SIMPLE CHAIN GRAMMARS

Anton Nijholt
Department of Mathematics
Vrije Universiteit, Amsterdam, The Netherlands.

ABSTRACT. A subclass of the LR(0)-grammars, the class of simple chain grammars is introduced. Although there exist simple chain grammars which are not LL(k) for any k, this new class of grammars is very close related to the class of LL(1) and simple LL(1) grammars. In fact it can be proved (not in this paper) that each simple chain grammar has an equivalent simple LL(1) grammar. A very simple (bottom-up) parsing method is provided. This method follows directly from the definition of a simple chain grammar and can easily be given in terms of the well-known LR(0) parsing method.

1. INTRODUCTION

In this paper we consider a subclass of the *LR(0)- grammars* which has some interesting properties. This class of grammars, called the *simple chain grammars*, has a very simple and natural bottom-up parsing method. Our definition of a simple chain grammar was motivated by the parsing method for *production prefix grammars* as introduced by Geller, Graham and Harrison [4]. However, they start with a method to construct a *parsing graph* for a context-free grammar and give conditions which should be satisfied in order that the parsing method works.
In our approach we start with a grammatical definition and as can be shown we can use a slightly adapted version of their parsing method. There is also a very strong and clear correspondence with the *LR(0) parsing method* [3].
This paper however is mainly concerned with properties of simple chain grammars and languages.
For the time being we consider only simple chain grammars for which no look-ahead is allowed. An extension with look-ahead seems to be straigtforward and is not considered here.

Besides the research reported in [4], work which is related to the ideas in this paper has been done by Lomet [14], Král and Demner [13], and Conway [2].

Preliminaries.

We assume the reader is familiar with the basic concepts of formal languages and automata theory [1]. Some of them are reviewed below for notational reasons.

A *context-free grammar* (cfg) is denoted by G= (N,T,P,S), V= NuT; elements of N (*nonterminals*) will be denoted by the Roman capitals A,...,T; elements of T (*terminals*) by the Roman lower case letters a,b,c,....; elements of V by the Roman capitals U,..., Z; in P are the (context-free) *productions* and S is the *start symbol*. Elements of T^* will be denoted by the Roman lower case letters u,v,w,x,y,z; elements of V^* by the Greek lower case letters $\alpha,\beta,\gamma,\delta,\ldots$. The *length* of $\alpha \in V^*$ is denoted by $|\alpha|$, the symbol ε is reserved for the *empty string*; if $\alpha \in V^+$ then $^{(1)}\alpha$ denotes the first symbol of α. The notation $\alpha \overset{*}{\underset{l}{\Longrightarrow}} \beta$ is used for a *leftmost derivation* of β from α; $\alpha \overset{*}{\underset{r}{\Longrightarrow}} \beta$ denotes a *rightmost derivation*.

DEFINITION 1.1. (cycle-free, ε-free, left-recursive)
A cfg G= (N,T,P,S) is said to be *cycle-free* if there is no derivation $A \overset{+}{\Longrightarrow} A$, for any $A \in N$. Cfg G is said to be *ε-free* if there are no productions of the form $A \to \varepsilon$ in P. A nonterminal A is said to be *left-recursive* if $A \overset{+}{\Longrightarrow} A\beta$ for some $\beta \in V^*$. A cfg is said to be left-recursive if G has at least one left-recursive nonterminal.

From now on we assume that all the context-free grammars in this paper are proper, i.e. reduced, cycle-free and ε-free.

DEFINITION 1.2. (LL, simple LL)

a. Let G= (N,T,P,S), $\delta \in V^*$, then
   FIRST (δ) = $\{a \in T \mid \delta \overset{*}{\Longrightarrow} a\phi$, for some $\phi \in V^*\}$.
b. G is said to be an *LL(1) grammar* if for every pair $A \to \alpha$ and $A \to \beta$ in P, if $\alpha \neq \beta$ then FIRST $(\alpha) \cap$ FIRST $(\beta) = \emptyset$.
c. G is said to be a *simple LL(1) grammar* if
   (i) every production is of the form $A \to a\phi$ $(a \in T, \phi \in V^*)$, and
   (ii) if $A \to a\phi$ and $A \to b\psi$ then $a \neq b$ or $a\phi = b\psi$.

