Interner Bericht

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

PARTITIONED CHAIN GRAMMARS

by

Peter Schlichtiger

20/79 November 1979

PARTITIONED CHAIN GRAMMARS

Peter Schlichtiger
Universität Kaiserslautern
Fachbereich Informatik
D-6750 KAISERSLAUTERN
Federal Republic of Germany

0. ABSTRACT

This paper introduces a new class of grammars, the partitioned chain grammars, for which efficient parsers can be automatically generated. Besides being efficiently parsable these grammars possess a number of other properties, which make them very attractive for the use in parser-generators. They for instance form a large grammarclass and describe all deterministic context-free languages. Main advantage of the partitioned chain grammars however is, that given a language it is usually easier to describe it by a partitioned chain grammar than to construct a grammar of some other type commonly used in parser-generators for it.

1. INTRODUCTION

In parsing the decision which action a parser is going to perform next largely depends on the already recognized part of the derivation tree. Thus all derivations of a grammar will have to obey certain conditions if a particular parsing-scheme is to work in linear time for it. Consequently the definition of those classes of grammars for which some parsing-scheme works in linear time is usually formulated in terms of restrictions on derivations. Derivations however are very complex structures, that make it very difficult for the constructor of a grammar to check whether the restrictions imposed on it by such a definition are really met. This must be considered a mayor drawback of all parser-generators which have so far been built for such grammars. The partitioned chain grammars show that much simpler structures than derivations, namely chains (as introduced in [Nijholt 77]) and a partition of the nonterminal alphabet, suffice to define a large class of efficiently parsable grammars. Using only simple structures in the definition of a grammarclass has two mayor advantages :

1. Testing, whether a certain grammatical construct obeys the definition becomes much easier.
2. By increasing the intelligibility of the definition many faulty constructs can be avoided in the first place.

Still, even the construction of such a grammar can be very difficult, if its grammarclass is not large enough. Obviously only a big class of grammars gives the constructor of a grammar a good chance, that the grammatical construct he might think of immediately to describe some language feature, will not be violating the definition. Thus it actually is the combination of both, a large grammarclass and a comprehensive definition, that distinguishes the partitioned chain grammars from all the other wellknown classes of grammars used for parser-generators.

Section 2 of this paper gives a formal definition of the partitioned chain grammars. It furthermore states some interesting properties of this grammarclass and compares it to other grammarclasses wellknown in the field of syntactical analysis. Section 3 contains the most interesting results about partitioned chain languages. Section 4 deals with a parsing-method for partitioned chain grammars.

The reader is assumed to be familiar with the basic concepts of context-free grammars and parsing, in particular with the definition and the parsing-methods for LR(k)-, LALR(k)-, SLR(k)- and LL(k)-grammars as described in [Aho,Ullman 72].

A context-free grammar (cfg) is denoted by $G = (N,T,P,S)$, where N is the set of <u>nonterminals</u> (denoted by $A,B,C,D,\ldots$), T is the set of <u>terminals</u> (denoted by $a,b,c,d,\ldots$), P is the set of <u>productions</u> and $S \in N$ is the <u>startsymbol</u>. Further on $N \cup T = V$, the elements of which are denoted by X,Y,Z. Elements of T^* will be denoted by u,v,w,x,y,z; elements of V^* by $\alpha,\beta,\gamma,\delta,\ldots$. The symbol ε is reserved for the empty word. In addition note, that

- ${}_1(\alpha)$ denotes the first symbol of α
- the <u>left-corner</u> of a production $A \to \alpha$ is ${}_1(\alpha)$

- a cfg $G=(N,T,P,S)$ is called ε-free if P contains no ε-productions (not even $S \to \varepsilon$)
- every cfg in this paper is reduced

2. PARTITIONED CHAIN GRAMMARS

Chains, as they are defined here, differ slightly from the definition by A.Nijholt in that a chain may contain a nonterminal or ε as its last element.

DEFINITION: (chain)

Let $G=(N,T,P,S)$ be a cfg.

If $X_o \in V$ then $CH(X_o)$, the set of chains of X_o, is defined by

$$CH(X_o) = \{<X_o,\ldots,X_n> \mid X_o\ldots X_n \in (N^*V \cup N^+\{\varepsilon\}) \text{ and } X_o \underset{L}{\Rightarrow} X_1\sigma_1 \underset{L}{\Rightarrow} \ldots \underset{L}{\Rightarrow} X_n\sigma_n,\ \sigma_i \in V^*, 1 \leq i \leq n\}$$

Other important notions in the definition of partitioned chain grammars are that of conflictchains and a k-follow set of a chain.

DEFINITION: (conflictchains)

Let $G=(N,T,P,S)$ be a cfg and let $\equiv$ be an equivalence relation on N.

Two different chains

$\pi_1 = <X_o,\ldots,X_n> \in CH(X_o)$, $X_o \in V$, $n \geq 0$ and

$\pi_2 = <Y_o,\ldots,Y_m> \in CH(Y_o)$, $Y_o \in V$, $m > 0$

are called conflictchains respecting $\equiv$ of type

a) iff $X_n = Y_m$, $n > 0$ and $X_{n-1} \not\equiv Y_{m-1}$

b) iff $X_n = Y_m$ and $n=0$

c) iff $X_n \in T$ and $Y_m = \varepsilon$

DEFINITION: (k-follow set of a chain)

Let $G=(N,T,P,S)$ be a cfg and let $k \geq 0$ be an integer.

