Interner Bericht

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

ON HOW TO CONSTRUCT EFFICIENTLY PARSABLE GRAMMARS

by

Peter Schlichtiger

22/80 February 1980

ON HOW TO CONSTRUCT EFFICIENTLY PARSABLE GRAMMARS

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

1. Introduction

All grammar classes, parser-generators have so far been built for, share two important properties:

1) an efficient parser can be generated for any grammar of the class

and

2) all language features commonly appearing in programming languages can be described (as far as they can at all be described by a context-free grammar).

Taking the view-point of an user of parser-generators, one further property will be of importance:

3) given some language, it should be easy to construct a grammar of the required type for it.

Although very desirable, this third requirement is only very poorly met by the wellknown grammar classes used for parser-generators. There are different reasons for this. The main reason seems to be either a too restricted grammar class (this for instance is the main reason why the construction of a LL(1)-grammar can become very cumbersome), or a definition, which is too complex to guide the construction of a grammar (this for instance is true for LR(1)-grammars).

Partitioned chain grammars, like all grammars used for parser-generators, satisfy the above requirements 1) and 2) (see [Schlichtiger2,3 79]). They differ from these classes in their response to the third requirement. Partitioned chain grammars define a large grammar class and possess an intelligible definition as well. They will therefore be easier to construct than one of the other types of grammars.Yet,the construction of a partioned chain grammar will of course not be trivial. That is why this paper introduces several algorithms to support their construction.

Section 2 of this paper gives a formal definition of partitioned chain grammars and section 3 states some results on the grammar- and language class. Section 4 introduces several algorithms and shows, how these can be used to ease the construction of a partitioned chain grammar.

The reader is assumed to be familiar with the basic concepts of context-free grammars and parsing as described in [Aho.Ullman 72] .
A context-free grammar (abbreviated cfg) is denoted by $G=(N,T,P,S)$,
where
- N is the set of nonterminals (denoted by A,B,C,D,...)
- T is the set of terminals (denoted by a,b,c,d,...)
- P is the set of productions
- $S \in N$ is the startsymbol

$N \cup T$ is denoted by V , the elements of which will be 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.

2. Definitions

The definition of a grammar, which is supposed to be understood easily, must avoid using complex structures like derivations. Basing a grammar definition on too simple structures will on the other hand severly restrict the grammar class defined. In this situation chains (first introduced by A.Nijholt in [Nijholt 77]) realize a good compromise. The example of partitioned chain grammars will show, that chains, although they are a much simpler structure than derivations, permit the definition of large grammar classes.

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

$$CH(X_o) = \{ \langle X_o,\ldots,X_n \rangle \mid n \geq 0,\ X_o \ldots X_{n-1} \in N^*,\ X_n \in (N \cup T \cup \{\varepsilon\}) \text{ and } X_o \underset{L}{\Rightarrow} X_1\sigma_1 \underset{L}{\Rightarrow} \cdots \underset{L}{\Rightarrow} X_n\sigma_n ,\ \sigma_i \in V^* ,\ 1 \leq i \leq n \}$$

Note, that chains, as they are defined here, differ from the chains defined by A.Nijholt in that they may contain a nonterminal or ε as their last element. Furthermore note, that $\langle\varepsilon\rangle$ is not a chain.

A very important notion in connection with the definition of partitioned chain grammars is that of a <u>k-follow set of a chain</u>.

