ANTWERP PAPERS IN LINGUISTICS

nr. 5. 1976

ON THE FORMAL PROPERTIES OF COMPLETION GRAMMARS AND THEIR RELATED AUTOMATA .

Luc Steels & Dirk Vermeir

UNIVERSITEIT ANTWERPEN

Universitaire Instelling Antwerpen

Departementen Ger. en Rom. Afdeling Linguïstiek

Universiteitsplein, 1 -B-2610 Wilrijk(Antwerpen) -TEL 031/28.25.28

Abstract

Completion grammars are a new class of rewriting systems designed to model case systems. In this paper we investigate some formal properties of these grammars and introduce a related class of automata. Also it will be shown that by an extension of the systems, it is possible to deal with weaker precedence relations. In this context an effectively computable measure for the degree of grammaticality is introduced. The paper concludes with a short discussion on the way in which the grammars are applied to (natural) language analysis(and synthesis).

CONTENTS

PREFACE

The model of completion grammars arose from research on the formulation of exact grammars for natural languages. Completion grammars are rewriting systems, but differ from phrase structure grammars in that functional information , in particular which case relations hold, forms the underlying linguistic viewpoint.

In a first section we give the basic definitions of completion grammars and the languages they generate. These definitions differ formally (but not substantially) from earlier literature on completion grammar. (E.g. Steels (1975a).
In a second section we investigate some of the formal properties, in particular the relation to the Chomsky hierarchy, as regards the weak generative capacity. Then the strong generative capacity will be discussed by defining the structure assigned to the grammars and by considering some interesting consequences of the splitting up of the final alphabet.

In the third section we introduce completion automata and proof their equivalence with completion grammars. Completion automata have two stacks and a finite control. In a fourth section we extend the notion of grammars and automata such that the strict precedence order imposed by the generation relations is weakened to such an extent that all possible combinations of a strict grammatical string are accepted (or generated) as well. In this context a degree of ungrammaticality and a degree of complexity is being introduced.

A final chapter deals with the linguistic intuitions about natural language functioning that formed the basis of the formal models. The linguistically oriented reader should perhaps first read this.

Most proofs use standard techniques of formal language theory and are as is usual in the field not produced in full. This is done to keep the main flow of thought clear.

Acknowledgement

We thank the members of the reading committee of the'Antwerp papers in linguistics' who accepted the paper for publication in the series.

§ 1 BASIC DEFINITIONS

Definition 1.

A *completion grammar* is a 6-tuple $G = \langle Va, Vp, P, AX, Vta, K \rangle$ where

1) Va, Vp are two finite sets called the set of *arguments* and the set of *procedure names or predicates*
2) $Va \cap Vp = \emptyset$; $V = Va \cup Vp$
3) P is a finite subset of $Va \times VpVa^{\star}$, the set of *productions*
4) $AX \subseteq Va$ is the *axiomset*
5) $Vta \subseteq Va$ is the set of *terminal arguments*
6) $K: P \rightarrow \{$d(epending) , n(ondepending), i(ndifferent$\}$ is a mapping

A production in P is denoted as p: $a \xrightarrow{x} A\,\sigma$ where $a \in Va$, $A \in Vp, \sigma \in Va^{\star}$, $x \in K(p)$. It is customary to omit K(p) if K(p) = i.

CG denotes the set of all completion grammars.

Definition 2.

1) The relation $\Rrightarrow$, i.e. *direct closed derivation* is defined as

$$(\forall x,y)_{V^{\star}} (x \Rrightarrow y) \quad \text{iff} \quad (\exists a)_{Va} \quad (\exists A)_{Vp}:$$
$$(x = x_1\, a\, x_2 \;,\; a \xrightarrow{n \text{ or } i} A\,\sigma \quad P \;,\; y = x_1 A\, \sigma x_2 \;)$$

2) The relation $\Longrightarrow$,i.e. *direct open derivation* is defined as

$$(\forall x,y)_{V^{\star}}(x \Longrightarrow y) \quad \text{iff} \quad (\exists a)_{Va} \quad (\exists A)_{Vp} \quad :$$
$$(x = x_1 a x_2 \;,\; a \xrightarrow{d \text{ or } i} A\; a_1\; \theta \quad P \;,\; y = x_1 a_1 A\, \theta x_2 \;)$$

(Remark: it may happen that $a_1 = \theta = \lambda$)

If $a = a_1$, then $\Longrightarrow$ is a *strict direct open derivation*

3) Let $\overset{\star}{\Rrightarrow}$ (called *closed derivation*) and $\overset{\star}{\Longrightarrow}$ (called *open derivation*) denote the reflexive and transitive closure of $\Rrightarrow$ and $\Longrightarrow$ respectively.

Let $\Rightarrow \; = \; \Rrightarrow \; \cup \; \Longrightarrow$ and let $\overset{\star}{\Rightarrow}$ be the reflexive and transitive closure of $\Rightarrow$.

Definition 3.