Our definition of an LL(1) grammar slightly differs from the usual one which is caused by the fact that our grammars are proper. LL-grammars are not left-recursive; each simple LL(1) grammar is LL(1).

2. SIMPLE CHAIN GRAMMARS

In this section we introduce the class of simple chain grammars and we discuss some of their properties. First we need a few more definitions.

DEFINITION 2.1. (prefix-free)
Let G= (N,T,P,S) be a cfg and let A ∈ N. A is said to be *prefix-free* if $A \overset{*}{\Longrightarrow} w_1$ and $A \overset{*}{\Longrightarrow} w_1w_2$ implies $w_2 = \varepsilon$. A cfg is said to be prefix-free if all nonterminals are prefix-free. A language L is prefix-free if $w_1 \in L$ and $w_1w_2 \in L$ implies $w_2 = \varepsilon$.

Extension of the definition of prefix-free for a string $\alpha \in V^+$ is straightforward.

DEFINITION 2.2. (chain)
Let G = (N,T,P,S) be a cfg, let $X_0 \in V$. The set of *chains* of X_0, denoted by $CH(X_0)$, is defined by

$$CH(X_0) = \{ \langle X_0,X_1,\ldots,X_n\rangle \mid X_0X_1\ldots X_n \in N^*T \;\&\; X_0 \underset{\ell}{\Longrightarrow} X_1\psi_1 \underset{\ell}{\Longrightarrow} \ldots \underset{\ell}{\Longrightarrow} X_n\psi_n,\ \psi_i \in V^*,\ 1\le i\le n\}.$$

If $\pi = \langle X_0,X_1,\ldots,X_n\rangle$ then $l(\pi) = X_n$, that is, $l(\pi)$ denotes the last element of a chain π.

EXAMPLE 2.1. Consider a cfg G with only productions S→AF, A→Ba, B→Cd, C→dF, F→Ga, G→Cb C→dB', B'→b, F→a. For this cfg we have for example CH(C) = {<C,d>}, CH(a) = {<a>} and CH(F) = {<F,a>,<F,G,C,d>}

DEFINITION 2.3. (chain-independent)
Let G = (N,T,P,S), X ∈ V. X is said to be *chain-independent* if for each pair π_1,π_2, $\pi_1 \neq \pi_2$, in CH(X), we have $l(\pi_1) \neq l(\pi_2)$. If each element of V is chain-independent then V is said to be chain-independent.
Let X, Y ∈ V, X ≠ Y. X and Y are said to be *mutually* chain-independent if for each pair $\pi_1 \in CH(X)$ and $\pi_2 \in CH(Y)$, $\pi_1 \neq \pi_2$, we have $l(\pi_1) \neq l(\pi_2)$. Notation: $X \not\equiv Y$. This notation is also used if X = Y and X is chain-independent.

Observe that a left-recursive nonterminal cannot be chain-independent and that each terminal is chain-independent. Moreover, if X is chain-independent then $X \not\equiv X$. For each cfg in Greibach normal form V is chain-independent.
For the cfg of example 2.1. both A and F are chain-independent. However A and F are not mutually chain-independent.

We are now sufficiently prepared to give our definition of a simple chain grammar.

DEFINITION 2.4. (simple chain-grammar)
A cfg $G = (N,T,P,S)$ is said to be a *simple chain grammar* if it satisfies the following three conditions:

(i) V is chain-independent.

(ii) if $A \to \alpha X \phi$ and $A \to \alpha Y \psi$ are in P then $X \not\equiv Y$.

(iii) if $A \to \alpha$ and $A \to \alpha\beta$ are in P then $\beta = \varepsilon$.

One can easily verify that the cfg of example 2.1. is a simple chain grammar. We give another example.

EXAMPLE 2.2. Consider the cfg with only productions $S \to aEc$, $S \to aEd$, $E \to aE$ and $E \to ab$. Clearly V is chain-independent since $CH(S) = \{<S,a>\}$ and $CH(E) = \{<E,a>\}$. Moreover condition (ii) is satisfied since $E \not\equiv E$, $c \not\equiv d$ and $E \not\equiv b$. Also condition (iii) is satisfied. Notice however that this cfg is not LL(1), moreover, there is no k such that it is LL(k).