<u>DEFINITION: (k-follow set of a chain)</u>

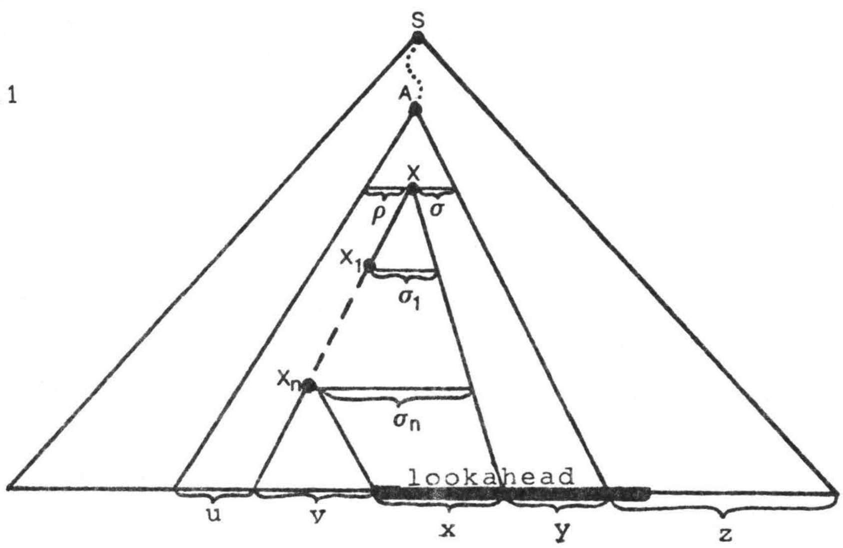
Let $G=(N,T,P,S)$ be a cfg and let $\equiv$ be an equivalence relation on N. Furthermore let $A \to \rho X \sigma$ be a production in P, let $\pi = \langle X_o,\ldots,X_n \rangle$ be a chain in $CH(X)$ and let $F_k(A)$ be a subset of $follow_k(A)$, the global follow set of A. Then

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

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

Although this definition might seem a little complicated at first sight, it actually describes a quite simple relationship between a lookahead of k symbols and a chain. This relationship is depicted in the following figure

Figure 2.1



where $\rho \overset{*}{\Rightarrow} u$, $X_n \overset{*}{\Rightarrow} v$, $\sigma_n \overset{*}{\Rightarrow} x$, $\sigma \overset{*}{\Rightarrow} y$, $z \in follow(A)$ and

$$lookahead = {}_k(xyz) \in f_k(\langle X,X_1,\ldots,X_n \rangle,\sigma,\ follow_k(A)).$$

Different chains, which may appear in a similar context, must to a certain extend be distinguishable on account of the lookahead. The following definition describes exactly which differences have to be recogniced.

<u>DEFINITION: (conflict chain)</u>

Let $G=(N,T,P,S)$ be a cfg and let $\equiv$ be an equivalence relation on N. Two chains $\pi_1 = \langle X_o,\ldots,X_n \rangle \in CH(X_o)$, $\pi_2 = \langle Y_o,\ldots,Y_m \rangle \in CH(Y_o)$, $X_o,Y_o \in V$, are called conflict chains (with respect to $\equiv$) of type

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

b) iff $n=0$, $m>0$ and $X_n = Y_m$

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

DEFINITION: (PC(k)-grammar)

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
$G_a = (N\cup\{S'\}, T\cup\{\Delta\}, P\cup\{S'\to\Delta S\}, S')$, where Δ is not in T and S' is not in N.
G is a partitioned chain grammars with k symbols lookahead (abbreviated PC(k)-grammar) iff there is an equivalence relation $\equiv$ on $N\cup\{S'\}$, such that the following conditions hold for G_a :

1) if $A\to\rho X\sigma$, $B\to\rho Y\overline{\sigma}$ $\in (P\cup\{S'\to\Delta S\})$,$\rho\neq\varepsilon$ and $A \equiv B$ then

   a) $f_k(\pi_1,\sigma,follow_k(A)) \cap f_k(\pi_2,\overline{\sigma},follow_k(B)) = \emptyset$
   for any two conflict chains $\pi_1 \in CH(X)$, $\pi_2 \in CH(Y)$ of type a) or b)

   and

   b) $first_k(a\ f_k(\pi_1,\sigma,follow_k(A)) \cap f_k(\pi_2,\overline{\sigma},follow_k(B)) = \emptyset$
   for any two conflict chains $\pi_1 \in CH(X)$, $\pi_2 \in CH(Y)$ of type c),
   where $\pi_1 = \langle X,\ldots,a\rangle$,$a \in T$.

