Interner Bericht

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

ON THE PARSING OF PARTITIONED CHAIN GRAMMARS

by

Peter Schlichtiger

21/79 December 1979

ON THE PARSING OF PARTITIONED CHAIN GRAMMARS

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

1. INTRODUCTION

One reason, why even nowadays parser-generators are not very often used by compiler-writers, is, that it usually is very difficult to construct a grammar, which obeys all the conditions required of an input-grammar. Actually this is quite impossible for someone not familiar with the theory of formal languages and syntactical analysis.
This paper introduces a new large class of grammars, the partitioned chain grammars, which, besides being efficiently parsable, also are comparatively easy to construct. This was mainly achieved by using only very simple structures, namely chains (as introduced by [Nijholt 77]) and a partition of the nonterminal alphabet, in the description of this grammarclass, instead of describing them in terms of restrictions on derivations, as this is normally done. Using only simple structures in the definition of a grammarclass has two major advantages:

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

Together with the fact, that partitioned chain grammars form a quite large class compared to other grammarclasses used for parser-generators, these advantages significantly facilitate the constructibility of such grammars.

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 grammar- and languageclasses wellknown in parsing.

Section 3 deals with a rather informal description of the parsing-method, while section 4 gives a parsing-algorithm realizing this method and shows how this algorithm can be optimized in various fashions.

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 (cfg) is denoted by $G=(N,T,P,S)$, where N is the set of nonterminals (denoted by $A,B,C,\ldots$), T is the set of terminals (denoted by $a,b,c,\ldots$), P is the set of productions and $S \in N$ is the startsymbol. Furtheron $V=N \cup T$ and the elements of V 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 :

- $_1(\alpha)$ denotes the first symbol of α
- the left-corner 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 assumed to be reduced

2. PARTITIONED CHAIN GRAMMARS AND LANGUAGES

The definition of most grammarclasses, an efficient parsing-algorithm is known for, shows, that derivations are structures, which are too complex to guide the construction of a grammar. Thus simpler structures than derivations should be used in grammardefinition. On the other hand many simple structures will not define sufficiently large grammarclasses. In this situation chains realize a good compromise.

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

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

It turns out, that it suffices to consider chains instead of derivations, to define a large class of efficiently parsable grammars. Naturally one will not be interested in all chains at a time, but only in those chains which are possible in similar contexts. As such chains may cause a conflict to the definition of partitioned chain grammars, they are called conflictchains.

DEFINITION : (conflictchains)

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

Two different chains

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

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

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 $Y_m = \varepsilon$

Before coming to the definition of partitioned chain grammars, one further definition, that of a socalled k-follow set of a chain , is needed. As will be seen in the sequel, this definition describes the relationship between a lookahead of k inputsymbols and a chain appearing in a certain context.

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 $\pi = \langle X_o,\ldots,X_n\rangle \in CH(X)$ 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 k-follow set of chain π with respect to $A \to \rho\underline{X}\sigma$, where the underlined symbol marks the beginning of chain π.

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

$G_a = (NU\{S'\}, TU\{\Delta\}, PU\{S'\to\Delta S\}, S')$, where Δ is not in T and S' is not in N.

G is called a partitioned chain grammar with k symbols lookahead (abbreviated PC(k)-grammar) iff there is an equivalence relation $\equiv$,such that the following conditions hold:

1) if $A\to\rho X\sigma$, $B\to\rho Y\overline{\sigma} \in (PU\{S'\to\Delta S\})$, $\rho\neq\varepsilon$ 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
   $f_k(\pi_1,\sigma,follow_k(A)) \cap f_k(\pi_2,\overline{\sigma},follow_k(B)) \neq \emptyset$
   and
   b) there are no conflictchains respecting $\equiv$ $\pi_1\in CH(X)$, $\pi_2\in CH(Y)$ of type c), where $\pi_1=<X,\ldots,a>$, $a\in T$, such that
   $first_k(a\ f_k(\pi_1,\sigma,follow_k(A))) \cap f_k(\pi_2,\overline{\sigma},follow_k(B)) \neq \emptyset$
2) if $A\to\rho$ and $B\to\rho\sigma$ are different productions in P and $A \equiv B$, then
   $follow_k(A) \cap first_k(\sigma\ follow_k(B)) = \emptyset$