Furthermore let $A \to \rho X \sigma$ be a production in P and let $\pi = <X_o,\ldots,X_n> \in CH(X)$ be a chain in G. Then

$$f_k(\pi,\sigma,follow_k(A)) = \{y \mid y \in first_k(\sigma_n \sigma\, follow_k(A)) \text{ and } X_o \underset{L}{\Rightarrow} X_1\sigma_1 \underset{L}{\Rightarrow} \ldots \underset{L}{\Rightarrow} X_n\sigma_n, \sigma_i \in V^*, 1 \leq i \leq n\}$$

is called the <u>k-follow set of chain π with repect to $A \rightarrow \rho\underline{X}\sigma$</u>, where the underlined symbol marks the beginning of chain π.

We are now ready to define the <u>p</u>artitioned <u>c</u>hain grammars with <u>k</u> symbols lookahead (abbreviated PC(k)-grammars).

<u>DEFINITION: (PC(k)-grammar)</u>

Let $G=(N,T,P,S)$ be a cfg and let $k \geq 0$ be an integer.
The augmented grammar for G is defined to be the grammar $G_a = (N \cup \{S'\}, T \cup \{\Delta\}, P \cup \{S' \rightarrow \Delta S\}, S')$, where Δ is not in T and S' is not in N.

G is a <u>PC(k)-grammar</u> iff there is an equivalence relation $\equiv$ such that the following conditions hold:

1) if $A \rightarrow \rho X \sigma$, $B \rightarrow \rho Y \bar{\sigma} \in (P \cup \{S' \rightarrow \Delta S\})$, $\rho \neq \varepsilon$ and $A \equiv B$ then
   a) there are no conflictchains respecting $\equiv$ $\pi_1 \in CH(X), \pi_2 \in CH(Y)$ of type a) or b) such that
      $f_k(\pi_1, \sigma, follow_k(A)) \cap f_k(\pi_2, \bar{\sigma}, follow_k(B)) \neq \emptyset$
   and
   b) there are no conflictchains respecting $\equiv$ $\pi_1 \in CH(X), \pi_2 \in CH(Y)$ of type c), where $\pi_1 = \langle X, \ldots, a \rangle, a \in T$, such that
      $first_k(a\ f_k(\pi_1, \sigma, follow_k(A))) \cap f_k(\pi_2, \bar{\sigma}, follow_k(B)) \neq \emptyset$

2) if $A \rightarrow \rho$ and $B \rightarrow \rho\sigma$ are different productions in P and $A \equiv B$ then
   $follow_k(A) \cap first_k(\sigma\ follow_k(B)) = \emptyset$

Since chains can apparently become infintely long, if a grammar contains leftrecursive nonterminals, this definition on a first glance may seem to make sense only for non-leftrecursive grammars. However this is only true for $k=0$. For $k \geq 1$ leftrecursive nonterminals may very well occur in a PC(k)-grammar. The main reason for this is, that one actually does not have to look at any chains, which contain some nonterminal more than $k+1$ times, to find out whether a grammar is PC(k). This is an immediate consequence of the following two lemmas.

<u>LEMMA 2.1</u>

All PC(k)-Grammars, $k \geq 0$, are cycle-free.

Proof:

The proof is ommitted here. It is quite simple for ε-free PC(k)-grammars (see [Schlichtiger1 79]).

□

LEMMA 2.2

Let $G=(N,T,P,S)$ be a cycle-free cfg and $k \geq 1$ an integer. If there is a chain $\pi \in CH(X), X \in N$, in G which contains some nonterminal $A \in N$ more than $k+1$ times, then there also has to be a chain $\pi' \in CH(X)$ in G which contains that nonterminal A at most $k+1$ times and for which the following holds:

1) $f_k(\pi',\sigma,follow_k(A)) = f_k(\pi,\sigma,follow_k(A))$

with respect to any production $A \to \rho \underline{X} \sigma \in (P \cup \{S' \to \Delta S\}), \rho \neq \varepsilon$,

and

2) the last two elements of π and π' are equal.

Proof:

As G is cycle-free, every leftrecursive leftmost derivation in G is of the form $A \overset{+}{\underset{L}{\Rightarrow}} A\sigma$, where $\sigma \overset{*}{\not\Rightarrow} \varepsilon$.