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

The class of PC(k)-grammars can be extended by paying closer attention to the context a production appears in in the derivation tree. As will be seen in the sequel,the right-context α of a production $A\to\rho X\sigma$ in some rightmost derivation $S \overset{*}{\underset{R}{\Rightarrow}} \alpha Az \underset{R}{\Rightarrow} \alpha\rho X\sigma z$ serves our purpose best. By making use of the right-context of a production the definition of PC(k)-grammars can be changed to the definition of what will be called an EPC(k)-grammar (abbreviation for extended PC(k)-grammar). Both definitions will actually only differ in the follow sets they use. Instead of considering the global follow set of the left-hand side of a production, the definition of EPC(k)-grammars will use follow sets, which also depend on the right-context of the production. These follow sets will therefore be called contextdependent.

DEFINITION: (contextdependent follow set)

Let $G=(N,T,P,S)$ be a cfg and let $k \geq 0$ be an integer.
The contextdependent k-follow set of a nonterminal A with respect to the right-context α (abbreviated $cdf_k(\alpha,A)$) is defined by
$cdf_k(\alpha,A) = \{y \mid S \overset{*}{\underset{R}{\Rightarrow}} \alpha Az \text{ and } y = {}_k(z)\}$

REMARK :

- $cdf_k(\alpha,A) = \emptyset$ if there is no rightmost derivation such that α is a valid right-context of A.
- $cdf_k(\alpha,A) \subset follow_k(A)$

The definition of EPC(k)-grammars is now attained by replacing every global follow set by contextdependent follow sets as shown below.

DEFINITION: (EPC(k)-grammar)

Let $G=(N,T,P,S)$ be a cfg and let $k \geq 0$ be an integer.
The augmented grammar G_a is defined as in the definition of PC(k)-grammars.
G is an EPC(k)-grammar iff there is an equivalence relation $\equiv$ on $N \cup \{S'\}$, such that the following conditions hold for G_a :

1) if $A \to \rho X\sigma$, $B \to \rho Y\overline{\sigma} \in (P \cup \{S' \to \Delta S\})$,$\rho \neq \varepsilon$ and $A \equiv B$, then
   a) $f_k(\pi_1,\sigma,cdf_k(\alpha,A)) \cap f_k(\pi_2,\overline{\sigma},cdf_k(\alpha,B)) = \emptyset$
      for any two conflict chains $\pi_1 \in CH(X)$, $\pi_2 \in CH(Y)$ of type a) or b) and any $\alpha \in (\Delta V^* \cup \{\varepsilon\})$

   and

   b) $first_k(a\ f_k(\pi_1,\sigma,cdf_k(\alpha,A)) \cap f_k(\pi_2,\overline{\sigma},cdf_k(\alpha,B)) = \emptyset$
      for any two conflict chains $\pi_1 \in CH(X)$, $\pi_2 \in CH(Y)$ of type c),
      where $\pi_1 = \langle X,\ldots,a\rangle$,$a \in T$, and any $\alpha \in (\Delta V^* \cup \{\varepsilon\})$

2) if $A \to \rho$ and $B \to \rho\sigma$,$A \equiv B$, are different productions in P then
   $cdf_k(\alpha,A) \cap first_k(\sigma\ cdf_k(\alpha,B)) = \emptyset$ for any $\alpha \in (\Delta V^* \cup \{\varepsilon\})$

3. Partitioned chain grammars and languages

The definition of PC(k)-grammars gives the constructor of a grammar a much better understanding of how his grammar should look like than for instance the definition of LR(k)-grammars. It would nevertheless be rather difficult to construct a PC(k)-grammar if very many different conflict chains would have to be considered. Luckily this will however not be the case with grammars for programming languages. The chains that have to be considered in such grammars are on the contrary rather short (an average length of about 3 or 4 should be realistic). There are mainly two reasons for this:

1) Only chains, which do not contain any nonterminal more than k+1 times (k, the length of the lookahead, will usually be 1) have to be examined.
   Note, that this implies that PC(k)-grammars may contain left recursive nonterminals for k>0.
2) The constructor of a grammar will use a certain nonterminal in a very limited environment only; he would otherwise run the risk of losing overview over his grammar. Chains will therefore hardly contain very many different nonterminals.

