$LR(k)$–Parsing of Coupled-Context-Free Grammars*

Gisela Pitsch
FB Informatik, Universität des Saarlandes
D-66123 Saarbrücken, Germany E-mail: pitsch@cs.uni-sb.de

Abstract

Coupled-Context-Free Grammars are a generalization of context-free grammars obtained by combining nonterminals to parentheses which can only be substituted simultaneously. Referring 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. Here, we present a generalization of the context-free $LR(k)$-notion, which characterizes subclasses of Coupled-Context-Free Grammars – and therefore for TAGs – which can be parsed in linear time. The parsing procedure described works incrementally so that it can be used for on-line parsing of natural language. Examples show that important Tree Adjoining Languages, e.g. those modelling cross-serial dependencies, can be generated by $LR(k)$–Coupled-Context-Free Grammars.

1 Introduction

In order to process natural languages, we first have to model the syntax formally. Many investigations as, e.g., [Hig84] show that this cannot be done by context-free grammars (CFGs). For context-sensitive grammars which are powerful enough, it is known that the analysis is 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 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 CFGs. Their suitability to model syntactical phenomena follows from the fact that they include the languages generated by the Tree Adjoining Grammars (TAGs) of [Jos87] as one subclass. 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. [Shi86]).

The formalism of Coupled-Context-Free Grammars has been introduced in [GHR92] and [Gua92]. It belongs to the family of regulated string rewriting systems investigated in [DP89]. The increased generative capacity is obtained by allowing to rewrite simultaneously a certain number of elements. Other regulated string rewriting systems as, e.g., the Scattered Context Grammars of [GH69] generalize CFGs by allowing simultaneous rewriting of arbitrary combinations of elements. In [DP89], 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 (cf. [Jos85]). In contrast to these, all languages defined by our formalism are semilinear because of two restrictions. First, only those elements can be rewritten simultaneously which were produced by the same rewriting. 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 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 grows with the rank. The smallest element of the hierarchy – the one of rank 1 – are CFGs. The next element, namely Coupled-Context-Free Grammars of rank 2, generates the same class of languages as the Tree Adjoining Grammars of [JLT75] and [Jos87]. Hence, all notions and algorithms designed for Coupled-Context-Free Grammars of rank 2 can easily be translated onto TAGs (cf. [Gua92]).

Because of the enlarged generative capacity, it is not surprising that the complexity of analysing languages generated by Coupled-Context-Free Grammars is larger than it is in the context-free case. It even increases with growing rank (cf. [HP94]). Therefore, we aim to characterize subclasses of the set of all languages generated by Coupled-Context-Free Grammars which are powerful enough to model the important phenomena of natural languages, but which are of a lower complexity.

The deterministic context-free parsing with $LR(k)$-grammars leads to a linear time analysis (cf. [Knu65]), the best possible. Therefore, its generalization is very attractive. A first attempt in this direction was done in [SV90]. But there, only TAGs are investigated. Here, we investigate the whole hierarchy of Coupled-Context-Free Grammars. Although their enlarged generative capacity seems to be contradictory to a linear time complexity of the parsing algorithm, we can present an $LR(k)$-notion for Coupled-Context-Free Grammars describing a class of languages, which can actually be analysed in linear time. This increase in power as to the linear-time analysis is paid by an expensive preprocessing. It is taking into account the complex relations between parentheses that involves the increase in complexity. However, these costs are to be paid only once for each grammar. The subclass described by our $LR(k)$-notion for a fixed k grows with the rank.

The algorithm of [SV90] for $LR(k)$-TAGs does not fulfill the important Valid Prefix Property. This means that for any prefix of the input already accepted, there exists a suffix such that the whole word is in the language analysed. It allows to detect illegal inputs as soon as possible, which is necessary for efficient parsing. Our algorithm fulfills this property. Additionally, the algorithm as well as the notion defined here represent generalizations of their context-free counterparts which are natural in the sense that they strictly contain the context-free situation as the special case of Coupled-Context-Free Grammars of rank 1.

*This research has been supported by a Graduiertenkolleg-fellowship of the Deutsche Forschungsgemeinschaft.