Let $G = \langle Va, Vp, P, AX, Vta, K \rangle \in CG$

1) The language of G, denoted as L(G) is defined by

$$L(G) = \{x \mid (\exists a)_{AX} : a \stackrel{*}{\Rightarrow} x,\ x \in (Vta \cup Vp)^*\}$$

We call $\mathcal{L}_{CG} = \{L(G) \mid G \in CG\}$

2) We say that G is a closed completion grammar iff

$(\forall p)_P\ (K(p) = \text{nondepending})$

CCG denotes the class of all closed completion grammars and

$\mathcal{L}_{CCG} = \{L(G) \mid G \in CCG\}$

3) We say that G is an open completion grammar iff

$(\forall p)_P\ (K(p) = \text{depending})$

OCG denotes the class of all open completion grammars

$\mathcal{L}_{OCG} = \{L(G) \mid G \in OCG\}$

Example 1.

Let $G = \langle Va, Vp, P, AX, Vta, K \rangle$ be a closed completion grammar with $Va = \{a,b\}$, $Vp = \{A\}$, $Vta = \{b\}$, $AX = \{a\}$ and P:

1. $a \xrightarrow{n} A\ b$
2. $a \xrightarrow{n} A\ a\ b$

Some derivations (the index is the applied production)

(i) $a \stackrel{1}{\Rightarrow} A\ b$

(ii) $a \stackrel{2}{\Rightarrow} A\ a\ b \stackrel{2}{\Rightarrow} A\ A\ a\ b\ b \stackrel{1}{\Rightarrow} A\ A\ A\ b\ b\ b$

Note that G generates the famous context-free language $\{A^n b^n : n \geq 1\}$

Example 2.

Let $G = \langle Va, Vp, P, AX, Vta, K \rangle$ be an open completion grammar with $Va = \{a,b\}$, $Vp = \{A\}$, $Vta = \{b\}$, $AX = \{a\}$ and P:

1. $a \xrightarrow{d} A\ b$
2. $a \xrightarrow{d} A\ a\ b$

Some derivations :

(i) $a \stackrel{1}{\Rightarrow} b\ A$

(ii) $a \stackrel{2}{\Rightarrow} a\ A\ b \stackrel{2}{\Rightarrow} a\ A\ b\ A\ b \stackrel{1}{\Rightarrow} b\ A\ A\ b\ A\ b$

Now the language is $\{(b\ A)\ (A\ b)^n : n \geq 0\}$

Example 3.

Let $G = \langle \{a\}, \{A\}, \{1.\ a \rightarrow A,\ 2.\ a \rightarrow A\ a\ a\}, \{a\}, \emptyset \rangle \in CG$

Some derivations

(i) $a \overset{2}{\Longrightarrow} a\ A\ a \overset{1}{\Longrightarrow} A\ A\ a \overset{1}{\Longrightarrow} A\ A\ A$

(ii) $a \overset{1}{\Longrightarrow} A$

(iii) $a \overset{2}{\longrightarrow} A\ a\ a \overset{1}{\longrightarrow} A\ A\ a \overset{1}{\longrightarrow} A\ A\ A$

It is clear that $L(G) = \{A^{2n+1} : n \geqslant 0\}$

Example 4.

Let $G = \langle \{a,b\}, \{A\}, \{1.\ a \longrightarrow A\ b\ a,\ 2.\ a \longrightarrow A\ b,\ 3.\ b \longrightarrow A\}, \{a\}, \emptyset \rangle$ $\in CG$

Some derivations:

(i) $a \overset{1}{\Longrightarrow} b\ A\ a \overset{3}{\longrightarrow} A\ A\ a \overset{2}{\longrightarrow} A\ A\ A\ b \overset{3}{\Longrightarrow} A\ A\ A\ A$

(ii) $a \overset{2}{\longrightarrow} A\ b \overset{3}{\Longrightarrow} A\ A$

Obviously $L(G) = \{A^{2n} : n \geqslant 1\}$

Intuitively nondepending productions define predicates in prefix-position whereas depending productions define predicates in infix-position. This becomes clear by the following example:

Example 5.

Let $G = \langle Va, Vp, P, AX, Vta, K \rangle$ be a completion grammar with $Va = \{log\}$ $Vp = \{AND, OR, IMPLIES, NOT\}$, $AX = \{log\}$, $Vta = \{log\}$ and P:

1. log $\rightarrow$ AND log log
2. log $\rightarrow$ OR log log
3. log $\rightarrow$ IMPLIES log log
4. log $\overset{n}{\rightarrow}$ NOT log

Where 'log' stands for 'being a logical variable'.

Some derivations:

(a) closed:

log $\overset{1}{\longrightarrow}$ AND log log $\overset{2}{\longrightarrow}$ AND OR log log log $\overset{4}{\longrightarrow}$ AND OR log NOT log log

(expressions in prefix-notation)

(b) open:

log $\overset{1}{\Longrightarrow}$ log AND log $\overset{2}{\Longrightarrow}$ log OR log AND log $\overset{4}{\Longrightarrow}$ log OR log AND NOT log

(expressions in infix-notation)

Note that with an open derivation for production 4, 'not' would be standing after the variable it is negating. It seems therefore that NOT is always occurring in a nondepending production.

Example 6.