The following theorems show, that PC(k)- and EPC(k)-grammars form a large grammar- and language class compared to other classes used for parser-generators. The corresponding proofs have been omitted in this paper for the sake of brevity.

THEOREM 3.1

1) The class of EPC(k)-grammars properly contains the class of PC(k)-grammars for any k>0. Both classes coincide for k=0.
2) The class of LL(k)-grammars is a proper subset of the class of EPC(k)-grammars and the class of PC(k)-grammars properly contains all strong LL(k)-grammars.
3) The class of simple chain grammars (see [Nijholt 77,78]) is a proper subset of the class of PC(0)-grammars. It is equal to the class of all ε-free PC(0)-grammars with respect to the equivalence relation =.
4) The partitioned LL(k)-grammars (see [Friede 79]), which are an extension of the strict deterministic grammars (see [Harrison,Havel 73]), are a proper subset of the class of PC(k)-grammars.
5) The class of predictive LR(k)-grammars (see [Soisalon,Ukkonen 76]) is a proper subset of the class of all EPC(k)-grammars. It is equal to the class of all EPC(k)-grammars with respect to the equivalence relation =.
6) Every EPC(k)-grammar is LR(k).

THEOREM 3.2

1) For every $k>0$ the class of EPC(k)-grammars generates the same language class as the class of PC(k)-grammars.
2) The PC(0)-grammars generate all deterministic prefix-free context-free languages.
3) For any $k\geq 1$ the class of PC(k)-grammars generates all the deterministic context-free languages.
4) For every $k\geq 1$ PC(k)-grammars with respect to the equivalence relation = generate exactly all LL(k)-languages.

4. Supporting the construction of partitioned chain grammars

The preceding chapters showed, that partitioned chain grammars form a large grammar class and possess a comprehensible definition as well. It should therefore in general be easier to construct a partitioned chain grammar than some other type of grammar, which does not share this property. This advantage of partitioned chain grammars can be increased further by combining the advantage of the simple definition of PC(k)-grammars with respect to the equivalence relation = with the advantage of the larger grammar class of general PC(k)-grammars or even EPC(k)-grammars in the following manner:

Let $k=1$, as this is the only case of any practical relevance.

The constructor is given the definition of a PC(1)-grammar with respect to the equivalence relation =. The grammar $G=(N,T,P,S)$ he will construct will however most probably not be PC(1) with respect to =.

The construction of a grammar, which really is PC(1), can then be supported by a kind of 'construction supporting system' , which works as follows:

First of all it will have to find out according to which partition of N G is PC(1).

There is a quite trivial way of doing so. One simply has to take one partition after another (there are only finite many different partitions of N) and check if G is PC(1) with respect to it. This method however has two major drawbacks. Firstly it is very inefficient and if G is not PC(1) it secondly does not provide the constructor of G with any information about how he should try to modify his grammar in order to make it PC(1).