An example of an important $LR(k)$-Coupled-Context-Free Grammar is the one generating the language $\{w\$w \mid w \in \{a,b\}^*\}$ which reflects the syntactical construction of cross-serial dependencies.

The paper starts by defining the Coupled-Context-Free Grammars. Then, we shortly recall the context-free LR-parsing procedure. Subsequently, the deterministic finite automaton used there to guide the analysis is modified such that it can handle Coupled-Context-Free Grammars. Based on it, the parsing algorithm for $LR(0)$-Coupled-Context-Free Grammars is derived. This results in the generalized definition of the $LR(0)$-notion. As for CFGs, the $LR(k)$-Coupled-Context-Free Grammars result from the $LR(0)$-ones by resolving decision conflicts using a lookahead of at most k symbols.

2 Coupled-Context-Free Grammars

Coupled-Context-Free Grammars are defined over 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. To extend the family of context-free languages by using them we consider parentheses of arbitrary finite order defined as follows:

Definition 1 (Parentheses Set)
A finite set $\mathcal{K} := \{(k_{i,1}, \ldots, k_{i,m_i}) \mid i, m_i \in \mathbb{N}\}$ *is a* Parentheses Set *iff it satisfies* $k_{i,j} \neq k_{l,m}$ *for* $i \neq l$ *or* $j \neq m$. *The elements of* $\mathcal{K}$ *are called* Parentheses. *All parentheses of a fixed length* r *are summarized as*

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

where $\mathcal{K}[0] := \{\varepsilon\}$. *($\varepsilon$ denotes the empty word.) The set of all (first) components of parenthesis in* $\mathcal{K}$ *is denoted by*

$$comp(\mathcal{K}) := \{k_i \mid (k_1, \ldots, k_i, \ldots, k_r) \in \mathcal{K}\} \text{ resp.}$$

$$comp_1(\mathcal{K}) := \{k_1 \mid (k_1, \ldots, k_r) \in \mathcal{K}\}.$$

Straightforward from this, we get

Definition 2 (Extended Semi-Dyck Set)
Let $\mathcal{K}$ *be a parentheses set and* T *an arbitrary set where* $T \cap \mathcal{K} = T \cap comp(\mathcal{K}) = \emptyset$. $ED(\mathcal{K}, T)$, *the* extended semi-Dyck set over $\mathcal{K}$ and T, *is inductively defined by*

- $(E1)$ $T^* \subseteq ED(\mathcal{K}, T)$.
- $(E2)$ $\mathcal{K}[1] \subseteq ED(\mathcal{K}, T)$.
- $(E3)$ $u_1, \ldots, u_r \in ED(\mathcal{K}, T), (k_1, \ldots, k_{r+1}) \in \mathcal{K}[r+1]$ $\Longrightarrow \quad k_1 u_1 \cdots k_r u_r k_{r+1} \in ED(\mathcal{K}, T)$.
- $(E4)$ $u, v \in ED(\mathcal{K}, T) \Longrightarrow u \cdot v \in ED(\mathcal{K}, T)$.
- $(E5)$ $ED(\mathcal{K}, T)$ *is the smallest set fulfilling conditions* $(E1)$-$(E4)$.

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

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

$$\{(k_1, \ldots, k_r) \to (\alpha_1, \ldots, \alpha_r) \mid (k_1, \ldots, k_r) \in \mathcal{K}, \alpha_1 \cdot \ldots \cdot \alpha_r \in ED(\mathcal{K}, T)\}.$$

The left and the right side of $p := (X_1, \ldots, X_r) \to (\alpha_1, \ldots, \alpha_r) \in P$ *is denoted by*

- $\mathcal{S}(p) := (X_1, \ldots, X_r)$, *the* source *of* p, *and*
- $\mathcal{D}(p) := (\alpha_1, \ldots, \alpha_r)$, *the* drain *of* p.

Now, we can define our grammars. The term "coupled" expresses that a certain number of context-free rewritings is executed in parallel and controlled by $\mathcal{K}$.