Let G $= \langle \{numb\}, \{+, \times, -, /\}, P, \{numb\}, \{numb\}, K \rangle \in CG$

P:

numb ⟶ + numb numb
numb ⟶ x numb numb
numb ⟶ - numb numb
numb ⟶ / numb numb

Some derivations:

(i) prefix

numb ⟹ + numb numb ⟹ + numb / numb numb

(ii) infix

numb ⟹ numb + numb ⟹ numb + numb - numb

§ 2. SOME FORMAL PROPERTIES

2.1. *Weak generative capacity*

Lemma 1. $\mathcal{L}_{CG} \subset \mathcal{L}_{CF}$

Proof:

Let G = $\langle Va, Vp, P, AX, Vta, K \rangle \in CG$. Define $\overline{G} = \langle Vn\ Vt, \overline{P}, S \rangle \in CF$ where

1) $Vn = \{\overline{a} \mid a \in Va\} \cup \{S\}$ where S is a new symbol
2) $Vt = Vta \cup Vp$
3) $\overline{P} = \{\overline{a} \rightarrow \phi_1(\sigma) \mid a \xrightarrow{n\ or\ i} \sigma \in P\} \cup$

$\{\overline{a} \rightarrow a \mid a \in Vta\} \cup$

$\{S \rightarrow \overline{a} \mid a \in AX\} \cup$

$\{\overline{a} \rightarrow \phi_2(\sigma) \mid a \xrightarrow{d\ or\ i} \sigma \in P\}$

where ϕ_1, ϕ_2 are mappings defined by

$\phi_1 : (Vn \cup Vt)^{\star} \rightarrow (Vn \cup Vt)^{\star}$

$a \rightarrow \overline{a} \quad \forall a \in Va$

$b \rightarrow b \quad \forall b \notin Va$

$\phi_2 : VpVa^{\star} \rightarrow Vn^{\star}$

$Aa_1\sigma \rightarrow \overline{a_1}\ A\ \phi_1(\sigma)$

It follows then that $L_{CG}(G) = L(\overline{G})$

□

Example 7.

Let G = $\langle \{a,b\}, \{A\}, \{a \xrightarrow{n} A\ a\ b,\ a \xrightarrow{n} A\ b\}, \{a\}, \{b\} \rangle \in CG$

By applying the construction of the lemma we obtain $\overline{G} = \langle Vn, Vt, \overline{P}, S \rangle$ where

$Vn = \{\overline{a}, \overline{b}, S\}$
$Vt = \{A, b\}$
$P = \{\overline{a} \rightarrow A\ \overline{a}\ \overline{b},\ \overline{a} \rightarrow A\ \overline{b},\ \overline{b} \rightarrow b,\ S \rightarrow \overline{a}\}$

Some derivations:

(i) $S \Longrightarrow \overline{a} \Longrightarrow A\ \overline{b} \Longrightarrow A\ b$

(ii) $S \Longrightarrow \overline{a} \Longrightarrow A\ \overline{a}\ \overline{b} \Longrightarrow A\ A\ \overline{a}\ \overline{b}\ \overline{b} \Longrightarrow A\ A\ A\ \overline{b}\ \overline{b}\ \overline{b} \Longrightarrow A\ A\ A\ b\ \overline{b}\ \overline{b}$

$\Longrightarrow A\ A\ A\ b\ b\ \overline{b} \Longrightarrow A\ A\ A\ b\ b\ b$

Example 8.

Let G $= \langle \{a,b\}, \{A\}, \{a \xrightarrow{d} A\ a\ b,\quad a \xrightarrow{d} A\ b\}, \{a\}, \{b\} \rangle$

By applying the construction of the lemma we obtain $\overline{G} = \langle Vn, Vt, \overline{P}, S \rangle$ where

$Vn = \{\overline{a}, \overline{b}, S\}$

$Vt = \{A, b\}$

$P = \{\overline{a} \longrightarrow \overline{a}\ A\ \overline{b},\ \overline{a} \longrightarrow \overline{b}\ A,\ \overline{b} \longrightarrow b,\ S \longrightarrow \overline{a}\}$

Some derivations:

(i) $S \Longrightarrow \overline{a} \Longrightarrow \overline{b}\ A \Longrightarrow b\ A$

(ii) $S \Longrightarrow \overline{a} \Longrightarrow \overline{a}\ A\ \overline{b} \Longrightarrow \overline{a}\ A\ \overline{b}\ A\ \overline{b} \Longrightarrow \overline{b}\ A\ A\ \overline{b}\ A\ \overline{b}$

$\Longrightarrow b\ A\ A\ \overline{b}\ A\ \overline{b} \Longrightarrow b\ A\ A\ b\ A\ \overline{b} \Longrightarrow b\ A\ A\ b\ A\ b$

The reader should compare example 7 and 8 with 1 and 2 respectively.

Lemma 2. $\mathcal{L}_{CF} \subset \mathcal{L}_{CCG}$

Proof:

Let $G = \langle Vn, Vt, P, S \rangle \in CF$. We may assume (see Salomaa, 1973) that G is in Greibach normal form, i.e. every production in P is of the form:

$A \rightarrow a\,\sigma$ where $\sigma \in Vn^{*}$ and $a \in Vt$.