THEOREM 2.1. EVERY LL(1) GRAMMAR IS A SIMPLE CHAIN GRAMMAR.

Proof. We consider the three conditions of the definition of a simple chain grammar. Let $G = (N,T,P,S)$ be a (proper) LL(1) grammar.

a. Let $X_0 \in V$ and suppose X_0 is not chain-independent. Then there are at least two chains $\pi_1 = <X_0,X_1,\ldots,X_n>$ and $\pi_2 = <X_0,X_1^1,\ldots,X_m^1>$, $\pi_1 \neq \pi_2$, such that $X_n = X_m^1$. Hence $FIRST(X_1) \cap FIRST(X_1^1) \neq \emptyset$, which contradicts G being LL(1).

b. Let $A \to \alpha X \phi_1$ and $A \to \alpha Y \phi_2$ in P and assume we do not have $X \not\equiv Y$. Hence there are chains π_1 in CH(X) and π_2 in CH(Y), $l(\pi_1) = l(\pi_2)$ and $X \neq Y$ (since V is chain-independent). Therefore $\alpha X \phi_1 \neq \alpha Y \phi_2$ and $\alpha = \varepsilon$ since otherwise $FIRST(\alpha X \phi_1) \cap FIRST(\alpha Y \phi_2) \neq \emptyset$. However, also $FIRST(X\phi_1) \cap FIRST(Y\phi_2) \neq \emptyset$ since $l(\pi_1) = l(\pi_2)$.

c. Let $A \to \alpha$ and $A \to \alpha\beta$ be in P. If $\beta \neq \varepsilon$ then $\alpha \neq \alpha\beta$ and $FIRST(\alpha) \cap FIRST(\alpha\beta) \neq \emptyset$, hence G is not LL(1). Contradiction □.

The cfg of example 2.2. is a simple chain grammar and it is not LL(1). Therefore the LL(1) grammars are properly included in the class of simple chain grammars. Another example is the cfg with only productions $A \to aBc$, $A \to aCd$, $B \to b$ and $C \to c$. Before going to some general properties of simple chain grammars and languages we take a closer look at the set CH(X) for any $X \in V$. If a cfg G is in GNF (Greibach normal form) then each chain of the finite set CH(X) is of length 1. If G is not left-recursive then CH(X) is finite. In general CH(X) is a regular set, which can easily be verified by constructing a regular grammar G_X for any $X \in N$.

THEOREM 2.2. EVERY SIMPLE CHAIN GRAMMAR IS PREFIX-FREE.

Proof. We have to prove that every nonterminal of a simple chain grammar is prefix-free. Let $G = (N,T,P,S)$ be a simple chain grammar. First notice that for all $u,v \in T^*$ such that $A \overset{*}{\underset{r}{\Longrightarrow}} u$ and $A \overset{*}{\underset{r}{\Longrightarrow}} uv$ this implies $v = \varepsilon$ iff for all $\alpha,\beta \in V^*$ such that $A \overset{*}{\underset{r}{\Longrightarrow}} \alpha$ and $A \overset{*}{\underset{r}{\Longrightarrow}} \alpha\beta$ this implies $\beta = \varepsilon$. By induction on the length of derivations we prove that every finite string $\mu \in V^+$ is prefix-free.
Basis. If $\mu \underset{r}{\Longrightarrow} w_1$ and $\mu \underset{r}{\Longrightarrow} w_1w_2$ then there exist C, z_1 and z_2 such that

$$\mu = w'Cw'' \underset{r}{\Longrightarrow} w'z_1w'' = w_1, \text{ and}$$

$$\mu = w'Cw'' \underset{r}{\Longrightarrow} w'z_2w'' = w_1w_2.$$

Therefore $w'z_1w''$ is a prefix of $w'z_2w''$ and from this it follows that z_1 is a prefix of z_2, which contradicts condition (iii) of the definition of a simple chain grammar. Hence $w_2 = \varepsilon$.
Induction. Assume for all $\mu \in V^+$, and derivations $\mu \overset{*}{\underset{r}{\Longrightarrow}} w_1$, and $\mu \overset{*}{\underset{r}{\Longrightarrow}} w_1w_2$ with lengths less than n and m respectively, we have $w_2 = \varepsilon$. Now consider derivations $\mu \overset{*}{\underset{r}{\Longrightarrow}} w_1$ and $\mu \overset{*}{\underset{r}{\Longrightarrow}} w_1w_2$, with lengths n and m respectively. There exist C, ρ_1, X, Y, ϕ_1, ϕ_2, v_1 and v_2 such that $C \to \rho_1 X \phi_1$ and $C \to \rho_1 Y \phi_2$ are in P, $X \neq Y$, and