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

At last, we give the definition of derivation in $CCFG$. Let $G = (\mathcal{K}, T, P, S) \in CCFG$ and $V := comp(\mathcal{K}) \cup T$. We define the relation $\Rightarrow_G$ as a subset of $V^* \times V^*$ consisting of all *derivation steps of rank* r for G with $r \geq 1$. $\varphi \Rightarrow_G \psi$ holds for $\varphi, \psi \in V^*$ if and only if there exist $(k_1, \ldots, k_r) \to (\alpha_1, \ldots, \alpha_r) \in P$, $u_1, u_{r+1} \in V^*$, and $u_2, \ldots, u_r \in ED(\mathcal{K}, T)$ such that

$$\varphi = u_1 k_1 u_2 k_2 \cdots u_r k_r u_{r+1} \quad \text{and}$$

$$\psi = u_1 \alpha_1 u_2 \alpha_2 \cdots u_r \alpha_r u_{r+1}.$$

$\overset{*}{\Rightarrow}_G$ denotes the reflexive, transitive closure of $\Rightarrow_G$. Obviously, $u_1 \cdot u_{r+1} \in ED(\mathcal{K}, T)$ follows from $S \overset{*}{\Rightarrow}_G \varphi$ for φ and ψ since the result of the substitution is a sequence of parentheses correctly nested if and only if the original word was. The language generated by G is defined as

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

A sequence $\varphi_1, \ldots, \varphi_n$ with $\varphi_i \Rightarrow_G \varphi_{i+1}$ for all $1 \leq i < n$ and $\varphi_1 = \varphi$, $\varphi_n = \psi$ is called a *derivation of* ψ *from* φ *in* G. A derivation is *rightmost* if and only if in each derivation step, the parenthesis ending at the rightmost point is substituted. In analogy to CFGs, it is obvious that for any derivation in $CCFG$ there exists exactly one rightmost derivation.

Example 1 $G = (\{S, (X, \overline{X})\}, \{a, b, c, d\}, P, S)$ *is in* $CCFG(2)$ *where* $P := \{S \to X\$\overline{X}, (X, \overline{X}) \to (aXb, c\overline{X}d) \mid (ab, cd)\}$. G *generates the language* $\{a^n b^n c^n d^n \mid n \geq 1\}$, *e.g.* $S \Rightarrow_G X\$\overline{X} \Rightarrow_G aXb\$c\overline{X}d \Rightarrow_G aaXbb\$cc\overline{X}dd$ $\Rightarrow_G aaabbbcccddd$

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

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

The following theorem proven in [Gua92] shows that $CCFG$ builds up an infinite hierarchy of languages and, at the same time, represents a proper extension of CFGs 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 all languages generated by TAGs and* $CCFL(l)$ *the one generated by* $CCFG(l)$. *It holds:*

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

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

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

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

3 Context-Free LR-Parsing

Now, we shortly recall the deterministic context-free $LR(k)$-parsing strategy of Knuth (cf. [Knu65]). For simplicity, we restrict ourselves on the case $k = 0$. The strategy essentially remains unchanged if lookahead is necessary. It uses a deterministic finite automaton (*dfa*) to drive a pushdown stack while scanning the input from left to right. Thus, it constructs a rightmost derivation bottom-up. The states of the *dfa* for a given $LR(0)$-CFG consist of subsets of the set of all context-free items for $G = (N, T, P, S)$, i.e. of the set $\{[X \to \alpha.\beta] \mid X \to \alpha\beta \in P\}$. They result from determining the deterministic version of the following nondeterministic automaton for G:

- Each context-free item is a state.
- There are three kinds of state transitions:
  - $[X \to \alpha.Y\beta] \stackrel{Y\in N}{\longrightarrow} [X \to \alpha Y.\beta]$,
  - $[X \to \alpha.a\beta] \stackrel{a\in T}{\longrightarrow} [X \to \alpha a.\beta]$, and
  - $[Y \to \gamma.X\delta] \stackrel{\varepsilon}{\longrightarrow} [X \to .\alpha]$.

In the deterministic version, all those context-free items are grouped in one state which can be reached from the initial state by the same sequence of symbols, with any possible number of ε-transitions in-between.