To construct an equivalent CCG, we proceed as follows:

1) $Vta = \emptyset$

2) $Va = Vn$

3) $Vp = Vt$

4) $(\forall\ p \in P)\ (K(p) = n)$

Let $H = \langle Va, Vp, P, \{S\}, \emptyset \rangle \in CCG$, then clearly $L(G) = L(H)$. The proof by induction on the number of steps in a derivation is left to the reader.

□

Example 9.

Let $G = \langle \{S, B\}, \{a, b\}, \{S \rightarrow a\ S\ B,\ S \rightarrow a\ B,\ B \rightarrow b\}, S \rangle$

(note that G is already in Greibach normal form)

By applying the construction of the lemma we obtain:

1) $Vta = \emptyset$

2) $Va = \{S, B\}$

3) $V_p = \{a, b\}$

4) P : $S \xrightarrow{h} a\,S\,B$

$S \xrightarrow{h} a\,B$

$B \xrightarrow{h} b$

$H = \langle Va, Vp, P, \{S\}, \emptyset, K \rangle$ and clearly $L(H) = L(G) = \{a^n b^n \mid n \geqslant 1\}$

Lemma 3. $\mathcal{L}_{CF} \subset \mathcal{L}_{UCG}$

Proof:

For this lemma we need a somewhat different version of Greibach normal form, in order to obtain this normal form we first proof a sublemma.

Sublemma: $(\forall G = \langle Vn, Vt, P, S \rangle \in CF)$ $\exists G' = \langle Vn, Vt, \overline{P}, S \rangle \in CF$ such that every production in $\overline{P}$ is of the form:

1) $A \to a\, b\sigma$ where $\sigma \in Vn^{\star}$, $a, b \in Vt$

2) $A \to a$

Proof of the sublemma:

We may assume G to be in Greibach normal form, i.e. every production is of the form

1) $A \to a\, B\sigma$ where $a \in Vt$, $A, B \in Vn$, $\sigma \in Vn^{\star}$

2) $A \to a$

Define $\overline{P}$ as follows:

$A \to x \in \overline{P}$: iff $(A \overset{\ell}{\Rightarrow} y \overset{\ell}{\Rightarrow} x)$ or ($x = a$ and $A \to a \in P$)

(the index ℓ denotes the leftmost direct derivation.

Clearly $L(G') = L(G)$ and the sublemma holds.

Next define

$$P' = \{A \to X\, b\sigma\ ,\ X \to a \mid A \to a\, b\sigma \in \overline{P}\} \cup$$
$$\{A \to a \quad A \to a \in \overline{P}\}$$

where for each $A \to a\, b\beta \in \overline{P}$, X is a new symbol.

Now, given an arbitrary grammar G, we use the preceeding construction to obtain G' = ⟨Vn', Vt, P', S⟩ where L(G) = L(G') and every production in G' is of one of the following forms:

(i) $A \rightarrow X\, b\, \sigma$, $A \in Vn$, $X \in Vn$, $b \in Vt$, $\sigma \in Vn^{\star}$

or

(ii) $A \rightarrow a$, $a \in Vt$

Define $H = \langle Va, Vp, P_h, AX, Vta, K \rangle \in OCG$ as follows:

1) Vp = Vt
2) Va = Vn'
3) $AX = \{S\}$
4) $Vta = \emptyset$
5) $P_h = \{A \rightarrow \sigma \mid A \rightarrow X\, b\, \sigma \in P' \text{ where } b \in Vt,\ \sigma \in Vn^{\star}\} \cup \{A \rightarrow a \mid A \rightarrow a \in P'\}$
6) $(\forall\, p \in P)\ (K(p) = d)$

From the definitions it now follows that L(H) = L(G') = L(G)

□

Example 10.

Let $G = \langle \{S\}, \{a,b\}, \{S \rightarrow a\, S\, b,\ S \rightarrow a\, b\}, S \rangle \in CF$.

First we construct a grammar in Greibach normal form :

$G' = \langle \{S, B\}, \{a, b\}, \{S \rightarrow a\, S\, B,\ S \rightarrow a\, B,\ B \rightarrow b\}, S \rangle$,

then we construct a new grammar G" according to the construction of the sublemma:

$G'' = \langle \{S,B,A\}, \{a,b\}, P'', S \rangle$ where $\overline{P}$ contains the following productions:

S → a a S B B
S → a a B B
S → a b
B → b

from this we construct P":

S → A a S B B
S → A a B B
S → A b
B → b
A → a

1) $Vp = \{a,b\}$
2) $Va = \{S, B, A\}$
3) $AX = \{S\}$
4) $Vta = \emptyset$
5) $P_h = \{S \xrightarrow{d} a\, A\, S\, B\, B,\ S \xrightarrow{d} a\, A\, B\, B,\ S \xrightarrow{d} b\, A,\ B \xrightarrow{d} b,\ A \xrightarrow{d} a\}$

Let H = ⟨Va, Vp, P_h, AX, Vta, K⟩, then clearly L(G) = L(G') = L(G") = L(H) = $\{a^n b^n \mid n \geqslant 1\}$

Some derivations:

(i) $S \Longrightarrow A\ b \Longrightarrow a\ b$

(ii) $S \Longrightarrow A\ a\ S\ B\ B \Longrightarrow a\ a\ S\ B\ B \Longrightarrow a\ a\ A\ b\ B\ B \Longrightarrow a\ a\ a\ b\ B\ B$

$\Longrightarrow a\ a\ a\ b\ b\ B \Longrightarrow a\ a\ a\ b\ b\ b$

Theorem 1. $\mathcal{L}_{OCG} = \mathcal{L}_{CCG} = \mathcal{L}_{CG} = \mathcal{L}_{CF}$

Proof:

This is an immediate consequence of lemma 1, lemma 2, lemma 3.