$$\mu \overset{*}{\underset{r}{\Longrightarrow}} \rho Cw' \underset{r}{\Longrightarrow} \rho\rho_1 X\phi_1 w' \overset{*}{\underset{r}{\Longrightarrow}} \rho\rho_1 Xv_1 w' \overset{*}{\underset{r}{\Longrightarrow}} w_1, \text{ and}$$

$$\mu \overset{*}{\underset{r}{\Longrightarrow}} \rho Cw' \underset{r}{\Longrightarrow} \rho\rho_1 Y\phi_2 w' \overset{*}{\underset{r}{\Longrightarrow}} \rho\rho_1 Yv_2 w' \overset{*}{\underset{r}{\Longrightarrow}} w_1w_2.$$

Since G is a simple chain grammar we have $X \neq Y$ and hence $\rho\rho_1 \neq \varepsilon$. Moreover, to obtain both w_1 and w_1w_2 there exist $w \neq \varepsilon$ and $\bar{w} \neq \varepsilon$ such that $\rho\rho_1 \overset{*}{\underset{r}{\Longrightarrow}} w\bar{w}$ and $\rho\rho_1 \overset{*}{\underset{r}{\Longrightarrow}} w$, both w and $w\bar{w}$ are prefixes of w_1, and both derivations are of length less than n and less than m. Since this contradicts the induction hypothesis we must conclude $w_2 = \varepsilon$. This concludes the proof that every $\mu \in V^+$, and hence every $A \in N$ is prefix-free. □

THEOREM 2.3. EVERY SIMPLE CHAIN GRAMMAR IS UNAMBIGUOUS.

Proof. Let $G = (N,T,P,S)$ be a simple chain grammar.
Suppose $S \overset{*}{\underset{r}{\Longrightarrow}} w$ by at least two different (rightmost) derivations. Then there are productions, say $A \to \rho X\phi_1$ and $A \to \rho Y\phi_2$, where $X \neq Y$, such that there exists w' in $L(\rho X\phi_1) \cap L(\rho Y\phi_2)$. Therefore there are two derivations

$$A \Longrightarrow \rho X\phi_1 \overset{*}{\underset{r}{\Longrightarrow}} w', \text{ and}$$

$$A \Longrightarrow \rho Y\phi_2 \overset{*}{\underset{r}{\Longrightarrow}} w'.$$

G is a simple chain grammar, hence $X \neq Y$ and we must conclude that ρ is not prefix-

free, which contradicts theorem 2.2. So the assumption that there are two such derivations must be false. □

A characteristic feature of simple chain grammars is mentioned in the following theorem.

THEOREM 2.4. LET G = (N,T,P,S) BE A SIMPLE CHAIN GRAMMAR AND SUPPOSE THERE ARE DERIVATIONS

$S \overset{n}{\underset{\ell}{\Longrightarrow}} wX\phi_1$ AND $S \overset{n}{\underset{\ell}{\Longrightarrow}} wY\phi_2$, WHERE $X \neq Y$, THEN $X \neq Y$.

Proof. The proof, which is omitted in this extended abstract, is by induction on the lengths of the derivations. □

DEFINITION 2.5. (LR(0) grammar)
The (proper) cfg G = (N,T,P,S) is said to be LR(0) iff:
for each w, w', $x \in T^*$; $\gamma,\alpha,\alpha',\beta,\beta' \in V^*$ and $A,A' \in N$, if
(i) $S \overset{*}{\underset{r}{\Longrightarrow}} \alpha Aw \underset{r}{\Longrightarrow} \alpha\beta w = \gamma w$, and
(ii) $S \overset{*}{\underset{r}{\Longrightarrow}} \alpha'A'x \underset{r}{\Longrightarrow} \alpha'\beta'x = \gamma w'$,
then $A \to \beta = A' \to \beta'$ and $|\alpha\beta| = |\alpha'\beta'|$.