If a grammar contains leftrecursive nonterminals,chains can apparently become infinitely long. Hence one might suspect, that for leftrecursive grammars it will not be possible to decide,if the above conditions really hold for all chains. Luckily this is not true. It is easily shown, that PC(0)-grammars cannot be leftrecursive - this would cause a violation of condition 1). For k>0 PC(k)-grammars can be leftrecursive. This can be infered from the important observation, that it suffices to only look at chains with less than k+2 repetitions of any nonterminal, to verify if the PC(k)-conditions are met, i.e. if these chains do not violate the PC(k)-conditions, any chain containing some nonterminal more than k+1 times cannot do so either.

Together with the fact, that grammars describing programming languages tend to use a particular nonterminal only in a very limited environment - a nonterminal TERM used for describing mathematical expressions will for instance hardly be used somewhere else in the grammar - this results in the astonishing observation, that the chains,which really have to be considered in a grammar for some programming language,usually

are quite short. An average length of 3 or 4 should be realistic.

The following theorems prove, that PC(k)-grammars, besides possessing a rather intelligible definition, also form a large grammar- and languageclass compared to other classes commonly used for parser-generators. For the sake of brevity the corresponding proofs have been omitted in this paper.

THEOREM 2.1

1. The class of strong LL(k)-grammars is a proper subset of the class of PC(k)-grammars.
2. Every PC(k)-grammar is LR(k).
3. The class of simple chain grammars (see [Nijholt 77,78]) is equal to the class of all ε-free PC(0)-grammars with respect to the equivalence relation =.
4. PC(k)-grammars can easily be extended to a grammarclass, which properly contains the predictive LR(k)-grammars (see [Soisalon,Ukkonen 76]). (This is achieved by replacing the global follow sets by socalled contextdependend follow sets (see [Schlichtiger 79]). In fact the predictive LR(k)-grammars coincide with the class of all PC(k)-grammars with respect to the equivalence relation =, which have been extended in this manner.
5. The partitioned LL(k)-grammars (see [Friede 78]) form a proper subset of the PC(k)-grammars.

REMARK 2.1

Anyone doubting, that the definition of PC(k)-grammars really is rather comprehensible, is invited to compare this definition to the definition of the apparently closely related class of predictive LR(k)-grammars.