□

For strict open completion grammars (SOCG) the situation is somewhat different.

Lemma 4.

Let G = ⟨Va, Vp, P, AX, Vta, K⟩ ∈ SOCG then
$\forall\, w \in L(G)$: $w = a\,\overline{w} \in Vta\ V^+$ and $(\forall\, n)_{\mathbb{N}_0}\ a\,\overline{w}^n \in L(G)$

Proof:

Suppose $w \in L(G)$. This implies that $pref_1(w) \in AX \cap Vta$ since

$$(\forall\, a)_{Vt} : (a \overset{*}{\Longrightarrow} x) \supset (pref_1(x) = a) \qquad (1)$$

is easily seen to be true.

From (1) it also follows that, if $pref_1(w) = a$, then $a \underset{\tau}{\overset{*}{\Longrightarrow}} a\,\overline{w} = w$ for some derivation and thus

$$a\,\overline{w} \overset{*}{\Longrightarrow} a\,\overline{w}\,\overline{w} \underset{\tau}{\overset{*}{\Longrightarrow}} a\,\overline{w}^3 \underset{\tau}{\overset{*}{\Longrightarrow}} \dots \underset{\tau}{\overset{*}{\Longrightarrow}} a\,\overline{w}^n \underset{\tau}{\overset{*}{\Longrightarrow}} \dots$$

□

Lemma 5. $\{a^n b^n \mid n \in \mathbb{N}\} \notin \mathcal{L}_{SOCG}$ and consequently $\mathcal{L}_{CF} \setminus \mathcal{L}_{SOCG} \neq \emptyset$

Proof:

This is an easy consequence of lemma 4. □

Lemma 6. $\{a^{2n}b \mid n \in \mathbb{N}\} \notin \mathcal{L}_{SOCG}$ and consequently $\mathcal{L}_{REG} \setminus \mathcal{L}_{SOCG} \neq \emptyset$

Proof:

Again this follows from lemma 4.

□

Lemma 7. $\exists L \in \mathcal{L}_{SOCG} \setminus \mathcal{L}_{REG}$

Proof:

Let $G = \langle \{a\}, \{B\}, \{a \rightarrow B\ a\ a\ a\}, \{a\}, \{a\}, K(p) = d \rangle \in$ SOCG

It should be clear that

$w \in L(G) \Rightarrow$ 1) $pref_1(w) = a$

2) $\#_a(w) = 2\ \#_B(w) + 1$

And also that

$(\forall n \in \mathbb{N})\ (\exists w = w_1 a^m w_2 \in L(G)$ where $m \geq n\}$

Suppose $L(G) \in \mathcal{L}_{REG}$ then:

$\exists$ dfa $\mathcal{Q} = \langle Q, \{a,B\}, \delta, q_o, F\rangle$ such that $L(\mathcal{Q}) = L(G)$

Let $w \in L(G) : w = w_1 a^n w_2$ with $n \geq 2(\#Q)$

Then clearly becuase by our assumption that $L(\mathcal{Q}) = L(G)$,

$(\exists q)_Q: (\exists v_1 = w_1 a^r):\ \hat{\delta}(q_o, v_1) = q$ and

$(\exists m)_{\mathbb{N}_o}(\delta(q, a^m) = q)$ $m \leq \#Q$

such that $v_1 a^{r+m} \in Pref(w)$

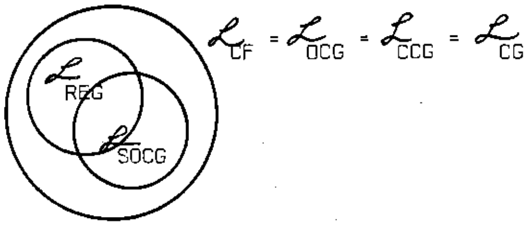
But this would imply that $w_1 a^{r+km} w_2 \in L(\mathcal{Q})(\forall K)_{\mathbb{N}}, K>0$. Since $w_1 a^{r+km} w_2 \notin L(G)$ (because of 2) this leads to a contradiction. Conclusion: $L(G) \in \mathcal{L}_{REG}$ □

Theorem 2. $\mathcal{L}_{SOCG} \subsetneq \mathcal{L}_{CF}$

$\mathcal{L}_{SOCG}$ and $\mathcal{L}_{REG}$ are incomparable

Proof: This follows from the previous lemmas. □

The results of theorem 1 and 2 are symbolized in the following diagram.



2.2. *Strong generative capacity*

The fact that the same type of language is generated by completion grammars and context-free grammars is an important and interesting result, this does not mean however that the way in which these grammars deal with language is the same.