The stack symbols are the states of the *dfa*. At first, the state containing the item $[S' \to .S]$ is pushed. (The additional production $S' \to S$ serves to define exactly the start and the end of the analysis.) Then, we iterate the following actions depending on the topmost state q:

(Shift) If q contains $[X \to \alpha.a\beta]$ and a is the next input symbol to be read, we push the state reached from q via a. (It contains at least $[X \to \alpha a.\beta]$.)

(Reduce) If q contains $[X \to \alpha.]$, we pop the $|\alpha|$ topmost states. Let q' be the state now on top of the stack. Then, we push the state reached via X from q'. (q' contains at least one item $[Y \to \gamma.X\delta]$ and $[X \to .\alpha]$ while the new topmost state contains $[Y \to \gamma X.\delta]$.)

The pushdown is driven deterministically by the *dfa* if this *dfa* contains no state where there are two different Reduce-items (Reduce-Reduce conflict) or as well a Shift- as a Reduce-item (Shift-Reduce conflict). A CFG is $LR(0)$ iff the states of its *dfa* show no Shift-Reduce and no Reduce-Reduce conflict. For $LR(k)$-grammars, conflicts in the $LR(0)$-*dfa* are solved by a lookahead of k symbols.

4 The Finite Automaton

One possibility to generalize *dfa* is to construct the usual *dfa* for $CF(G)$, $G \in CCFG$. In principle, this idea is used in [SV90]. The following example shows that this produces unnecessary conflicts: Let $G = (\{S, (X, \overline{X}), D\}, \{a, b, c, d\}, P, S) \in CCFG(2)$ for $P := \{S \to X\overline{X}D\$, D \to Dd \mid d, (X, \overline{X}) \to (b, c) \mid (ab, cd)\}$ and $L(G) = \{bcd^n\$, abcdd^n\$ \mid n \geq 1\}$. Its *dfa* is shown in Figure 1. G is *not* $LR(0)$ in this way since this *dfa* obviously has a Shift-Reduce conflict (in the box doubly lined). This conflict cannot be solved by lookahead since at this point, the lookahead is always d^k. Therefore, G is not $LR(k)$ for any $k \geq 0$. But this conflict is not necessary. E.g., when analysing $bcdd$ bottom-up, we first have to reduce $X \to b$. This implies that before coming to the conflict state, we have to choose $\overline{X} \to c$ in order to get a correct derivation. This is the case because X and $\overline{X}$ resulting from applying the production $S \to X\overline{X}D\$$ are coupled and therefore have to be substituted by coupled productions.

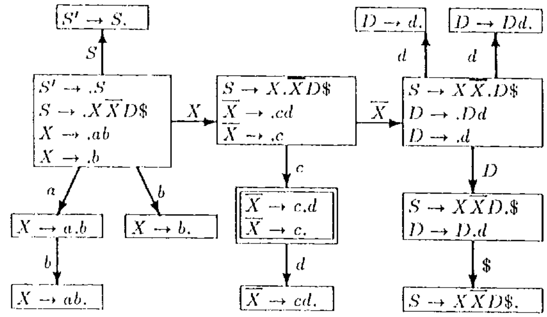

Figure 1: $dfa(G)$

To avoid these conflicts, we extend the *dfa*. If we use the context-free LR-parsing strategy, we know which production we have to choose for any $X_i \in comp(\mathcal{K}) \setminus comp_1(\mathcal{K})$ because we first encounter and reduce the corresponding $X_1 \in comp_1(\mathcal{K})$. Suppose that we can store the information about $X_2, \ldots, X_r$, $(X_1, \ldots, X_r) \in \mathcal{K}[r]$, when X_1 is reduced, let us say as the "future". (How to do this is shown in Section 5.) Can we use this to avoid the conflict? Now, our automaton needs additional transitions under such $p_i \in CF(P)$ where $\mathcal{S}(p_i) \notin comp_1(\mathcal{K})$ holds. Thus, we split ways inside the *dfa* which lead to conflict states. To formalize our automaton, we need the following