These drawbacks are avoided by the following algorithm :

ALGORITHM 4.1

```
input :  - a cfg  G=(N,T,P,S) , where  N={A_1,...,A_n}
output:  - if  G  is PC(1) :  a partition  W  according to which  G
                              is PC(1)
         - if  G  is not PC(1) :  a partition  W  and a list of conflicts
                                  with respect to W
method:
W̄ := {{A_1},...,{A_n}};  co the partition induced on  N  by  ≡  oc
conflict := false;
repeat
W := W̄;
for  all productions A→α,B→β ∈ (P∪{S'→ΔS}), where A ≡ B  do
 begin
a: if  α=ρXσ , β=ρYσ̄  and  ρ≠ε
   then  begin
  a1:     for  all chains  π=<Y_o,...,Y_m> ∈ CH(Y), where m>0 ,Y_m = X and
               π does not contain any nonterminal more than twice
          do   if  f_1(<X>,σ,follow_1(A)) ∩ f_1(π,σ̄,follow_1(B) ≠ ∅
               then  begin
                     conflict := true;
                     report that there are conflict chains <X> and π of
                     type b) such that  A→ρXσ, B→ρYσ  violate condition 1a)
                     of PC(1)-grammars with respect to the partition W;
                     end;
  a2:     if  there is a chain  π_2= <Y,..,ε> ∈ CH(Y)
          then for  all  a ∈ T  such that there is a chain  <X,...,a> ∈ CH(X)
               do  if  a ∈ f_1(π_2,σ̄,follow_1(B))
                   then  begin
                         conflict := true;
                         report that there are conflict chains <X,..,a>,
                         <Y,...,ε>  of type c) such that  A→ρXσ , B→ρYσ
                         violate condition 1b) of   PC(1)-grammars
                         with respect to the partition W;
                         end;
         end;
b; if  (A→α) ≠ (B→β)  and  β=ασ ,σ∈V*  and  follow_1(A) ∩ first_1(σ follow_1(B))≠∅
   then  begin
         conflict := true;
```

```
        report that  A→α , B→β  violate condition 2) of PC(1)-grammars
        with respect to the partition  W.
        end;
   end;
  if not conflict
  then for all productions A→α , B→β ∈ (PU{S'→ΔS}) , where A ≡ B do
c:     if α=ρXσ , β=ρYσ̄ and ρ≠ε
       then for all chains π₁=<X₀,...,Xₙ>∈CH(X), π₂=<Y₀,...,Yₘ>∈CH(Y),where
                n,m>0 ,Xₙ= Yₘ, and where neither π₁ nor π₂ contain
                any nonterminal more than twice
            do if Xₙ₋₁≢ Yₘ₋₁ and
                  f₁(π₁,σ,follow₁(A)) ∩ f₁(π₂,σ̄,follow₁(B)) ≠ ∅
               then begin
                    co the class of X in W is denoted by [X] oc
                    W̄ := (W̄ - [Xₙ₋₁]) - [Yₘ₋₁];
                    W̄ := W̄ ∪ {[Xₙ₋₁] ∪ [Yₘ₋₁]} ;
                    end;
  until conflict or W̄ = W ;
```

The only conflicts, that can be solved by introducing a partition of the nonterminal alphabet into a grammar, are violations of condition 1a) by conflict chains of type a) (this case is marked by c: in algorithm 4.1). It suffices to change the partition by joining the classes of the last but one element of both conflict chains to eliminate such a conflict (as the resulting partition will contain the last but one element of both chains in the same class, they no longer are conflict chains).

If any conflict of some other type (marked by a1:,or a2: ,or a3: in algorithm 4.1) occurs during the construction of a partition by the algorithm, the grammar cannot be PC(1). Thus the constructor will have to eliminate these conflicts by himself. For that purpose algorithm 4.1 provides him with the partition W constructed so far and a precise description of all conflicts of the kind marked by a1: , or a2: ,or b: in algorithm 4.1 occuring with respect to W. Note, that conflicts of these types are much easier to survey than the kind of conflict removed from the grammar by the algorithm.

After all reported conflicts have been eliminated by the constructor, the modified grammar can again be examined by algorithm 4.1 . The algorithm will either find, that the grammar now is PC(1) with respect to W, or it will again have to change W by joining different classes because some conflict chains of type a) violate condition 1a). In the latter case new conflicts of the kind marked by a1: , a2: , or b: may occur with respect to the changed partition. These conflicts will again have to be eliminated by the constructor, before algorithm 4.1 can continue to construct a valid partition in the manner already described.