In this section we define the structures assigned by completion grammars and discuss some consequences of the subdivision of the terminal alphabet.

2.2.1. Relation structures

Definition 4.

Let G = ⟨Va, Vp, P, AX, Vta, K⟩ ∈ CG then there corresponds with each derivation a unique graph called the *relation structure* R, where a relation structure is a labelled plane rooted graph to be constructed as follows:

(i) if $x \Rightarrow y$ holds, i.e. if $x = x_1 a\, x_2$, $a \xrightarrow{n \text{ or } i} A\alpha \in P$ and $y = x_1 A \alpha x_2$ with $\alpha = a_1 \dots a_n$, then nodes for A, $a_1, \dots, a_n$ are added to the structure and a directed line from A to a and from $a_1, \dots, a_n$ to A.

(ii) Similarly, if $x \Rrightarrow y$ holds, i.e. if $x = x_1 a x_2$, $a \xrightarrow{d \text{ or } i} A\ a_1 \alpha \in P$ $y = x_1 a_1 A \alpha x_2$, with $\alpha = a_2 \dots a_n$, then nodes for a_1, A, $a_2, \dots, a_n$ are added to the structure and a directed line from A to a and from $a_1, \dots, a_n$ to A.

(iii) This construction process is easily extended to the reflexive and transitive closure of $\Rightarrow$ and $\Rrightarrow$ respectively.

Clearly for an arbitrary x ∈ L(G) there corresponds a relation structure R_x with a the root of R_x for $a \overset{*}{\Rrightarrow} x$

Notation: For the sake of clarity we draw circles around each label denoting an element of Vp and squares around each label denoting an element of Va.

Example 11.

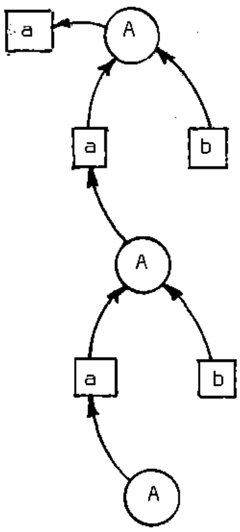
Let G = ⟨Va, Vp, P, AX, Vta, K⟩ ∈ CG with Va = {a,b} , Vp = {A}
AX = {a} , Vta = {b} and

P: a → A b
   a → A a b

then with the derivation

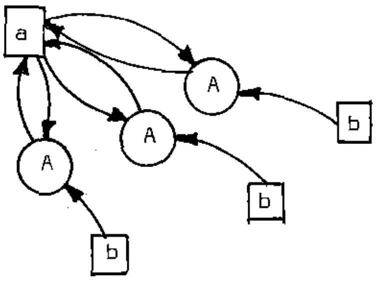
$$a \Rightarrow A\ a\ b \Rightarrow A\ A\ a\ b\ b \Rightarrow A\ A\ A\ b\ b\ b$$

corresponds the following relation structure:



and with the derivation

a $\Longrightarrow$ a A b $\Longrightarrow$ a A b A b $\Longrightarrow$ b A A b A b   corresponds the relation structure·



Remark:

Relation structures differ clearly from constituent structure trees in that

(i) they are graphs and not trees

(ii) lines can enter terminal elements

(iii) a functional relation among the elements is expressed and not a dominance relation. (More about this in the final section)

In other words completion grammars express other structural information than context-free grammars but they deal with the same sort of languages.

The most important deviation from phrase structure grammars is the splitting up of the alphabet in arguments and procedures. The consequences of having two disjoint sets as terminal alphabet can however not be studied in relation to phrase structure grammars (because the distinction does not exist in this system). The reader will remember from lemma 2 and 3 that in our construction process we defined each time Vta to be empty. That this distinction has however deep consequences will be clear

from the following lemma's.
First we extend the notion of a CF grammar to such an extent that discussion on the relation of CF grammars and CG grammars becomes meaningful.

2.2.2. Consequences of dividing the alphabet

Definition 5.

A pa grammar (denoted as PACF) is a 5-tuple $G = \langle Vn, Vta, Vtp, P, S \rangle$ where $\overline{G} = \langle Vn, Vta \cup Vp, P, S \rangle \in CF$ and $Vta \cap Vtp = \emptyset$

The pa-cf language of a PACF grammar is defined by:

$$L_{pa}(G) = \langle L(\overline{G}), Vta, Vtp \rangle$$

In the sequel symbols in Vtp will be denoted by capital letters A, B, ... symbols in Vta by small letters a,b,... .

Definition 6.

Let G, G' $\in$ PACF we say that G and G' are PA-equivalent ($G \underset{PA}{\equiv} G'$) and also that $L_{pa}(G) \underset{PA}{\equiv} L_{pa}(G')$ iff the following holds:

$$\exists \phi_a, \phi_p \text{ isomorphisms: } \quad \phi_a : Vta \rightarrow Vta' \qquad \phi_p : Vtp \rightarrow Vtp'$$

such that $\phi(L(\overline{G})) = L(\overline{G'})$ where $\phi = \phi_a \cup \phi_p$

Definition 7.

$$\mathcal{L}_{pa} CF = \{ L_{pa}(G) \mid G \in PACF \}$$

Now we are in a position to compare the generative power of PACF, OCG, CCG, CG taking into account the difference between procedures and arguments.