Definition 6 (1-Closure)
For all $X \in comp_1(\mathcal{K})$, let $reachable(X) :=$

$$\{Y \in comp_1(\mathcal{K}) \mid \exists X \to Y\alpha \in CF(P)\}.$$

$reachable^*(X)$ denotes its reflexive transitive closure. For any $q \in \wp(\{[X \to \alpha.\beta] \mid X \to \alpha\beta \in CF(P)\})$, we define the 1-Closure(q) as q united to the set

$$\{[X \to .\alpha] \mid \begin{array}{l} X \in comp_1(\mathcal{K}), X \to \alpha \in CF(P) \text{ and} \\ \exists Y \in comp_1(\mathcal{K}) : (\exists[Z \to \beta.Y\gamma] \in q \\ \text{and } X \in reachable^*(Y))\}. \end{array}$$

1-*Closure* formalizes the construction of the deterministic version of a nondeterministic finite automaton as it is done for the *dfa* of CFGs. Its special feature is that it uses only those $X \to \alpha \in CF(P)$ fulfilling $X \in comp_1(\mathcal{K})$. If $X \in comp(\mathcal{K}) \setminus comp_1(\mathcal{K})$, the expanding production is determined by the corresponding first component.

Definition 7 ($DFA(G)$)
Let $G = (\mathcal{K}, T, P, S) \in CCFG$. The Deterministic Finite Automaton *for G is defined as $DFA(G) :=$*

$$(Q_G, \Sigma_G, \delta_G, S_G, F_G)$$

where $S_G :=$ 1-Closure$(\{[S' \to .S]\})$ is the initial state, *$\Sigma_G := comp(\mathcal{K}) \cup T \cup \{p \in CF(P) \mid \mathcal{S}(p) \notin comp_1(\mathcal{K})\}$ the* input alphabet, *δ_G the* transition function *defined for $\xi \in comp(\mathcal{K}) \cup T$ and $f_i \in CF(P)$, $\mathcal{S}(f_i) \notin comp_1(\mathcal{K})$, by*

$$\delta_G(q, \xi) := \text{1-}Closure(\{[X_j \to \alpha_j\xi.\beta_j] \mid [X_j \to \alpha_j.\xi\beta_j] \in q\}),$$

$$\delta_G(q, f_i) := \text{1-}Closure(\{[\mathcal{S}(f_i) \to .\mathcal{D}(f_i)] \mid \exists[X_j \to \alpha_j.\mathcal{S}(f_i)\beta_j] \in q\}),$$

Q_G is the set of the states *given by*

$$\{q \mid \exists u \in (comp(\mathcal{K}) \cup T \cup CF(P))^* : \delta_G(S_G, u) = q\},$$

and $F_G := \{q \in Q_G \mid [X \to \alpha.] \in q, X \to \alpha \in CF(P)\}$ is the set of the final states.

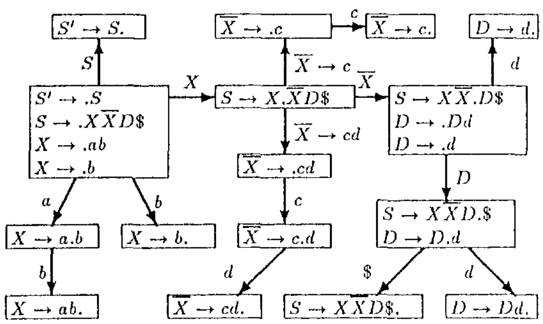

Figure 2: $DFA(G)$

The first difference to the usual context-free automaton is that we allow transitions under $f_i \in CF(P)$, if we have $\mathcal{S}(f_i) \notin comp_1(\mathcal{K})$. The second point is that we use 1-*Closure* instead of the usual closure. $DFA(G)$ for the example grammar is shown in Figure 2. The conflict is removed because we can now distinguish two cases by looking at the information additionally stored. In [SV90], only the first idea was realized obviously leading to a weaker automaton.

5 The Analysis

To use $DFA(G)$, the usual pushdown is extended by a data-structure consisting in a list of partial derivation trees. This list *future* collects all information determined by Reduce's relative to first nonterminal components and is used to drive the transitions under $p \in CF(P)$ in $DFA(G)$ as soon as we have to investigate nonterminal components $X_i \notin comp_1(\mathcal{K})$. The change between the two different kinds of control leads to a new characterization of conflicts.

