_k = \gamma'_k N^{E(1,\dots,r)}_{(X_1,\dots,X_r)}$
                   then
                       $\gamma_k := \gamma'_k$
                   fi;
               od;
           od;
           $P := P \setminus \{p\} \cup \{N^{E(1,\dots,r)}_{(X_1,\dots,X_r)} \to \varepsilon\}$;
       else
           $P := P \setminus \{p\} \cup \{N^{E(j_1,\dots,j_s)}_{(X_1,\dots,X_r)} \to (\alpha_1,\dots,\alpha_{j_1-1},\alpha_{j_1+1},\dots,\alpha_{j_s-1},\alpha_{j_s+1},\dots,\alpha_r)\}$;
       fi;
    od;
    if $N^{E(1)}_S \to \varepsilon \in P$
    then
        $\mathscr{K} := \mathscr{K} \cup \{S'\}$; $P := P \cup \{S' \to \alpha \mid S \to \alpha \in P\} \cup \{S \to \varepsilon\}$;
        substitute $S$ by $S'$ in the right-hand side of all partial productions;
    fi;
```

(c) Reduce G by removing all elements from $\mathscr{K}$ which produce nothing and, at the same time, all the $p \in P$ showing one of these elements in $rhs(p)$. Additionally, remove all productions $N^{E(1,\dots,r)}_{(X_1,\dots,X_r)} \to \varepsilon$ for $(X_1,\dots,X_r) \in \mathscr{K}$ since they are unreachable now.

Step 4: Removal of all pure renaming productions

```
for all $(X_1,\dots,X_r) \in \mathscr{K}$
do
    $U[(X_1,\dots,X_r)] := \{(Y_1,\dots,Y_r) \in \mathscr{K}[r] \mid (X_1,\dots,X_r) \overset{*}{\Rightarrow}_G (Y_1,\dots,Y_r)\}$
od;
$P := P \setminus \{(X_1,\dots,X_r) \to (Y_1,\dots,Y_r) \mid (X_1,\dots,X_r),(Y_1,\dots,Y_r) \in \mathscr{K}[r]\}$;
for all $(X_1,\dots,X_r) \in \mathscr{K}$
do
    while $U[(X_1,\dots,X_r)] \neq \emptyset$
    do /* let $(Y_1,\dots,Y_r) \in U[(X_1,\dots,X_r)]$ fixed */
        $P := P \cup \{(X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \mid (Y_1,\dots,Y_r) \to (\alpha_1,\dots,\alpha_r) \in P\}$;
        $U[(X_1,\dots,X_r)] := U[(X_1,\dots,X_r)] \setminus \{(Y_1,\dots,Y_r)\}$;
    od;
od;
```

Resulting from these steps, we get

Theorem 5. *For all $G = (\mathscr{K},T,P,S) \in CCFG(l)$, $l \geqslant 1$, there exists $G' \in CCFG(l)$ in generalized Chomsky normal form fulfilling $L(G) = L(G')$. G' can be determined in time $O(m_G \cdot 4^l \cdot |\mathscr{K}| \cdot |P|)$ increasing $|P|$ to $O(m_G \cdot 4^l \cdot |\mathscr{K}| \cdot |P|)$ where $m_G := \max\{|\alpha_1 \cdot \ldots \cdot \alpha_r| \mid (X_1,\dots,X_r) \to (\alpha_1,\dots,\alpha_r) \in P\}$.*

Proof. The existence of G' directly follows from the construction above. Step 1 costs $O(|P| \cdot m_G)$ operations increasing the sets $\mathscr{K}$ and P at most by $|T|$ elements each. Step 2 also costs $O(|P| \cdot m_G)$ operations if it is implemented such that each production is shortened completely before going on to the next one. Here, $|\mathscr{K}|$ and $|P|$ can grow up to $O(|P| \cdot m_G)$. For step 3, we have $m_G \leqslant 2l$. Therefore, it cannot produce more than $O(4^l \cdot |P|)$ new productions and $O(2^l \cdot |\mathscr{K}|)$ new elements in $\mathscr{K}$. Its complexity is proportional to the number of productions generated. This is true for the last step, too, but it can generate $O(|\mathscr{K}| \cdot |P|)$ new productions. Altogether, the grammar constructed here can be of the size $O(m_G \cdot 4^l \cdot |\mathscr{K}| \cdot |P|)$. □

7. More efficient subclasses

In order to describe subclasses of *CCFG* which can be analysed faster, we reduce the set of production nodes in *Graph*($CF(G), w$) in two different ways. Then, the algorithm can be estimated more restrictively. The first idea is to start with the notion of context-free direct ambiguity.