Hence every leftmost derivation belonging to a chain $\pi = \langle X_o, \ldots, X_n \rangle$ in $CH(X_o), X_o \in N$, which contains some nonterminal $A \in N$ $k+\ell$ times, $\ell > 1$, has to be of the form

$$X_o \overset{*}{\underset{L}{\Rightarrow}} A\gamma \overset{+}{\underset{L}{\Rightarrow}} A\sigma_1\gamma \overset{+}{\underset{L}{\Rightarrow}} A\sigma_2\sigma_1\gamma \overset{+}{\underset{L}{\Rightarrow}} \cdots \overset{+}{\underset{L}{\Rightarrow}} A\sigma_{k+\ell-1}\cdots\sigma_1\gamma$$
$$\overset{*}{\underset{L}{\Rightarrow}} X_n\beta\sigma_{k+\ell-1}\cdots\sigma_1\gamma,$$

where $\beta,\gamma \in V^*$ and $\sigma_i \in V^+$ for $1 \leq i \leq (k+\ell-1)$.

So for any production $A \to \rho \underline{X} \sigma \in (P\ \{S' \to \Delta S\}), \rho \neq \varepsilon$,

$$f_k(\pi,\sigma,follow_k(A)) = \{ y \mid y \in first_k(\beta\sigma_{k+\ell-1}\cdots\sigma_1\gamma\sigma\ follow_k(A)) \text{ and } X_o \overset{*}{\underset{L}{\Rightarrow}} A\gamma \overset{+}{\underset{L}{\Rightarrow}} A\sigma_1\gamma \overset{+}{\underset{L}{\Rightarrow}} \cdots \overset{+}{\underset{L}{\Rightarrow}} A\sigma_{k+\ell-1}\cdots\sigma_1\gamma \overset{*}{\underset{L}{\Rightarrow}} X_n\beta\sigma_{k+\ell-1}\cdots\sigma_1\gamma \text{ belongs to } \pi. \}$$

Now consider the chain $\pi' \in CH(X_o)$ which results from π by eliminating the first $\ell-1$ occurences of A in π. Obviously every leftmost derivation belonging to π' must be of the form

$$X_o \overset{*}{\underset{L}{\Rightarrow}} A\gamma \overset{+}{\underset{L}{\Rightarrow}} A\sigma_\ell\gamma \overset{+}{\underset{L}{\Rightarrow}} \cdots \overset{+}{\underset{L}{\Rightarrow}} A\sigma_{k+\ell-1}\cdots\sigma_\ell\gamma \overset{*}{\underset{L}{\Rightarrow}} X_n\beta\sigma_{k+\ell-1}\cdots\sigma_\ell\gamma$$

where $\beta,\gamma \in V^*$ and $\sigma_j \in V^+$ for $\ell \leq j \leq (k+\ell-1)$

and hence we have for every production $A \to \rho \underline{X_o} \sigma \in (P \cup \{S' \to \Delta S\}), \rho \neq \varepsilon$:

$$f_k(\pi', \sigma, follow_k(A)) = \{x \mid x \in first_k(\beta \sigma_{k+\ell-1} \cdots \sigma_\ell \gamma \sigma \; follow_k(A)) \text{ and } X_o \overset{*}{\underset{L}{\Rightarrow}} A\gamma \overset{+}{\underset{L}{\Rightarrow}} A\sigma_\ell \gamma \overset{+}{\underset{L}{\Rightarrow}} \cdots \overset{+}{\underset{L}{\Rightarrow}} A\sigma_{k+\ell-1} \cdots \sigma_\ell \gamma \overset{*}{\underset{L}{\Rightarrow}} X_n \beta \sigma_{k+\ell-1} \cdots \sigma_\ell \gamma \text{ belongs to } \pi' \}$$

As G is assumed cycle-free no $\sigma_i, 1 \leq i \leq (k+\ell-1)$, can generate the empty word. Consequently each word in $first_k(\beta \sigma_{k+\ell-1} \cdots \sigma_\ell)$ has to be at least k terminals long, which proves that $f_k(\pi', \sigma, follow_k(A)) = f_k(\pi, \sigma, follow_k(A))$ with respect to any production $A \to \rho \underline{X_o} \sigma \in (P \; \{S' \to \Delta S\}), \rho \neq \varepsilon$.
Moreover the tail of chain π beginning with the ℓ'th A equals the tail of chain π' beginning with the first A. As this tail consists of exactly k+1 A's and as $k \geq 1$, π and π' must at least agree in their last two elements.

□

<u>THEOREM 2.1</u>
To decide if a cfg $G=(N,T,P,S)$ is a PC(k)-grammar for some integer $k \geq 1$ only those chains have to be considered, which do not contain any nonterminal more than k+1 times.
<u>Proof:</u>
Let π_1, π_2 be conflictchains and let π_1 contain some nonterminal more than k+1 times. According to lemma 2.2 there has to be another chain π_1', which contains that nonterminal at most k+1 times, such that π_1', π_2 are conflictchains too.

□

Mainly as a consequence of theorem 2.1 it is sufficient to look at chains up to a maximal length of $(k+1) \cdot |N| + 1$ links, to decide if a given grammar is a PC(k)-grammar for a certain $k \geq 0$. Looking at grammars for programming languages one will find, that the chains occuring in such grammars are much shorter than $(k+1) \cdot |N| + 1$. An average length of 3 or 4 links should be realistic.