THEOREM 2.2

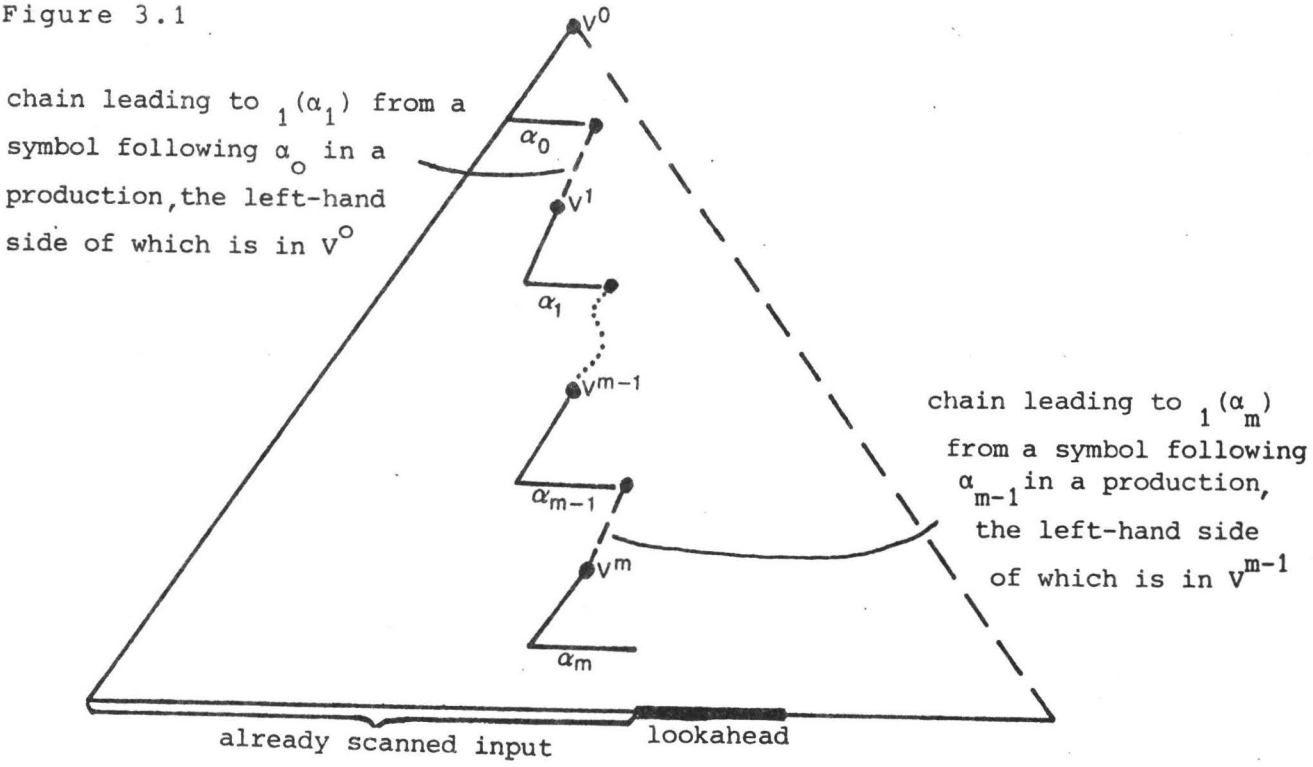
1. The PC(0)-grammars generate exactly all deterministic prefixfree context-free languages.
2. The PC(k)-grammars describe all deterministic context-free grammars, for any $k>0$.
3. For $k>0$ the PC(k)-grammars with respect to the equivalence relation = generate exactly the LL(k)-languages.

3. ON THE PARSING OF PC(k)-GRAMMARS

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

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

Figure 3.1



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, he is presently trying to recognize, or if α_m already is that whole right-hand side. Condition 2) for PC(k)-grammars guarantees, that this can be decided by simply looking at the lookahead.

a) If α_m is a proper prefix, the parser will have to compute

the symbol immediately to the right of α_m in this right-hand side. This is done by trying to recognize the chain, which begins with the symbol next to α_m and leads to either ε or the next input-symbol. For this purpose the parser 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, the next input-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 the 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 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 off 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 examining the lookahead finds, that α_m is the right-hand side it has been looking for, the next step will be to determine the left-hand side of this production exactly. Condition 2) requires, that dependend on the lookahead it must be possible to decide, which nonterminal in V^m is the left-hand side of α_m. Let $A \in N$ denote this nonterminal. That completes the recognition of this production. As A is the predecessor of $_1(\alpha_m)$ in the chain, which 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 its sole element, which means, that A may itself be the symbol next to α_{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. If A 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, which again leaves us 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 dependend 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 A), 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 $S' \rightarrow \Delta S$ is recognized. If at that time all the input has been scanned, the inputword will be accepted.

4. A PARSING-ALGORITHM FOR PC(k)-GRAMMARS

Figure 3.1 actually does not show a really general situation of the parsing-process. For this purpose Figure 3.1 has to be extended by adding a chain, which starts from some (yet unknown) symbol immediately to the right of α_m and leads to some symbol $X \in (V \cup \{\varepsilon\})$, where X is the symbol, which has been recognized last by the parser. X is ε, if the parser has decided, that ε has to be the last symbol of the chain, it is a terminal, if the parser has decided, that the last symbol has to be the next input-symbol (see Chapter 3 part a)). If $X \in N$, the parser has just recognized a production completely, the left-hand side of which is X (see Chapter 3 part b)).