Used in this stepwise fashion, algorithm 4.1 will be a great help in the construction of PC(1)-grammars. It however still requires some assistance by the constructor, if the grammar is not PC(1). One way to reduce the amount of assistance needed during the construction is to let the constructor decide not to eliminate the conflicts reported to him, but to ask the construction supporting system to check whether the grammar is EPC(1). Only if the grammar is not EPC(1) either, will the constructor in this case be borthered.

Two algorithms are necessary to check whether a given grammar G=(N,T,P,S) is EPC(1):
First of all the construction supporting system has to compute all different pairs $(cdf_1(\gamma,A), cdf_1(\gamma,B)), \gamma \in V^*$, $A,B \in N$, of nonempty contextdependent 1-follow sets. The algorithm doing so is closely related to the wellknown algorithm for constructing the canonical collection of sets of LR(1)-items (see [Aho,Ullman 72]). This is an immediate consequence of the following observation:

Let $I_1(\gamma)$ be a set of valid LR(1)-items for the viable prefix γ .
Then the following holds for any LR(1)-item $[A\to.\alpha\ ,\ a] \in I_1(\gamma)$:
$[A\to.\alpha\ ,\ a] \in I_1(\gamma)$ iff $\exists\ S \overset{*}{\underset{R}{\Rightarrow}} \gamma Aw$, $A\to\alpha \in P$ and $a = {}_1(w)$.
Hence $cdf_1(\gamma,A) = \{\ a\ |\ [A\to.\alpha\ ,\ a] \in I_1(\gamma)\ \}$.

Let $P_1(A,B)$, $A,B \in N$, denote the set of all pairs $(cdf_1(\gamma,A), cdf_1(\gamma,B))$, $\gamma \in V^*$, $cdf_1(\gamma,A) \neq \emptyset$ and $cdf_1(\gamma,B) \neq \emptyset$. Then the following extension of the algorithm for constructing the cononical collection of sets of LR(1)-items will do, to compute all different pairs:

<u>for</u> all $I_1(\gamma)$ belonging to the canonical collection of sets of
    LR(1)-items
<u>do</u> <u>for</u> all $A,B \in N$ <u>do</u>

<u>if</u> $\{a \mid [A\to .\alpha\ ,\ a] \in I_1(\gamma)\} \neq \emptyset$ <u>and</u> $\{b \mid [B\to .\beta\ ,\ b] \in I_1(\gamma)\} \neq \emptyset$
<u>then</u> $P_1(A,B) = P_1(A,B) \cup (\{a \mid [A\to .\alpha\ ,a] \in I_1(\gamma)\}\ ,\ \{b \mid [B\to .\beta\ ,b] \in I_1(\gamma)\})$;

For further details see [Schlichtiger1 79].

After all sets of pairs $P_1(A,B)$ have been computed, a partition according to which G will be EPC(1) has to be constructed. This can be accomplished by a straightforward modification of algorithm 4.1, which replaces all global 1-follow sets by contextdependent 1-follow sets. Instead of for instance asking if

$$f_1(\pi_1,\sigma,\text{follow}_1(A)) \cap f_1(\pi_2,\overline{\sigma},\text{follow}_1(B)) \neq \emptyset \quad ,$$

the algorithm has to check whether

$$f_1(\pi_1,\sigma,\text{cdf}_1(\gamma,A)) \cap f_1(\pi_2,\overline{\sigma},\text{cdf}_1(\gamma,B) \neq \emptyset$$

for all pairs $(\text{cdf}_1(\gamma,A),\text{cdf}_1(\gamma,B))$,$\gamma \in V^*$, in $P_1(A,B)$.

If this algorithm finds, that G is not EPC(1), the constructor will be asked to eliminate the reported conflicts. By modifying G step by step as described before, an EPC(1)-grammar can be constructed.