The following theorems show, that the class of PC(k)-grammars is indeed quite large compared to other grammarclasses used in parser-generators. Unfortunately most of the proofs have to be omitted in this paper. The main reason for this is, that most of the grammarclasses and the important properties of these classes used in parsing are defined in terms of derivations. It however can become very difficult to prove such properties for grammars which are defined in terms of much simpler structures like chains. First of all one would have to show which influence the restrictions on chains have on the structure of derivations.

The proofs of all the theorems can be found in [Schlichtiger1 79] for an ε-free version of the PC(k)-grammars. These proofs are further aggravated if grammars with ε-productions are considered.

THEOREM 2.2

Every strong LL(k)-grammar is PC(k)

Proof: (Sketch)

Let $G=(N,T,P,S)$ be a cfg, $k \geq 0$, and assume G is not PC(k). Then G in particular cannot be a PC(k)-grammar with respect to the equivalence relation $=$ on $N \cup \{S'\}$.

1) A violation of condition 2) for PC(k)-grammars with respect to $=$ quite immediately causes a conflict to the definition of the strong LL(k)-grammars.

2) If there is a violation of condition 1) for PC(k)-grammars repecting $=$, then there are productions $A \to \rho \underline{X} \sigma$, $A \to \rho \underline{Y} \bar{\sigma}$ in $P \cup \{S' \to \Delta S\}$, where $\rho \neq \varepsilon$ and $\pi_1 = \langle X_0, \ldots, X_n \rangle \in CH(X)$, $\pi_2 = \langle Y_0, \ldots, Y_m \rangle \in CH(Y)$ are conflictchains for which $first_k(X_n f_k(\pi_1, \sigma, follow_k(A))) \cap first_k(Y_m f_k(\pi_2, \bar{\sigma}, follow_k(A))) \neq \emptyset$.
If $A \to \rho X \sigma$ and $A \to \rho Y \bar{\sigma}$ are different productions a violation of the defintion of strong LL(k)-grammars is evident.
If these productions are equal, then a LL(k)-conflict cannot be shown that easily. Nevertheless one has to exist. □

THEOREM 2.3

Every PC(k)-grammar is LR(k).

Proof:

The proof, which is rather difficult and lengthy, is omitted in this paper. □

An analogous theorem is not true for LALR(k)- and SLR(k)-grammars. Instead the following theorem holds.

THEOREM 2.4

There are

1) PC(k)-grammars, which are not LALR(k) (SLR(k))

and

2) SLR(k)- (LALR(k)-) grammars which are not PC(k).

Proof:

1) The grammar $G_1=(\{S,A,B,C,D,E\},\{a,b\},P_1,S)$, where
   $P_1= \{S\to aA\ ,\ S\to bB\ ,\ A\to Ca\ ,\ A\to Db\ ,\ B\to Cb\ ,\ B\to Da\ ,\ C\to E\ ,\ D\to E\ ,\ E\to\varepsilon\}$,
   is a LL(1)-grammar. According to theorem 2.2 it then is a PC(1)-grammar too. It however is not LALR(1) (the set of LALR(1)-items valid for the viable prefixes aE and bE $\{[C\to E.\ ,\ a|b]\ ,\ [D\to E.\ ,\ a|b]\}$ is inconsistent).
   As the class of SLR(1)-grammars is a proper subset of the class of LALR(1)-grammars, G_1 is not SLR(1) either.

2) The grammar $G_2(\{S,A\},\{a,b\},P_2,S)$, where
   $P_2= \{S\to aaab\ ,\ S\to aAa\ ,\ A\to aa\}$,
   is SLR(1). It however is not PC(1) (Consider the productions $S\to aaab$, $S\to aAa$. There are two conflictchains of type b) <a> and <A,a> which violate condition 1), because
   $f_1(\langle a\rangle,ab,follow_1(S)) \cap f_1(\langle A,a\rangle,a,follow_1(S)) = \{a\} \neq \emptyset)$.
   □

Besides the very wellknown classes of LL(k)- and LR(k)-grammars other interesting classes, for which efficient parsing is possible, have been developed. The simple chain grammars [Nijholt 77,78] are such a class. In a way the PC(k)-grammars can be regarded as an extension of simple chain grammars. For both types of grammars chains are the essential structure.

DEFINITION: (simple chain grammar)

An ε-free cfg $G=(N,T,P,S)$ is said to be a simple chain grammar if it satisfies the following two conditions

1) if $A\to\rho X\sigma$ and $A\to\rho Y\bar{\sigma}$ are in P and $X \neq Y$ then
   $first_k(X) \cap first_k(Y) = \emptyset$

2) if $A \rightarrow \rho$ and $A \rightarrow \rho\sigma$ are in P then $\sigma = \varepsilon$.

<u>THEOREM 2.5</u>

Every simple chain grammar is an ε-free PC(O)-grammar with respect to the equivalence relation = , and vice versa.

<u>Proof:</u> (Sketch)

Condition 2) for simple chain grammars and for PC(O)-grammars respecting = coincide.

If condition 1) for simple chain grammars is not met, then there must be two conflictchains of type a) or b) that violate condition 1) for PC(O)-grammars respecting =.