For better explanation, we use a list *past* parallel to *future* where all Reduce operations performed so far are stored. An example for the new data-structures is shown in Figure 3. We use it to explain how they are built up during the analysis. The first operations on this *past* were Shift(w_1), Shift(w_2), Reduce($A \to w_2$). From CFGs, we know that any Reduce takes place at the end of the sentential form generated so far. This remains true. Thus, we can argue completely analogous as far as *past* is concerned.

But we investigate coupled productions as, e.g., $(Z_1, Z_2) \to (w_1 A, U_1 U_2)$, A, (Z_1, Z_2), $(U_1, U_2) \in \mathcal{K}$. We know that coupled nonterminal components are located at the same depth of the derivation tree and that they are substituted by components of the same coupled production. Therefore, when inserting any p, $\mathcal{S}(p) \in comp_1(\mathcal{K})$, in *past*, e.g., $Z_1 \to w_1 A$, we additionally insert the coupled productions, e.g., $Z_2 \to U_1 U_2$, in *future*. In general,

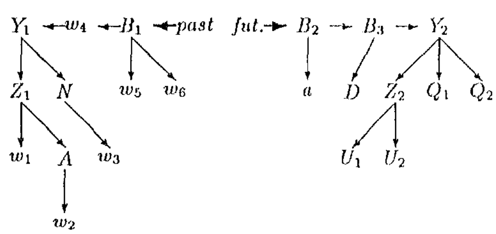

Figure 3: The New Data-Structures

there are two cases to distinguish depending on p_1 inserted in *past*. If $\mathcal{D}(p_1)$ contains only symbols in $\mathcal{K}[1] \cup T$ (i.e. only uncoupled ones), the coupled $p_2, \ldots, p_r$ are inserted as the first up to the $(r-1)$th element in *future*. (E.g. for $(Z_1, Z_2) \to (w_1 A, U_1 U_2)$.) Otherwise, we behave as it is done for $(Y_1, Y_2) \to (Z_1 N, Z_2 Q_1 Q_2)$ in Figure 3. I.e. the subtrees in *future* for those symbols in $\mathcal{D}(p_2)$, ..., $\mathcal{D}(p_r)$ coupled to first components in $\mathcal{D}(p_1)$ become the sons of these elements. Thus, we maintain the property that the symbols at each fixed depth in *past* and *future* together form an element of $ED(\mathcal{K}, T)$.

Thereby, in addition to Shift's which are handled as usual, we know what to do during a sequence of Reduce operations relative to elements of $comp_1(\mathcal{K})$. Now, let us be in the situation that we have to use the information in *future*, e.g. a transition under $B_2 \to a =: p_i$ from the topmost state. Then, we create a pointer *presence* walking on *future*. We push $\delta_G(q_{top}, p_i)$ and make *presence* point onto the first son ξ of $\mathcal{D}(p_i)$. Let q be the new topmost state. We have to distinguish three cases:

$\xi \in T$: If ξ is the next input symbol, we push $\delta_G(q, \xi)$. Otherwise, the whole input is rejected. *presence* now points on the brother of ξ.

$\xi \in comp(\mathcal{K}) \setminus comp_1(\mathcal{K})$: *future* already stores the expansion $\xi \to \beta$. We push $\delta_G(q, \xi \to \beta)$. *presence* now points on the first symbol in β.

$\xi \in comp_1(\mathcal{K})$: *future* does not store information about ξ, but ξ and its coupled components represent a complete independent analysis problem which has to be solved recursively. E.g., this is the case for D, (U_1, U_2), and (Q_1, Q_2). The recursive call of the procedure starts with the topmost state since it contains all items $[\xi \to .\alpha]$. Each recursion needs separate data-structures. Details are described in [Pit93].

If *presence* encounters no brother, we have to reduce. Let $Y \to \gamma$ be the production at whose last symbol *presence* points. We pop $|\gamma| + 1$ states. The additional pop compared to the context-free case results from the transition under $Y \to \gamma$. *presence* walks to the brother of Y in *future* and we push $\delta_G(q', Y)$ if q' is the new topmost state. If Y is the root of the first tree in *future*, its complete subtree is moved from *future* to *past* and *presence* is deleted.