Definition 9 (*Context-free direct ambiguity*). Let $G = (N, T, P, S)$ a context-free grammar, $p : X \to X_1 \ldots X_k \in P$, and $w \in T^*$. $da(w, p, G)$, the *degree of direct ambiguity of w relative to p in G*, is defined as the number of distinct decompositions $w^{(1)}, \ldots, w^{(k)}$ of w, i.e. $w = w^{(1)} \ldots w^{(k)}$, where $X_i \overset{*}{\Rightarrow}_G w^{(i)}$ holds. $da(G)$, the *degree of direct ambiguity of G*, is defined as the minimal k fulfilling

$$\forall p \in P \, \forall w \in T^* : da(w, p, G) \leqslant k,$$

if such a k exists. Otherwise, $da(G) := \infty$.[2]

In [7], this notion was transferred onto *CCFG* as

Definition 10 (*Direct ambiguity in CCFG*). Let $G \in CCFG$. $da(G)$, the *degree of direct ambiguity of G*, is defined as $da(G) := da(CF(G))$.

We only look at the case $da(G) = 1$ because even in the context-free case it is open whether there exist languages whose degree of direct ambiguity in the sense of [23] is neither 1 nor ∞.

Let $G = (\mathscr{K}, T, P, S) \in CCFG$ in generalized Chomsky normal form fulfilling $da(G) = 1$ and let $p := X \to AB \in CF(P)$ fixed. If $[X, i, j]$ is a node in $Graph(CF(G), w)$, it follows from the definition of direct ambiguity that there exists only one k in between 0 and j for which there exists a node $[p, i, k, j]$. This equals the fact that $Graph(CF(G), w)$ contains $O(|CF(P)| \cdot |w|^2)$ nodes. It follows

Theorem 6. *Let $G = (\mathscr{K}, T, P, S) \in CCFG(l)$, $l \geqslant 1$, in generalized Chomsky normal form fulfilling $da(G) = 1$ and $w \in T^*$, $n := |w|$. The procedure generalized_Younger analyses w relative to G in time $O(|CF(P)| \cdot n^3 + |P| \cdot n^{2l})$.*

The second notion used to reduce the set of production nodes in the derivation graph characterizes *CCFG*'s where each element of P has at most i components producing two elements of $comp(\mathscr{K})$ while all the others are only of length 1. It can be formalized as

Definition 11 (*i-limited*). $G = (\mathscr{K}, T, P, S) \in CCFG(l)$, $l \geqslant 1$, in generalized Chomsky normal form is *i-limited* for some $1 \leqslant i < l$, if and only if for all $(X_1, \ldots, X_r) \to (\alpha_1, \ldots, \alpha_r) \in P$, it holds

$$|\{j \mid |\alpha_j| = 2, 1 \leqslant j \leqslant r\}| \leqslant i.$$

Theorem 7. *Let $G = (\mathscr{K}, T, P, S) \in CCFG(l)$, $l \geqslant 1$, i-limited for an $i \geqslant 1$. The generalized Younger algorithm solves the word problem for any $w \in T^*$, $n := |w|$, relative to G in time $O\left(|P| \cdot n^{2l+i}\right)$.*

[2] In [23], this notion was defined first and only for $p : S \to X_1 \ldots X_k \in P$ and $w \in L(G)$. Thus, our $da(G)$ can be larger than $da(G)$ in the sense of [23]. If we denote by $da_{Sha}(G)$ the [23]-version of the above notion, we can state $da(G) = \max\{da_{Sha}(N, T, P, X) \mid X \in N\}$.

Proof. Each tuple of production nodes inquired contains only nodes $[p_j, s, k, t]$ where $1 \leqslant j \leqslant r$ for a fixed and r-ary $p \in P$. Because of the definition of i-limited, at most i of them fulfill $k \neq 0$. Thus, for any r-ary production, this tuple has at most $3i + 2(r-i) = 2r + i$ variable indices, which range from 0 to n. □

The notion i-limited is especially important because in [18], the following theorem is shown by a construction increasing the grammar size at most by the factor l:

Theorem 8. *For any $G \in CCFG(l)$, $l \geqslant 2$, there exists a 2-limited $G' \in CCFG(l)$ in generalized Chomsky normal form fulfilling $L(G) = L(G')$.*

Corollary 1. *Let $G = (\mathscr{K}, T, P, S) \in CCFG(l)$, $l \geqslant 2$, in generalized Chomsky normal form. The procedure generalized_Younger solves the word problem for any $w \in T^*$, $n := |w|$, relative to G in time $O\left(|P| \cdot n^{2l+2}\right)$.*

8. Conclusions

In this paper, we generalized the normal form of Chomsky for context-free grammars and the algorithm of Younger onto the class of all Coupled-Context-Free Grammars. This results in the first algorithm solving the problem of analysing such grammars of arbitrary rank. The time complexity of our algorithm amounts to $O(|CF(P)| \cdot n^3 + |P| \cdot n^{3l})$, where n denotes the length of the input word and l the rank of the grammar. It is of special importance that for any rank, the time complexity depends only linearly on the size of the grammar. In contrast to our algorithm, all known parsing procedures for TAGs (i.e., [10, 22, 25]) and Coupled-Context-Free Grammars of rank 2 (i.e., [7]), respectively, depend at least quadratically on the size of the grammar as to their time complexity.