If condition 1) for ε-free PC(O)-grammars with respect to = is not satisfied, i.e. if there are two productions $A \rightarrow \rho X \sigma$, $A \rightarrow \rho Y \overline{\sigma} \in (P\ \{S' \rightarrow \Delta S\})$ and conflictchains $\pi_1 \in CH(X)$, $\pi_2 \in CH(Y)$, then condition 1) for simple chain grammars is obviously violated if $X \neq Y$, because π_1 and π_2 have to end with the same symbol. If $X = Y$ this is not as obvious. It can however be concluded from the fact that π_1 and π_2 must be different.

□

The <u>p</u>redictive <u>LR(k)</u>-grammars [Soisalon,Ukkonen 76] (abbreviated PLR(k)-grammars) are another interesting class of grammars. Comparison with the PC(k)-grammars yields some astonishing results.

<u>DEFINITION: (PLR(k)-grammar)</u>

A grammar $G=(N,T,P,S)$ is <u>PLR(k)</u>, $k \geq 0$, if G is LR(k) and in the augmented grammar $G_a=(N \cup \{S'\}, T \cup \{\Delta\}, P \cup \{S' \rightarrow \Delta S\}, S')$ the conditions

1) $S' \overset{*}{\underset{R}{\Rightarrow}} \rho A r \underset{R}{\Rightarrow} \rho X \alpha r$

2) $S' \overset{*}{\underset{R}{\Rightarrow}} \overline{\rho} B \overline{r} \underset{R}{\Rightarrow} \overline{\rho} \beta X \gamma \overline{r} = \rho X \gamma \overline{r}$

3) $first_k(\alpha r) \cap first_k(\gamma \overline{r}) \neq \emptyset$

always imply : $\rho A = \overline{\rho} B$.

One can show that PC(k)-grammars with respect to the equivalence relation = are quite closely related to PLR(k)-grammars.

It turns out that these two classes mainly differ in how much attention is payed to the context a lookahead may appear in. In PLR(k)-grammars the context of a lookahead is considered more important than in PC(k)-grammars. One can however quite easily extend the PC(k)-grammars to a class, which properly contains the PLR(k)-grammars. This is achieved by considering so-called context-dependent follow sets instead of global follow sets in the definition of PC(k)-grammars ($cdf_k(\rho,A)=\{y \mid S \overset{*}{\underset{R}{\Rightarrow}} \rho Ar, A\in N, r\in T^* \text{ and } y={}_k(r)\}$ is called the follow set of A depending on the context ρ) Without going into detail -the interested reader is refered to [Schlichtiger1 79] -,we just state the relationship between the such defined class of extended PC(k)-grammars and the PLR(k)-grammars.

THEOREM 2.6

1) The class of extended PC(k)-grammars properly contains all PLR(k)-grammars.
2) The class of all extended PC(k)-grammars with respect to the equivalence relation = is equal to the class of PLR(k)-grammars.

Proof:

The proof is very lengthy and is therefore omitted in this paper.

□

At first sight noone would surely have suspected this close relationship. As far as clarity and comprehension of grammar-definitions are concerned, the definition of PLR(k)-grammars is even worse than that of LR(k)-grammars. It hardly gives the constructor of a grammar a chance to make it a PLR(k)-grammar. Thus this example quite drastically emphasises the necessity of trying to make grammardefinitions used in parser-generators as understandable as possible. It at the same time shows that it may be worth the effort.

The last grammarclass we are going to look at,is the class of partitioned LL(k)-grammars [Friede 78] . This class is of particular interest because it is an extension of the wellknown strict deterministic grammars [Harrison,Havel 73].

Partitioned LL(k)-grammars in contrast to strict deterministic grammars make use of a lookahead of length k.

DEFINITION: (partitioned LL(k)-grammars)

A cfg $G=(N,T,P,S)$ is said to be a partitioned LL(k)-grammar for an integer $k \geq 0$ iff there is an equivalence relation $\equiv$ on N, such that for any two productions $A \to \alpha\beta$, $B \to \alpha\gamma$ in P, where $A \equiv B$ and $\alpha, \beta, \gamma \in V^*$, the following condition is satified:

If $first_k(\beta\ follow_k(A)) \cap first_k(\gamma\ follow_k(B)) \neq \emptyset$ then

either i) ${}_1(\beta), {}_1(\gamma) \in N$ and ${}_1(\beta) \equiv {}_1(\gamma)$

or ii) ${}_1(\beta), {}_1(\gamma) \in T$

or iii) $\beta = \varepsilon$ and $\gamma = \varepsilon$ and $A = B$

Comparing PC(k)- and partitioned LL(k)-grammars yields

THEOREM 2.7

The partitioned LL(k)-grammars form a proper subset of the class of PC(k)-grammars.

Proof:

A thorough proof is not given.

Assuming that a cfg is not PC(k) one can first show, that it then cannot be a partitioned LL(k)-grammar either. Thus every partitioned LL(k)-grammar has to be PC(k). That this inclusion is proper can be immediately infered from the fact, that PC(k)-grammars in contrast to partitioned LL(k)-grammars may be leftrecursive. □

3. PARTITIONED CHAIN LANGUAGES

Theorem 2.2 to theorem 2.7 show, that the class of PC(k)-grammars is a large grammarclass. The same is true for the class of context-free languges (cfl) described by PC(k)-grammars.