An LR(0) grammar according to this definition does not necessarily generate a prefix-free language. For example, the cfg G with only productions S→Sb and S→a is LR(0) and L(G) is not prefix-free. G is not LR(0) according to the definition in Aho and Ullman [1], see also [5] and especially [6] in which a lot of definitions for LR(k) grammars are compared.

THEOREM 2.5. EVERY SIMPLE CHAIN GRAMMAR IS AN LR(0) GRAMMAR.
Proof. The proof, which starts by assuming that a cfg is a simple chain grammar and not LR(0), is omitted in this extended abstract. □

Observe that, since we are only concerned with ε-free grammars, the combination of theorems 2.1 and 2.5 does not load to the incorrect result that every LL(1) grammar (not necessarily ε-free) is an LR(0) grammar. Clearly every simple LL(1) grammar is a simple chain grammar. The class of simple chain grammars is properly included in the class of LR(0) grammars since the cfg with only productions S→aB|eB, B→cD|cF, D→b and F→b is LR(0) but it is not a simple chain grammar.

3. SIMPLE CHAIN LANGUAGES.

We list, without proofs, some properties of the languages generated by simple chain grammars. A cfg is in *Greibach normal form* (GNF) if each production is of the form A→aα, where $a \in T$ and $\alpha \in N^*$. If $\alpha \in V^*$ then we say that the cfg is in *pseudo*-GNF. Clearly, if a cfg G is in (pseudo-) GNF then V is chain-independent. Our results on simple chain languages are listed in the following corollary.

COROLLARY 3.1

a. Every simple chain grammar can be transformed to an equivalent simple chain grammar in GNF.
b. Each simple chain grammar can be transformed to an equivalent simple LL(1)-grammar (or s-grammar [12]).
c. The simple chain grammars generate exactly the class of simple deterministic languages.
d. It is decidable whether two simple chain grammars are equivalent.

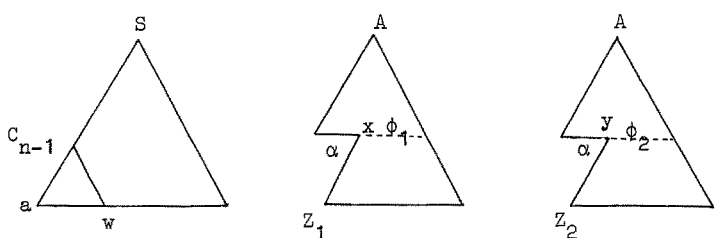
In this paper we do not consider the question whether the transformation to a simple LL(1)-grammar can be given in such a way that the new grammar *right-covers* the original grammar (see for definitions [1, p.276] and [7]).

4. THE PARSING OF SIMPLE CHAIN GRAMMARS.

Intuitively we can introduce the parsing method by considering the following two situations. The first one is a start-situation, the second is an arbitrary situation occuring later during the parsing process.

Let $a \in T$ in figure 1. There is only one chain π in $CH(S)$ with $l(\pi) = a$. Therefore the pair (S,a) determines chain π, and thus if $\pi = \langle S, C_1, \ldots, C_{n-1}, a \rangle$ then we know that a is a prefix of a right-hand side of a production with left-hand side C_{n-1}. This information should be held on a stack and we can enter a new situation.

Figure 1. Situations during the parsing process.



situation 1. situation 2.

For example, if we leave situation 1. then $A = C_{n-1}$ and $\alpha = a$ (if we assume that a is not the complete right-hand side). After having recognized α we want to recognize the remainder of the right-hand side $\alpha X \phi_1$ or $\alpha X \phi_2$. If $X \neq Y$ then, by condition (ii) of the definition of a simple chain grammar, $Z_1 \neq Z_2$, Hence, in this situation X (or Y) is uniquely determined. By condition (i) the chain from X to Z_1 is uniquely determined. So we know also the symbol to which the right-hand side with prefix Z_1 should be reduced. This information is also held on the stack which we will use.