We ouput $p \in P$ when reducing its last component. Thus, our result is a rightmost derivation in inverse order.

6 The Definition

So far, we did not discuss the situation that there are distinct transitions fitting for the same state in $DFA(G)$. Shift-Reduce and Reduce-Reduce conflicts are forbidden as they are for CFGs. The new conflicts result if we have to decide whether we push $\delta_G(q_{top}, f_j)$, $f_j \in CF(P)$, or Shift resp. Reduce as usual. If a state q shows such a "new" conflict, it contains two items of the kind $[Z_i \to \gamma_i . Y_j \eta_i]$ and $[X_l \to \alpha_l . \xi \beta_l]$, $\xi \in T$, or $[X_l \to \alpha_l .]$. This is easy to decide as far as we are walking on *future*, since the information necessary is stored there. Thus, we only have a real conflict if $i = 1$ and $l = 1$ holds for the above items. Obviously, this cannot be decided deterministically, since we would have to know about the structure of the derivation tree not constructed so far. E.g., in the first conflict, we would have to say whether ξ is a son of Y_j (choose $\delta_G(q, Y_j \to \mathcal{D}(f_j))$) or whether ξ is a son of X_1 (choose $\delta_G(q, \xi)$). It follows that we need a modified definition of "conflicts" compared to CFGs.

Definition 8 (Conflict)
For any $G = (\mathcal{K}, T, P, S) \in CCFG$, $DFA(G)$ shows a conflict *if at least one of its states contains a subset of the following kind:*

$$
\begin{aligned}
(R\text{-}R)\ &\{[X \to \alpha.], [Y \to \beta.] \mid X, Y \in comp_1(\mathcal{K}), \\
&\qquad X \to \alpha, Y \to \beta \in CF(P)\} \\
(S\text{-}R)\ &\{[X \to \alpha.], [Y \to \beta.a\gamma] \mid X, Y \in comp_1(\mathcal{K}), \\
&\qquad X \to \alpha, Y \to \beta a\gamma \in CF(P), a \in T\} \\
(S\text{-}E)\ &\{[X \to \alpha.a\beta], [Z \to \gamma.Y_j\eta] \mid X, Z \in comp_1(\mathcal{K}), \\
&\qquad X \to \alpha a\beta, Z \to \gamma Y_j\eta \in CF(P), \\
&\qquad a \in T, Y_j \in comp(\mathcal{K}) \setminus comp_1(\mathcal{K})\} \\
(R\text{-}E)\ &\{[X \to \alpha.], [Z \to \gamma.Y_j\eta] \mid X, Z \in comp_1(\mathcal{K}), \\
&\qquad Y_j \in comp(\mathcal{K}) \setminus comp_1(\mathcal{K}), \\
&\qquad X \to \alpha, Z \to \gamma Y_j\eta \in CF(P), \}
\end{aligned}
$$

Definition 9 ($LR(0)$ **in** $CCFG$)
$G \in CCFG$ *is* $LR(0)$ $\iff$ $DFA(G)$ *has no conflicts.*

Theorem 2 *Let $G \in CCFG$ be $LR(0)$. Our algorithm deterministically solves the wordproblem for any $w \in T^*$, $n := |w|$, in time $O(n)$ by constructing a rightmost derivation relative to G if $w \in L(G)$, and, if $w \notin L(G)$, by rejecting the input. In addition, the algorithm shows the Valid Prefix Property.*

Proof: The linear time complexity follows since we only need a constant amount of additional steps per context-free step for *past* and *future*. $DFA(G)$ is determined only once for each G. The VPP holds since it holds for the context-free algorithm and *future* additionally ensures that the coupling is correct. ∎

$LR(k)$-Coupled-Context-Free Grammars result from the above by resolving conflicts in $DFA(G)$ by adding a lookahead set to the items which are involved in a conflict. For this purpose, we use the mappings $FIRST_k$ and $FOLLOW_k$ as defined for $LL(k)$-Coupled-Context-Free Grammars in [Pit94]. There, these mappings are generalized such that they take the coupling between the components of each nonterminal into account instead of working simply on $CF(G)$. Thus, we treat only complete parentheses as a context-free nonterminal and the result is much more exact as, e.g., in [SV90]. This results in an adequate generalization of the $LR(k)$-notion for $CCFG$.