THEOREM 3.1

The PC(O)-grammars generate exactly all deterministic prefix-free context-free languages.

Proof:

According to theorem 2.3 PC(O)-grammars cannot generate more than the LR(O)-languages, which are equal to the class of all deterministic prefixfree cfl.
According to theorem 2.7 the class of PC(O)-grammars generates at least all the partitioned LL(O)-languages. The partitioned LL(O)-grammars however are exactly the strict deterministc grammars, which are known to describe all deterministic prefixfree deterministic cfl.

□

THEOREM 3.2

The PC(1)-grammars describe exactly all deterministic cfl.

Proof:

According to theorem 2.3 PC(1)-grammars cannot describe more than LR(1)-grammars, that is, they cannot generate more than all deterministic cfl.
According to theorem 2.7 the PC(1)-grammars must at least describe all the partitioned LL(1)-languages, which are all deterministic cfl.

□

REMARK:

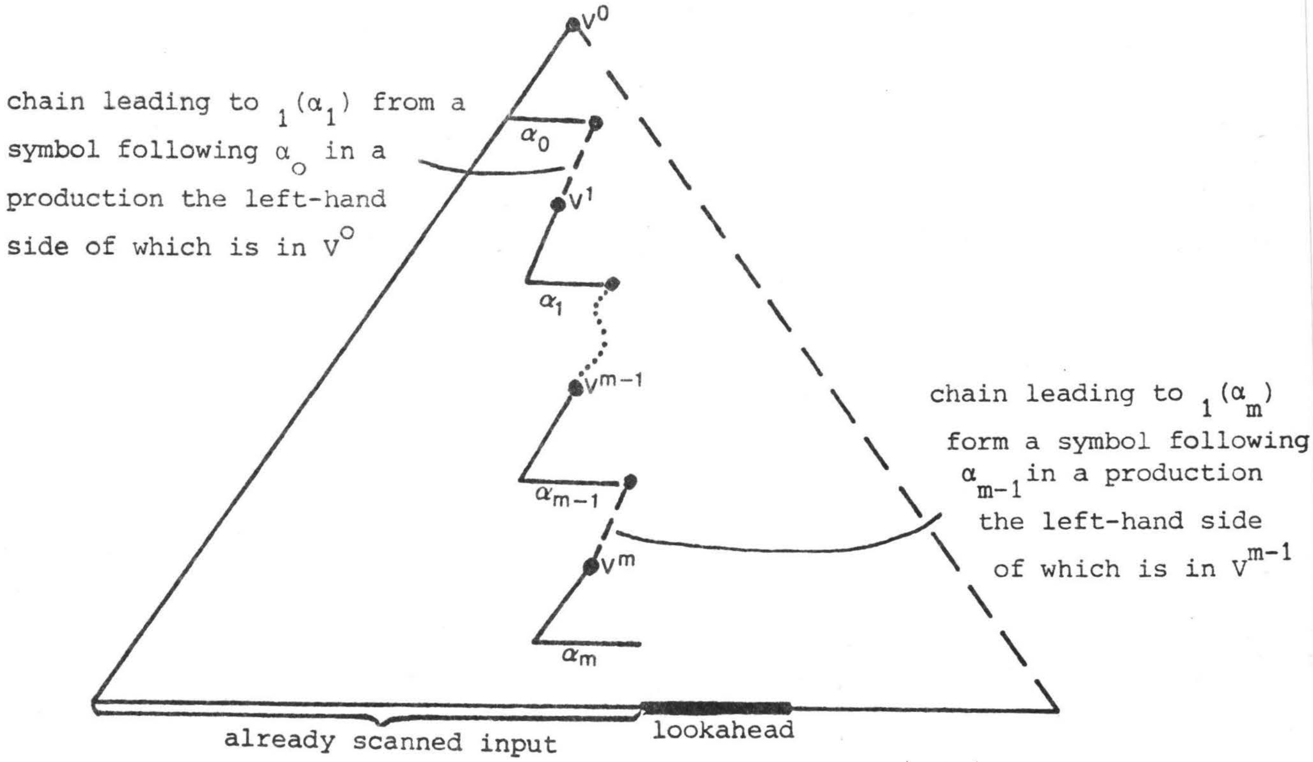
For k>O , the PC(k)-grammars with respect to the equivalence relation = generate exactly the LL(k)-languages, which are a proper subset of the deterministic cfl. This shows,that partitions must be considered a powerfull tool in language-description.

4. THE PARSING OF PARTITIONED CHAIN GRAMMARS

The parsing method for PC(k)-grammars will only be discussed rather informally here. A precise description of a PC(k)-parsing-algorithm can be found in [Schlichtiger2 79] (for ε-free PC(k)-grammars also in [Schlichtiger1 79]).

Let G=(N,T,P,S) be a PC(k)-grammar with respect to some equivalence relation ≡ and let W be the partition induced on N∪{S'} by ≡.