(In case for example X is a terminal the appropriate chain will be <X> and αX is prefix of a right-hand side which should be reduced to A).
Condition (iii) of the definition of a simple chain grammar determines if the complete right-hand side of the production has been recognized and then reduction can take place, that is, an appropriate number of symbols will be popped from the stack and the production will be given as output.

The reader who is familiar with *strict-deterministic grammars* [9] and their parsing method [10] will have noticed some similarities. Elaboration of this will not be done here. The next step to a formal definition of the parsing method introduces the parsing graph.
The parsing method for simple chain grammars will turn out to be very simple. It is a modified version of the method for production prefix grammars a presented in [4], or if one wishes so, a modified version of the LR(0) parser (see e.g. DeRemer [3]).
From the informal discussion given above we can conclude that the parsing decisions can be made if we know the *configuration* (A,α), where $A\rightarrow\alpha\phi$ is a production and prefix α has already been recognized. These configurations will be the *nodes* of a *parsing graph* which controls a *pushdown stack* in which we store subsequent configurations of productions of which the right-hand sides have not yet been completely recognized. This is of course the same idea as for LR(0) parsers in which case each node of the parsing graph represents a configuration set of a more complex nature than in our case.

To be more precisely, and using the terminology of DeRemer [3],in our case each node (except the start node) represents a *basis set* which has only configurations of the form $A\rightarrow\alpha.\phi_i$, where i runs from 1 to the number of productions which have left-hand side A and prefix α ($\alpha \neq \varepsilon$), and its related *closure set*. This means that the pair (A,α) uniquely determines the configuration set and we can simply speak of the confiquration, or the node, (A,α). In an LR(0) parser each node can represent a configuration set such that a configuration $B\rightarrow\beta.\psi$, where $\beta \neq \varepsilon$ and $A\alpha \neq B\beta$, may also be contained in the basis set.

ALGORITHM 4.1. (parsing graph)

Input: A simple chain grammar $G = (N,T,P,S)$.

Output: A parsing graph for G.

Method: Each node of the parsing graph will correspond to a configuration. The start node is (S).

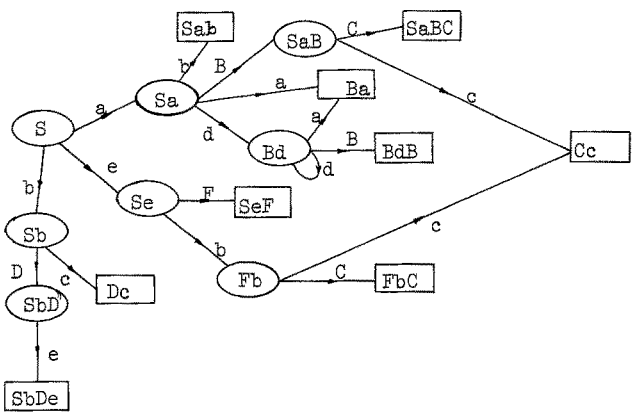
I. Let $A\rightarrow\gamma$ be in P. A configuration is denoted by $(A\gamma')$, where $\gamma' \neq \varepsilon$ and γ' is a prefix of γ. If $\gamma = \gamma'$ then the configuration is denoted by $[A\gamma]$.The corresponding nodes are in the form of a circle and a square respectively.

II. Let $(A\alpha)$ be a configuration, $\alpha \in V^*$. If $A \to \alpha X \phi$ is in P then $(A\alpha X)$ is a (basis) X-successor of $(A\alpha)$ and an edge with label X is drawn from node $(A\alpha)$ to node $(A\alpha X)$.

Furthermore, for all Y such that $X \overset{*}{\underset{\ell}{\Longrightarrow}} Y\delta \underset{\ell}{\Longrightarrow} Z\delta'$, for some δ and δ' in V^* and $Y \in N$ we have that (YZ) is an (closure) Z-successor of $(A\alpha)$, and an edge with label Z is drawn from node $(A\alpha)$ to node (YZ).□

This algorithm is illustrated with an example. In figure 2 we display the productions and the parsing graph of a simple chain grammar.

Figure 2. Parsing graph.



In the following algorithm we describe the parsing method.

ALGORITHM 4.2. (parsing algorithm)

Input: A parsing graph for a simple chain grammar $G = (N,T,P,S)$ and a string $w \in T^*$.