For the parser to work as described in the preceding chapter, it is not necessary to store all of the structure, which so far has been recognized by the parser. It suffices to store the not yet totally known productions. A structure, as shown in Figure 3.1, can for instance be represented by the sequence:
$[V^o,\alpha_o]\ [V^1,\alpha_1]\ \ldots\ [V^{m-1},\alpha_{m-1}]\ [V^m,\alpha_m]$
According to what has been said at the beginning of this section, the last element of this sequence is extended by a third component X denoting the end of the chain starting immediately to the right of α_m. As only the last element of such a sequence will be used and manipulated by a PC(k)-parser, this sequence will be stored in a stack, the topmost element of which will be $[V^m,\alpha_m,X]$.

The PC(k)-parsing-algorithm :

Let $G=(N,T,P,S)$ be a PC(k)-grammar with respect to some equivalence relation $\equiv$. Let $W = \{V_o,\ldots,V_n\}, n \geq 0$, be the partition induced on $N \cup \{S'\}$ by $\equiv$, where $S' \in V_o$.

At the beginning the stack will just contain the element $[V_o,\varepsilon,\Delta]$. The parser will manipulate this and any other topmost stackelement as described below. It will stop as soon as the parser accepts the inputword or finds out, that it is not in $L(G)$.

Now let $[V_i,\alpha,X]$, $0 \leq i \leq n$, $\alpha \in V^*$, $X \in V \cup \{\varepsilon\}$ be the present topmost stackelement, let u be the present lookahead and let a be the next input-symbol to be scanned. The parser will then proceed as follows :

a) if there is a production $A \rightarrow \alpha X \beta \in (PU\{S' \rightarrow \Delta S\})$, where $A \in V_i$, $\beta \neq \varepsilon$ (i.e. $\beta = Y\overline{\beta}$, $Y \in V$) and $u \in first_k(\beta follow_k(A))$, then $[V_i,\alpha,X]$ will be replaced by
   - $[V_i,\alpha X,\varepsilon]$,if there is a chain $\pi \in CH(Y)$ leading to ε, such that $u \in f_k(\pi,\overline{\beta},follow_k(A))$
   - $[V_i,\alpha X,a]$,otherwise (this implies that there has to be a chain $\pi \in CH(Y)$ leading to the next input-symbol a , such that $u \in first_k(a\, f_k(\pi,\overline{\beta},follow_k(A)$

b) if there is a production $A \rightarrow \alpha B \beta \in (PU\{S' \rightarrow \Delta S\})$, where $A \in V_i$, $\alpha \neq \varepsilon$, and if there is a chain $\pi \in CH(B)$ longer than 1 of the form $\pi = \langle B,\ldots,Y,X\rangle$, where $Y \in V_j$, $0 \leq j \leq n$, and $u \in f_k(\pi,\beta,follow_k(A))$, then $[V_i,\alpha,X]$ will be replaced by $[V_i,\alpha]\,[V_j,\varepsilon,X]$.

c) if there is a production $A \rightarrow \alpha X \in P$, where $A \in V_i$, such that $u \in follow_k(A)$, then
   i) this production is printed
   ii) $[V_i,\alpha,X]$ is deleted. Now let $[V_h,\gamma]$ be the topmost stackelement.
   iii) $[V_h,\gamma]$ is extended to $[V_h,\gamma,A]$, if there is a production $C \rightarrow \gamma Y \delta$, $C \in V_h$, and a chain $\pi \in CH(Y)$ ending with A.

d) if $[V_i,\alpha,X] = [V_o,\Delta,S]$ and $u = \varepsilon$, then the inputword is accepted.

e) if none of the alternatives a) through b) applies, an errormessage is printed indicating that the inputword cannot be in L(G).