We do this through the following sequence of lemma's:
($\mathcal{L}_{pa}CG$, $\mathcal{L}_{pa}REG$ are defined in the obvious way)

Lemma 8. $\mathcal{L}_{pa}SOCG \subseteq \mathcal{L}_{pa}OCG \subseteq \mathcal{L}_{pa}CF$

$\mathcal{L}_{pa}CCG \subseteq \mathcal{L}_{pa}CF$

Proof:

Similar to the proof in lemma 1. □

Lemma 9.

$$(\forall L)_{\mathcal{L}_{pa}CG} : \left[(\exists w)_L \quad (Pref_1(w) \in Vta) \supset L \notin \mathcal{L}_{pa}CCG \right]$$

Proof:

Trivial from the definitions

□

Lemma 10.

$$\exists L_1 \in \mathcal{L}_{pa}CCG \setminus \mathcal{L}_{pa}OCG$$

$$\exists L_2 \in \mathcal{L}_{pa}OCG \setminus \mathcal{L}_{pa}CCG$$

$$\exists L_3 \in \mathcal{L}_{pa}OCG \cap \mathcal{L}_{pa}CCG$$

Proof:

(1) $L_1 = \{A^n b^n \mid n \geqslant 1\}$. It is obvious that $L_1 \in \mathcal{L}_{pa}CCG$
It should also be clear that $\nexists G \in OCG$: $Ab \in L_{pa}(G)$ and consequently $L_1 \notin \mathcal{L}_{pa}OCG$.

(2) $L_2 = \{(aB)^n \mid n \geqslant 1\}$
It is trivial that $L_2 \in \mathcal{L}_{pa}OCG \setminus \mathcal{L}_{pa}CCG$ (use lemma 9)

(3) $L_3 = \{(A\ B)^n \mid n \geqslant 1\}$
Clearly $L_3 = L_{pa}(G)$ where

$$G = \langle \{s, a\}, \{A, B\}, \{s \xrightarrow{d} B\ a\ s,\ a \xrightarrow{d} A,\ s \xrightarrow{d} B\ a\}, \{s\}, \emptyset \rangle \in OCG$$

and also $L_3 = L_{pa}(G')$
where

$$G' = \langle \{s, b\}, \{A, B\}, \{s \xrightarrow{n} A\ b\ s,\ s \xrightarrow{n} A\ b,\ b \xrightarrow{n} B\}, \{s\}, \emptyset \rangle \in CCG$$

□

Lemma 11. $\exists L \in \mathcal{L}_{pa}CG \setminus (\mathcal{L}_{pa}OCG \cup \mathcal{L}_{pa}CCG)$

Proof:

Let $L = \{A^n b^n \mid n \geqslant 1\} \cup \{(a\ B)^n \mid n \geqslant 1\}$

The rest of the (easy) proof is left to the reader.

□

Lemma 12.

$$\exists L_1 \in \mathcal{L}_{pa}\text{REG} \setminus \mathcal{L}_{pa}\text{CCG}$$

$$\exists L_2 \in \mathcal{L}_{pa}\text{CCG} \setminus \mathcal{L}_{pa}\text{REG}$$

Proof:

(i) Take $L_1 = \{(aA)^n \mid n \in \mathbb{N}\}$

Clearly $L_1 \in \mathcal{L}_{pa}\text{REG}$

and by lemma 9, $L_1 \notin \mathcal{L}_{pa}\text{CCG}$

(ii) Take $L_2 = L_1$ (from lemma 10)

□

Lemma 13.

$$\exists L_1 \in \mathcal{L}_{pa}\text{OCG} \setminus \mathcal{L}_{pa}\text{REG}$$

$$\exists L_2 \in \mathcal{L}_{pa}\text{REG} \setminus \mathcal{L}_{pa}\text{OCG}$$

Proof:

(i) Take $L_1 = L_3$ from lemma 10

(ii) Take $L_2 = \{(Ab)^n \mid n \geq 1\}$

Clearly $L_2 \in \mathcal{L}_{pa}\text{REG}$ and $L_2 \notin \mathcal{L}_{pa}\text{OCG}$

(by a similar argument as in lemma 10, (i))

□

Lemma 14.

$$\exists L \in \mathcal{L}_{pa}\text{CF} \setminus \mathcal{L}_{pa}\text{CG}$$

Proof:

Take $L = \{a^n b^n \mid n \geq 1\}$

by definition $L \notin \mathcal{L}_{pa}\text{CG}$

□

Lemma 15.

$$\exists L \in \mathcal{L}_{pa}\text{OCG} \setminus \mathcal{L}_{pa}\text{SOCG}$$

Proof:

Take $L = L_2$ from lemma 10, by lemma 4 it follows that $L \notin \mathcal{L}_{pa}\text{SOCG}$

□

Lemma 16.

$$\exists L_1 \in \mathcal{L}_{pa}\text{SOCG} \setminus \mathcal{L}_{pa}\text{REG}$$