References

[1] A. Abbeillé, Parsing French with tree adjoining grammars: Some linguistic accounts, in: *Proc. 12th Internat. Conf. on Computational Linguistics* (1988) 7–12.

[2] J. Bresnan, R. Kaplan, P. Peters and A. Zaenen, Cross-serial dependencies in Dutch, *Linguistic Inquiry* **13** (1982) 613–635.

[3] N. Chomsky, On Certain Formal Properties of Grammars, *Inform. and Control* **2** (1959) 137–167.

[4] N. Chomsky and N. Schützenberger, The algebraic theory of context–free languages, in: *Computer Programming and Formal Systems* (North Holland, Amsterdam, 1963) 118–161.

[5] J. Dassow and G. Păun, *Regulated Rewriting in Formal Language Theory* (Springer, Berlin 1989).

[6] S.A. Greibach and J.E. Hopcroft, Scattered Context Grammars, *J. Comput. System Sci.* **3** (1969) 232–247.

[7] Y. Guan, Klammergrammatiken, Netzgrammatiken und Interpretationen von Netzen., Ph.D. Thesis, University of Saarbrücken, 1992.

[8] Y. Guan and G. Hotz, TAGs as a semantic of trees, in: *Proc. 1th Internnat. Workshop on Tree Adjoining Grammars* (1990) 34-37.

[9] Y. Guan, G. Hotz and A. Reichert, Tree grammars with multilinear interpretation, Technical Report, University of Saarbrücken, 1992.

[10] K. Harbusch, An efficient parsing algorithm for tree adjoining grammars, in: *Proc. 28th Ann. Meeting of the Ass. for Computational Linguistics* (1990).
[11] J. Higginbotham, English is not a Context-Free Language, *Linguistic Inquiry* **15** (1984) 225–235.
[12] G. Hotz, Erzeugung formaler Sprachen durch gekoppelte Ersetzungen, 4. Colloquium Automatentheorie, München (1967) 62–73.
[13] A.K. Joshi, Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions, in: D. Dowty, L. Karttunen, and A. Zwicky, eds., *Natural Language Parsing – Psycholinguistic, Computational, and Theoretical Perspectives* (Cambridge University Press, Cambridge, 1985) 206–250.
[14] A.K. Joshi, An introduction to tree adjoining grammars, in: A. Manaster-Ramer, ed., *Mathematics of Language* (Benjamin, New York, 1987).
[15] A.K. Joshi, L.S. Levy and M. Takahashi, Tree Adjunct Grammars, *J. Comput. System Sci.* **10** (1975) 136–163.
[16] T. Kroch, Unbounded Dependencies and Subjacency in a Tree Adjoining Grammar, in: A. Manaster-Ramer, ed., *Mathematics of Language* (Benjamins, New York, 1987) 143–172.
[17] T. Kroch and A.K. Joshi, The linguistic relevance of tree adjoining grammars, Technical Report MS-CIS-85-16, University of Pennsylvania, Philadelphia, 1985.
[18] G. Pitsch, Analyse von Klammergrammatiken, Ph.D. Thesis, University of Saarbrücken, 1993.
[19] G. Pitsch, $LR(k)$-parsing of coupled-context-free grammars, in: *Proc. 15th Internat. Conf. on Computational Linguistics* (1994) 401–405.
[20] G. Pitsch, $LL(k)$-parsing of coupled-context-free grammars, *Comput. Intell.* **10** (1994) 563–578.
[21] D. Radzinski, Chinese Number-Names, Tree adjoining grammars and mild context-sensitivity, *Computational Linguistics* **17** (1991) 277–299
[22] Y. Schabes and A.K. Joshi, An early-type parsing algorithm for tree adjoining grammars, in: *Proc. 26th Annual Meeting of the Ass. for Computational Linguistics* (1988).
[23] E. Shamir, Some inherently ambiguous context-free languages, *Inform. and Control* **18** (1971) 355–363.
[24] S. Shieber, Evidence against context–freeness of natural language, *Linguistics and Philosophy* **8** (1986) 333–343.
[25] K. Vijay-Shanker and A. Joshi, Some computational properties of tree adjoining grammars, in: *Proc. 23th Annual Meeting of the Ass. for Computational Linguistics* (1985) 82–93.
[26] D. Younger, Recognition and parsing of context-free languages in time n^3, *Inform. and Control* **10** (1967) 189–208.