The definition of PC(k)-grammars guarantees that at most one of the alternatives a) through b) applies at a time.

Optimization of PC(k)-parsers

The above parsing-algorithm will obviously spend most of its time trying to find out, which of the above alternatives applies. As this decision merely depends on the topmost stack-element and the lookahead, it is possible to compute the decision and the

appropriate actions to be taken by the parser in advance for all possible topmost stackelements and all lookaheads and store them in a control-table. To determine its next parsing-action, a PC(k)-parser will then simply have to look up the entry for the present topmost stackelement and the present loookahead in the control-table. Using a control-table speeds-up the parser, it however at the same time creates a serious space-problem. The control-table will be very large if there are many different topmost stackelements. One way of optimizing a PC(k)-control-table will therefore be, to decrease the amount of different topmost stackelements. Without going into detail, two possibilities how to do so are stated here.

1. All topmost stackelements containing ε as their third component can be removed in the following manner: Instead of replacing $[V_i,\alpha,X]$ by $[V_i,\alpha X,\varepsilon]$ in alternative a), one replaces $[V_i,\alpha,X]$ by $[V_i,\alpha X,A]$ directly, where A is the predecessor of ε in the chain π. (On the basis of condition 2) the predecessor of ε in a chain can always be determined exactly by looking at the lookahead)
2. All topmost stackelements containing ε as their second component can be removed in the following manner: Instead of replacing $[V_i,\alpha,X]$ by $[V_i,\alpha][V_j,\varepsilon,X]$ in alternative b), one replaces $[V_i,\alpha,X]$ by $[V_i,\alpha]$ followed by the entry for $[V_j,\varepsilon,X]$ and the lookahead u in the control-table.

There are quite a few further methods for optimizing control-tables, wellknown from the literature, which, when applied to PC(k)-tables, will lead to considerable sizereductions too.

An efficient implementation of a PC(k)-parser will probably code the topmost stackelements into integers, to speed-up access to the control-table. In this case it will of course no longer be possible to determine the new topmost stackelement after a reduction in alternative c) step iii) simply by extending the momentary topmost stackelement $[V_h,\gamma]$ by the left-hand side of the production just reduced. A similar problem arises in alternative a) if $k=0$, because in this case

the next input-symbol will not be known in advance. These problems can be solved by introducing a goto-table, which, given some stackelement $[V_i,\alpha]$ (also coded by an integer) and some $X \in N$ ($X \in NUT$ if $k=0$), will tell the integer that stands for $[V_i,\alpha,X]$ (if $[V_i,\alpha,X]$ is a valid topmost stackelement).

Space-efficiency of a PC(k)-parser

Comparing the space required by a PC(k)-parser to the amount of space required by a LALR(k)-parser yields the following results:

1) PC(k)-parsers need a much smaller run-time stack than LALR(k)-parsers.
2) PC(k)-control-tables can be smaller than LALR(k)-control-tables, but there are examples for the contrary too.
3) PC(k)-goto-tables are smaller than LALR(k)-goto-tables for $k \leq 1$.

One further observation is important in this respect:
As far as grammars describing programming languages are concerned, the goto-tables tend to be bigger than the control-tables for $k \leq 1$. This is quite obvious for $k=0$. It also holds true for $k=1$, because grammars describing programming languages usually contain much more nonterminals than terminals.

Alltogether this leads to the conclusion that for $k=0$ and $k=1$ – the only cases of any practical relevance – PC(k)-parsers will generally use less space than the appropriate LALR(k)-parsers.

5. CONCLUSION

PC(k)-grammars prove to be well suited for parser-generators in that efficient parsers can be constructed for them. However this is also true for a number of other grammarclasses. The main argument in favour of using PC(k)-grammars instead of one of these other classes is, that, given a language, it is comparatively easy to construct a PC(k)-grammar for it.

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)

[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