Assume that the parser has reached a configuration, which describes the following structure



where

- $V^i \in W$ for $0 \leq i \leq m$
- $\alpha_i \neq \varepsilon$, $0 \leq i \leq m$, is a nonempty prefix of the right-hand side of a not yet completely recognized production, the left-hand side of which is in V^i
- $S' \in V^o$ and $\alpha_o = \Delta$

Note that at the beginning $m = 0$.

The parser proceeds as follows :
First of all he has to find out, if α_m is a proper prefix of the right-hand side of the production he is presently trying to recognize, or if α_m already is that whole right-hand side. On the basis of condition 2) for PC(k)-grammars this decision can be made by simply looking at the look-ahead.

a) If α_m is a proper prefix, the parser will have to compute the symbol immediately right to α_m in this right-hand side. He does this by trying to recognize the chain, which has to start with the symbol next to α_m and leads to either ε or the next input- symbol. For this purpose he looks at all chains (with less than $k+2$ repetitions), which end with either ε or the next input-symbol and which begin with any symbol that can immediately follow α_m in a production, the left-hand side of which is in V^m. If there are such chains ending with ε as well as chains ending with the next input-symbol,condition 1b) guarantees, that by inspecting the lookahead it can be determined which kind of chain is correct in the present context. After this decision the last element of the chain presently under consideration is known. If it is the next input-symbol, this symbol is scanned, thereby of course changing the lookahead. If it is ε ,then because of condition 1a) for conflictchains of type a), the parser can determine the equivalence class of the predecessor of ε in the chain, again by examining the lookahead. As this predecessor must be the left-hand side of an ε-production, by condition 2) it is moreover possible to decide exactly which nonterminal in this equivalence class is the correct one. Let X denote the next input-symbol or this nonterminal respectively.

If there is a chain of length 1 among the chains leading to X from some symbol to the right of α_m , then the only element of this chain may be the symbol next to α_m the parser has been trying to find. On the basis of condition 1a) for conflictchains of type b) the parser can decide this question by inspecting the present lookahead. If X really is the symbol following α_m , then α_m is extended by X and the parser has apparently reached a situation similar to the one this description started with.

If only chains longer than 1 have to be considered, condition 1a) for conflictchains of type a) guarantees, that by looking at the lookahead, the class V^{m+1} of the predecessor of X in the chain the parser is presently trying to recognize can be determined.Note,that V^{m+1} actually is the class of the left-hand side of a production with left-corner $X = \alpha_{m+1}$. Before being able to go on in recognizing the chain, this production has to be recognized completely. This again leaves the parser in a situation similar to the one we started with.

b) If the parser by examing the lookahead finds, that α_m is the right-hand side he has been looking for, his next step will be to determine the left-hand side of this production exactly. Condition 2) requires,that dependent on the lookahead it must be possible to decide which nonterminal, say A , in V^m is the left-hand side of α_m. That completes the recognition of this production.
As A is the predecessor of $_1(\alpha_m)$ in the chain the parser must now continue to compute, in order to find the symbol immediately right to α_{m-1} , there must be at least one chain going to A from some symbol following α_{m-1} in a production,whose left-hand side is in V^{m-1}.
Now, one of these chains can of course contain A as it's sole element, which means, that A may itself be the symbol next to a_{m-1} the parser is looking for. As before this can be decided on the basis of condition 1a) for conflictchains of type b) by inspecting the present lookahead and if it turns out to be the next symbol of the right-hand side beginning with α_{m-1}, then α_{m-1} is extended by A ,leaving the parser in a situation analogous to the one we started off from.
If on the other hand the present lookahead only permits chains longer than 1 , condition 1a) for conflictchains of type a) demands,that dependent on the lookahead the class (call it V^m again) of the predecessor of A in the chain to be recognized can be determined. As before this is the class of the left-hand side of a production (with left-corner X) ,which must be recognized next. So the parser once again has come to a situation,which resembles the initial one.

The parser goes on recognizing the parse-tree in this manner node by node until the production $S' \to \Delta S$ is recognized. If at that time all the input has been scanned, then the input-word will be accepted.

For this very intuitively presented parsing method an efficient parsing-algorithm has been developed, which works in linear time and for $k<2$ will generally use less space than a LALR(k)-parser.

5. CONCLUSION

PC(k)-grammars prove to be very well suited for parser-generators. This is so for three reasons:

1) Efficient parsers can be constructed for PC(k)-grammars
2) PC(k)-grammars form a large class of grammars and languages
3) The definition of PC(k)-grammars can be understood and verified easily

PC(k)-grammars differ from other wellknown grammarclasses used for parser-generators in that 2) and 3) usually do not occur together. Nevertheless this is a desirable combination, which helps to make the construction of a grammar much easier. Ease of construction however is an important argument in favor of using parser-generators in practice. That is why in [Schlichtiger1 79] further attention has been given to methods and algorithms, which can be used to support the design of PC(k)-grammars. Among others an efficient algorithm for constructing a partition according to which a given grammar G is PC(k) (if such a partition at all exists) is for instance introduced.