If G is EPC(1), it can be transformed into an equivalent PC(1)-grammar $\tilde{G} = (\tilde{N},T,\tilde{P},\tilde{S})$, where

- $\tilde{N} = \{\langle A,\text{cdf}_1^G(\gamma,A)\rangle \mid A \in N,\ \ S \overset{*}{\underset{R}{\Rightarrow}} \gamma Aw \text{ in } G\}$
- $\tilde{P} = \{\langle A,\text{cdf}_1^G(\gamma,A)\rangle \to \langle \alpha,\text{cdf}_1^G(\gamma,A)\rangle \mid A\to\alpha \in P\ ,\ \langle A,\text{cdf}_1^G(\gamma,A)\rangle \in \tilde{N}\}$,
  where $\langle \alpha,\text{cdf}_1^G(\gamma,A)\rangle$ is defined as follows:
  - if $\alpha \in T^*$ then $\langle \alpha,\text{cdf}_1^G(\gamma,A)\rangle = \alpha$
  - if $\alpha = z_o B_1 z_1 \ldots z_{i-1} B_i z_i \ldots z_{m-1} B_m z_m$,
    $m \geq 1$, $z_o, z_i \in T^*$ and $B_i \in N$, $1 \leq i \leq m$

    then $\langle \alpha,\text{cdf}_1^G(\gamma,A)\rangle = z_o \langle B_1,\text{cdf}_1^G(\gamma_1,B_1)\rangle z_1 \ldots z_{i-1} \langle B_i,\text{cdf}_1^G(\gamma_i,B_i)\rangle z_i \ldots z_{m-1} \langle B_m,\text{cdf}_1^G(\gamma_m,B_m)\rangle z_m$,

    where $\gamma_1 = \gamma z_o$ and $\gamma_j = \gamma z_o B_1 z_1 \ldots z_{j-2} B_{j-1} z_{j-1}$,$2 \leq j \leq m$.

The main idea behind this transformation is to replace each occurence of a nonterminal A in some right-sentential form γAw by a new nonterminal of the form $\langle A,\text{cdf}_1^G(\gamma,A)\rangle \in \tilde{N}$. This new nonterminal is characterized by its right-context γ in such a way, that its global 1-follow set in $\tilde{G}$, $\text{follow}_1^{\tilde{G}}(\langle A,\text{cdf}_1^G(\gamma,A)\rangle)$, is equal to $\text{cdf}_1^G(\gamma,A)$, the contextdependent 1-follow set of A with respect to the right-context γ in G.

Consequently $\tilde{G}$ will be PC(1) with respect to a partition $\tilde{W}$ of $\tilde{N}$

which is defined by :

$$<A,cdf_1^G(\gamma,A)> \equiv_{\tilde{W}} <B,cdf_1^G(\gamma,B)> \quad \text{iff} \quad A \equiv_W B$$

if G was EPC(1) with respect to W.

$\tilde{G}$ possesses one further important property as far as parsing is concerned. It right covers the original grammar G. That is, each right parse according to $\tilde{G}$ can be transformed into a valid right parse for the same input word according to G by a homomorphism. The homomorphism h needed in this case is very simple. It is defined by:

$$h(\ <A,cdf_1^G(\gamma,A)> \rightarrow <\alpha,cdf_1^G(\gamma,A)>\ ) = A \rightarrow \alpha\ .$$

Before generating a parser for a PC(1)-grammar G ,constructed with the help of a construction supporting system like the one described above, the user is strongly recommended to look at his grammar once more. The partition W computed by algorithm 4.1 is the finest partition according to which G is PC(1). That is, W is a refinement of any other partition according to which G is PC(1) too. The main reason for choosing the finest partition instead of for instance the coarsest one is, that the delay of error detection of the parser caused by the use of a partition can be considerably aggravated by using a coarse partition. On the other hand,the parser will use less space if a coarse partition is chosen. The only reasonable way out of this dilemma is to let the constructor of the grammar decide to what extend he wants to delay error detection in favour of more space-efficiency. It should therefore be left to the user to find a coarser partition if he wishes, all the more as such a partition can be attained very easily by joining classes of W. Of course not all unions of classes of W will result in a partition according to which G is PC(1), but the only PC(1)-conflicts that can occur are easily recognized (they have to be of the kind marked by a1: , a2: , and b: in algorithm 4.1) and can therefore be avoided.