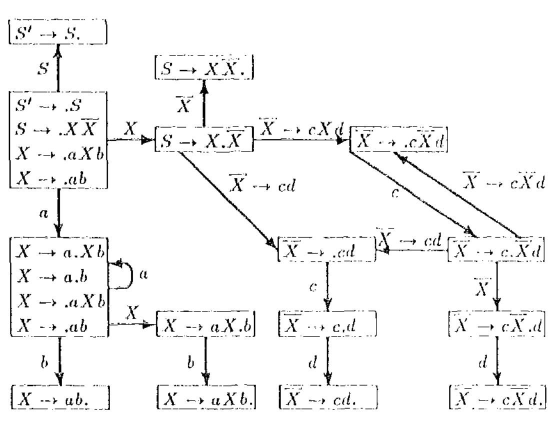

Figure 4: $L = \{a^n b^n c^n d^n \mid n \geq 1\}$

Example 2 *The language $\{a^n b^n c^n d^n \mid n \geq 1\}$ generated by the grammar in Example 1 shows the $LR(0)$-property. $DFA(G)$ is shown in Figure 4.*

Example 3 *The language $\{w\$w \mid w \in \{a,b\}^*\}$ modelling cross-serial dependencies can be generated by the $LR(1)$-grammar $(\{S, (X, \overline{X})\}, \{\$, a, b\}, P, S) \in CCFG(2)$ where $P := \{S \to X\$\overline{X}, (X, \overline{X}) \to (aX, a\overline{X}) \mid (bX, b\overline{X}) \mid (\varepsilon, \varepsilon)\}$.*

References

[DP89] J. Dassow, G. Păun: *Regulated Rewriting in Formal Language Theory.* Springer 1989

[GH69] S.A. Greibach, J.E. Hopcroft: *Scattered Context Grammars.* J. Comput. Syst. Sci. 3 (1969), pp. 232-247

[GHR92] Y. Guan, G. Hotz, A. Reichert: *Tree Grammars with multilinear Interpretation.* Technical Report, Univ. Saarbrücken 1992

[Gua92] Y. Guan: *Klammergrammatiken, Netzgrammatiken und Interpretationen von Netzen.* PhD thesis, Univ. Saarbrücken 1992

[HP94] G. Hotz, G. Pitsch: *Fast Uniform Analysis of Coupled-Context-Free Grammars.* In Proc. of ICALP'94

[Hig84] J. Higginbotham: *English is not a Context-Free Language.* Ling. Inquiry 15 (1984), pp. 225-235

[JLT75] A.K. Joshi, L.S. Levy, M. Takahashi: *Tree Adjunct Grammars.* J. Comput. Syst. Sci. 10 (1975), pp. 136-163

[Jos85] A.K. Joshi. *Tree Adjoining Grammars: How Much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions.* In Natural Language Parsing, Cambridge University Press 1985, pp. 206-250

[Jos87] A.K. Joshi: *Introduction to Tree Adjoining Grammars.* In Mathematics of Language, John Benjamins Publishing Company 1987

[Knu65] D.E. Knuth. *On the Translation of Languages from Left to Right.* Information and Control 8 (1965), pp. 607-639

[Pit93] G. Pitsch: *Analyse von Klammergrammatiken.* PhD thesis, Univ. Saarbrücken 1993

[Pit94] G. Pitsch: *LL(k)-Parsing of Coupled-Context-Free Grammars.* to appear in Computational Intelligence 1994

[Shi86] S. Shieber. *Evidence against Context-Freeness of Natural Language.* Linguistics and Philosophy 8 (1986), pp. 333-343

[SV90] Y. Schabes, K. Vijay-Shanker: *Deterministic Left to Right Parsing of Tree Adjoining Languages.* ACL'90, pp. 276-283