Output: If $w \in L(G)$ then a sequence of productions used in a rightmost derivation of w, in a reversed order. If $w \notin L(G)$ then an error is declared.

Method: We maintain a stack on which the (representations of the) configurations will be stored. We refer to the symbol on top of this pushdown stack as the *current state* . The start state is (S), which will be on top of the stack as the parsing starts. Observe that condition (iii) of definition 2.4. provides that the current state is either a read state, that is, of the form (...), or a reduce state, that is, of the form [...].

I. If the current state is a read state then read the next symbol of w and place the successor of this symbol on top of the stack. If there is no such successor then declare an error and halt.

II. If the current state is a reduce state then the number of symbols corresponding to the length of the righthand side of the production involved is popped from the

stack. The production is given as output. The successor of the lefthand side of this production for the current state (after popping the stack) is placed on the stack.□

Since this parsing algorithm is nothing more than the wellknown method in [3] the validity of our way of parsing simple chain grammars is guaranteed by the following observations on the parsing graph.

OBSERVATION 4.1.

a. Condition (i) and (ii) of definition 2.4. guarantee that for each element $Z \in V$ each node has at most one Z-successor. Otherwise, let X,Y and Z in V and $(A\alpha)$ is a node of the parsing graph. Suppose $A \to \alpha X \phi_1$ and $A \to \alpha Y \phi_2$ are two (possibly equal) productions. If $(A\alpha)$ has two Z-successors then we have one of the following two situations:

I. There exist $X_1, X_2 \in N$, $\psi_1, \psi_2, \delta_1, \delta_2 \in V^*$ such that
$X \overset{*}{\underset{\ell}{\Longrightarrow}} X_1\psi_1$ and $X_1 \to Z\delta_1$, and
$Y \overset{*}{\underset{\ell}{\Longrightarrow}} X_2\psi_2$, $X_1 \neq X_2$ and $X_2 \to Z\delta_2$.

II. $X = Z$ (or the symmetrix case $Y = Z$) and there exist $X_2 \in N$ and $\psi_2, \delta_2 \in V^*$ such that $Y \overset{*}{\underset{\ell}{\Longrightarrow}} X_2\psi_2$ and $X_2 \to Z\delta_2$ $(X_2 \neq Z)$.

That is, we have Z-successors (X_1Z) and (X_2Z) in case I or $(A\alpha Z)$ and (X_2Z) in case II. If $X = Y$ then case I is impossible since X is chain-independent. If $X \neq Y$ then case I is impossible since $X \not\equiv Y$. Also case II is contradicted by $X \not\equiv Y$.

b. Condition (iii) of definition 2.4. provides that each node denotes either a reduce or a read state.

c. Since there is no edge with label S leading away from (S), which is guaranteed by condition (i) of definition 2.4., the parsing properly terminates if a reduction to S has been made.□

The use of the algorithm is illustrated by parsing the sentence of the grammar in the example following algorithm 4.1. (figure 2). Without comment we display the contents of an input tape, the stack and the output which is emitted. In this table the top of the pushdown stack (the current state) is on the right-hand side.

input tape	stack	output
adac	(S)	-
dac	(S)(Sa)	-
ac	(S)(Sa)(Bd)	-
c	(S)(Sa)(Bd)[Ba]	-
c	(S)(Sa)(Bd)	B↦a
c	(S)(Sa)(Bd)[BdB]	-
c	(S)(Sa)	B↦dB
c	(S)(Sa)(SaB)	-
-	(S)(Sa)(SaB)[Cc]	-
-	(S)(Sa)(SaB)	C↦c
-	(S)(Sa)(SaB)[SaBC]	-
-	(S)	S↦aBC

Table I. Actions of the parser on adac.

5. CONCLUSIONS.

In this paper we introduced a proper subclass of the LR(0) grammars, the class of simple chain grammars. We showed that every simple chain grammar is prefix-free. The simple chain grammars generate exactly the class of simple deterministic (or simple LL(1)) languages. A parsing method, very close related to, and inspired by the method of production prefix parsing was introduced, and the relation to LR(0)-parsers was shown.