5. Conclusion

Partitioned chain grammars form a new class of efficiently parsable grammars. They differ from other grammar classes wellknown in syntax analysis in that they are comparatively easy to construct. Ease of construction, which must be considered a very important argument in favour of using parser-generators, can be increased even further for partitioned chain grammars by making use of the various possibilities to support the construction of such grammars.

6. References

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

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

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

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

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

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

[Mayer 78] O.Mayer : Syntaxanalyse , Bibliographisches Institut Mannheim (1978)

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

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

[Nijholt1 79] A.Nijholt : Simple Chain Grammars and Languages Theoretical Computer Science 9 (1979) , 282-309

[Nijholt2 79] A.Nijholt : Structure Preserving Transformation on Non-Left-Recursive Grammars , Lecture Notes in Computer Science 71 (1979), 446-459

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

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

[Schlichtiger2 79] P.Schlichtiger : Partitioned Chain Grammars , Interner Bericht 20/79 (1979), University of Kaiserslautern

[Schlichtiger3 79] P.Schlichtiger : On the Parsing of Partitioned Chain Grammars , Interner Bericht 21/79 (1979), University of Kaiserslautern

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

[Ukkonen 79] E.Ukkonen : A Modification of the LR(k) Method for Constructing Compact Bottom-up Parsers , Lecture Notes in Computer Science 71 (1979) , 646-658

Bisher im Fachbereich Informatik erschienene Interne Berichte:

1. Dausmann, Persch, Winterstein — Jan. 79
"Concurrent Logik".

2. Balzert — Febr. 79
"Die Programmiersprache PLASMA 78".

3. Avenhaus, Madlener — März 79
"String Matching and Algorithmic Problems in Groups".

4. Patock — März 79
"Jahresbericht des Informatikrechenzentrums".

5. Hartenstein, Hubschneider, Rosebrock, Wiedemann — April 79
"Ein SC/MP Multi-Mikrorechner-System zur Straßenverkehrs-Datenerfassung".

6. Dausmann — März 79
"MODULA 7/32
A version of MODULA for the INTERDATA 7/32".

7. Bergsträßer — Jan. 79
"Ein Assembler für Lisp-Werk einer Lisp- Maschine".

8. Hartenstein — Febr. 79
"Verallgemeinerung der Prinzipien Mikroprogrammierter Rechnerstrukturen".

9. Dieckmann — Juni 79
"Entwurf spezialisierter Rechnernetze zur Unterstützung Modularer Programmierung".

10. Hartenstein, v. Puttkamer — Juni 79
"Ansätze für Integrierten Hardware/Software Entwurf".

11. Konrad — Mai 79
"Asynchroner Datenpfad zur losen Kopplung von Mikro-rechnern".

12. Hartenstein — Juni 79
"LSI Chip Design: from Evolution to Revolution".

13. Hartenstein, Hörbst, v. Puttkamer — Juni 79
"Loosely coupled Micros - Distributed Function Architectures: a Design Kit and Development Tool".

14. Konrad — Aug. 79
"Communication And Testing In a Loosely Coupled Multi Microcomputer System".

15. Hartenstein, v. Puttkamer — Juli 79
"KARL
A Hardware Description Language as Part of a CAD Tool for VLSI".

16. Avenhaus, Madlener — Juli 79
"AN ALGORITHM FOR THE WORD PROBLEM IN HNN EXTENSIONS AND THE DEPENDENCE OF ITS COMPLEXITY ON THE GROUP REPRESENTATION".

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

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

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

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

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

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