REFERENCES

[Aho, Ullman 72] A.V. Aho , J.D.Ullman : The Theory of Parsing, Translation and Compiling I,II Prentice Hall, Inc. (1972)

[DeRemer 71] F.L. DeRemer: Simple LR(k)-Grammars CACM 14, pp 453-460 (1971)

[Friede 78] D. Friede: Über deterministische kontextfreie Sprachen und rekursiven Abstieg Bericht Nr.49 d. FB Informatik Universität Hamburg (1978)

[Friede 79] D. Friede: Partitioned LL(k) Grammars Lecture Notes in Computer Science 71 pp 245-255 (1979)

[Ginsburg, Greibach 66] S. Ginsburg, S.A. Greibach: Deterministic Context-Free Languages Information and Control 9,pp 620-648 (1966)

[Harrison 78] M.A. Harrison: Introduction to Formal Language Theory Addison-Wesley, Reading, Mass. (1978)

[Harrison, Havel 73] M.A.Harrison, I.M.Havel Strict Deterministic Grammars JCSS 7,pp 237-277 (1973)

[Harrison, Havel 74] M.A. Harrison, I.M. Havel: On the Parsing of Deterministic Languages JACM 21,pp 525-548 (1974)

[Mayer 78] O. Mayer: Syntaxanalyse Bibliographisches Institut Mannheim

[Nijholt 77] A. Nijholt: Simple Chain Grammars Lecture Notes in Computer Science 52 pp 352-364 (1977)

[Nijholt 78] A. Nijholt: On the Parsing and Covering of Simple Chain Grammars Lecture Notes in Computer Science 62 pp 330-344 (1978)

[Rosenkrantz, Lewis 70] D.J. Rosenkrantz, P.M. Lewis II:
Deterministic Left Corner Parsing
IEEE Conf. Rec. of the 11'th Annual Symp. on Switching and Automata Theory, pp 139-152 (1970)

[Schlichtiger1 79] P. Schlichtiger:
Kettengrammatiken - ein Konzept zur Definition handhabbarer Grammatikklassen mit effizientem Analyseverhalten
Doctorial Thesis Uni. Kaiserslautern (1979)

[Schlichtiger2 79] P. Schlichtiger:
On the Parsing of Partitioned Chain Grammars
unpublished manuscript, (1979)

[Soisalon, Ukkonen 76] E. Soisalon-Soininen, E. Ukkonen:
A Characterisation of LL(k)-Languages
Proc. of the 3rd Coll. on Automata, Languages and Programming, pp 20-30 (1976)

Bisher im Fachbereich Informatik erschienene Interne Berichte:

1.	Dausmann, Persch, Winterstein "Concurrent Logik".	Jan.	79
2.	Balzert "Die Programmiersprache PLASMA 78".	Febr.	79
3.	Avenhaus, Madlener "String Matching and Algorithmic Problems in Groups".	März	79
4.	Patock "Jahresbericht des Informatikrechenzentrums".	März	79
5.	Hartenstein, Hubschneider, Rosebrock, Wiedemann "Ein SC/MP Multi-Mikrorechner-System zur Straßenverkehrs-Datenerfassung".	April	79
6.	Dausmann "MODULA 7/32 A version of MODULA for the INTERDATA 7/32".	März	79
7.	Bergsträßer "Ein Assembler für Lisp-Werk einer Lisp- Maschine".	Jan.	79
8.	Hartenstein "Verallgemeinerung der Prinzipien Mikroprogrammierter Rechnerstrukturen".	Febr.	79
9.	Dieckmann "Entwurf spezialisierter Rechnernetze zur Unterstützung Modularer Programmierung".	Juni	79
1o.	Hartenstein, v. Puttkamer "Ansätze für Integrierten Hardware/Software Entwurf".	Juni	79
11.	Konrad "Asynchroner Datenpfad zur losen Kopplung von Mikro-rechnern".	Mai	79
12.	Hartenstein "LSI Chip Design: from Evolution to Revolution".	Juni	79
13.	Hartenstein, Hörbst, v. Puttkamer "Loosely coupled Micros - Distributed Function Architectures: a Design Kit and Development Tool".	Juni	79
14.	Konrad "Communication And Testing In a Loosely Coupled Multi Microcomputer System".	Aug.	79
15.	Hartenstein, v. Puttkamer "KARL A Hardware Description Language as Part of a CAD Tool for VLSI".	Juli	79
16.	Avenhaus, Madlener "AN ALGORITHM FOR THE WORD PROBLEM IN HNN EXTENSIONS AND THE DEPENDENCE OF ITS COMPLEXITY ON THE GROUP REPRESENTATION".	Juli	79

17. Nehmer "Implementierungstechniken für Monitore". Jan. 80

18. Nehmer "The Implementation of Concurrency for a PL/I-like Language". Jan. 80

19. Patock "Jahresbericht des Informatikrechenzentrums". Febr. 80

20. Schlichtiger "PARTITIONED CHAIN GRAMMARS". Nov. 79

21. Schlichtiger "ON THE PARSING OF PARTITIONED CHAIN GRAMMARS". Dez. 79

22. Schlichtiger "ON HOW TO CONSTRUCT EFFICIENTLY PARSABLE GRAMMARS". Febr. 80