We want to spent some notes on, what are in our eyes, the most important features of simple chain grammars. In the first place we want to mention the possibility to transform each simple chain grammar to a simple LL(1) grammar [15]. What class of grammars is obtained after a similar transformation if we extend the definition of simple chain grammars with look-ahead? In the second place we have to mention the definition of simple chain grammars, which is entirely in terms of the finite sets of productions, nonterminals and terminals, instead of the (in general infinite) set of derivations. Moreover, the very simple parsing method follows directly from this definition and can be considered as a restricted way of LR(0)-parsing [3].

In the third place we have the following question. In [8] Hammer introduced a method to obtain LL(k) grammars from LR(k) grammars. On a much lower level we are doing something like that. As we show in [15] the simple chain grammars, which can be parsed using a bottom-up parsing method, can be transformed to a class of grammars (the simple LL(1) grammars) which have a top-down parsing method. Immediately from this we come to our fourth and last note on possible future work on the simple chain grammars.

We can ask what kind of covers are possible from simple chain grammars and their extensions to simple LL(1) and probably less restrictive classes of grammars. Although given in an informal way, in [8] the transformation and cover is such that right parses are mapped on left parses. In [15] we show that in general a left cover from simple chain grammars to simple LL(1) grammars is impossible. Therefore also the question of possible covers is interesting.
Of course we are aware of the fact that only a very restricted class of deterministic languages is generated by the class of simple chain grammars. We think however that extensions of the definition of simple chain grammar are possible, which preserve some of the appealing properties of simple chain grammars and their parsing method, and which remain rather simple.

REFERENCES

1. Aho A.V. and J.D. Ullman, 'The theory of parsing, translation and compiling', Vol.I and II, Prentice Hall, Englewood Cliffs, 1972 and 1973.
2. Conway M.E., Design of a seperable transition-diagram compiler, C.ACM 6,(1963), No.7. p.396-408.
3. DeRemer F.L., Simple LR(k) grammars, C.ACM 14, (1971), No.7, p.453-460.
4. Geller M.M., S.L.Graham and M.A.Harrison, Production prefix parsing, in 'Automata, Languages and Programming', J.Loeckx (ed.), 1974, Lecture Notes in Computer Science 14, Springer-Verlag, Berlin, p.232-241.
5. Geller M.M. and M.A.Harrison, Strict deterministic versus LR(0) parsing, Conf. Record of ACM Sympos.on Principles of programming languages, Boston, Massachusetts, 1973, oct 1-3, p.22-32.
6. Geller M.M. and M.A.Harrison, On LR(k) grammars and languages, manuscript.
7. Gray J. and M.A.Harrison, On the covering and reduction problems for context-free grammars, J.Assoc.Comput. Mach.19, (1972), No.3, p.385-395.
8. Hammer M., A new grammatical transformation into LL(k) form, Conf.Record of 6th Ann. ACM Sympos. on Theory of Computing, 1974, p.266-275.
9. Harrison M.A. and I.M.Havel, Strict deterministic grammars, J.Comput.System Sci. 7, (1973), No.3, p.237-277.
10. Harrison M.A. and I.M. Havel, On the parsing of deterministic languages, J.Assoc. Comput.Mach.21, (1974), No.4, p.525-548.
11. Knuth D.E., On the translation of languages from left to right, Info. and Control 8, (1965), No.6, p.607-639.
12. Korenjak A.J. and J.E. Hopcroft, Simple deterministic languages, IEEE Conf.Record of 7th Annual Sympos. on Switching and Automata Theory, 1966, p.34-46.
13. Král J. and J.Demner, Parsing as a subtask of compiling, Sympos. on Mathematical Foundations of Computer Science, 4th, Mariánské Lázně, 1975, Lecture Notes in

Computer Science 32, Springer-Verlag, Berlin, p.61-74.

14. Lomet D.B., Automatic generation of multiple exit parsing subroutines, in 'Automata, Languages and Programming', J.Loeckx (ed.), 1974, Lecture Notes in Computer Science 14, Springer-Verlag, Berlin, p.214-231.

15. Nijholt A., Simple chain languages, manuscript, march 1977.

Acknowledgements.

I am grateful to prof.L.A.M. Verbeek for his comments on an earlier version of this paper. I thank ms. Carla Reuvecamp for typing this paper.