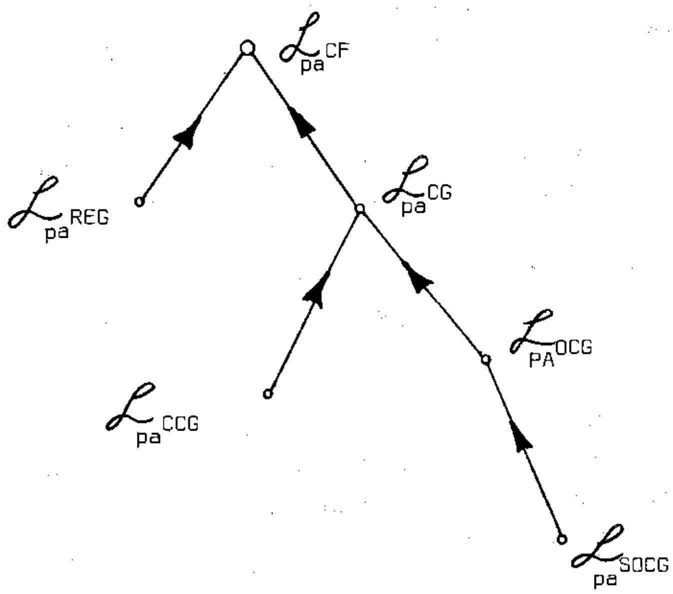
$$\exists L_2 \in \mathcal{L}_{pa}\text{REG} \setminus \mathcal{L}_{pa}\text{SOCG}$$

Proof:

Similar to the proof of lemma 6 and 7

The results from lemma 8 - 16 may be combined in the following diagram

Theorem 3.



Where an arrow indicates strict inclusion and no arrow between two classes means that the classes are incomparable but not disjoint.

§ 3. RECOGNIZERS

As a consequence of theorem 1, the construction of the automaton which accepts the language generated by an arbitrary completion grammar G is a straightforward task: first we construct a context-free grammar G' where L(G) = L(G') , then we construct a pushdown automaton P, on the basis of the context-free grammar G' with L(P) = L(G') . This is well known to be possible.
P is the required automaton.

There are however reasons not to do so:

(i) to preserve the strong generative capacity of completion grammars, it is necessary to develop recognizers which are structurally equivalent to their related grammars,

(ii) when we extend the model with a more precise treatment of the imposed order relations (see next chapter) it will prove to be necessary to have a way of coping with non-preferentially ordered expressions by means of an automaton.

In this section we therefore define constructs called completion automata and algorithms to translate completion grammars into completion automata and vice-versa.

A completion automaton is essentially a finite automaton with a pushdown store (also called the stack) upon which certain states are being stored in a last in first out manner, and with certain elements of the alphabet (called the final elements) associated with each final state of the automaton.

We can describe the activities of such an automaton as follows:

(i) symbols are being read from the linear input tape in a sequential manner from left to right

(ii) on top of the stack we find the current state of the automaton, if we can make a transition from one state to another one, the current state is removed from the stack and replaced by the new state.
If we cannot make such a transition the current state is pushed further on the pds. and the initial state is put on top of the stack. If we can make a transition the initial state is replaced by the new state, if we cannot make a transition, the string is rejected.

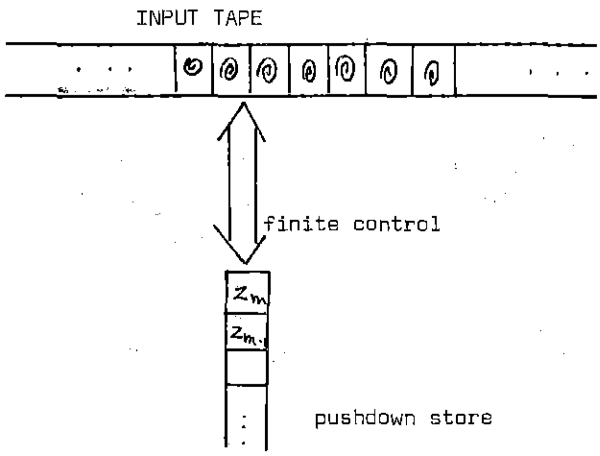
(iii) if the state on top of the stack is a final state, this state is popped up from the stack and the symbol that is being associated with the final state is written in front of the remaining string. (so that it will be the first symbol that is being read.)

Words are accepted iff the stack is empty and there is one final element left on the input tape.

Note that the completion automaton is very similar to the basic transition networks introduced by Woods (1970), except for the fact that elements are associated with final states.

Let us now make this picture more exact.

Schematically:



<u>Definition. 8.</u>

A *completion automaton* R is a 9-tuple R $=\langle Va, Vp, Vta, \mathcal{A}, A, AX\rangle$ where

1) Va and Vp are two finite nonempty sets called the set of *arguments* and the set of *procedure names or predicates* respectively.
2) $Va \cap Vp = \emptyset$, $V = Va \cup Vp$
3) $Vta \subseteq Va$ is the set of *terminal arguments*
4) $\mathcal{A} = \langle K, V, \Sigma, q_0, \delta, F\rangle$ constitutes a finite automaton (called the *embedded automaton* where
   - K is a finite nonempty set of states
   - Σ is a finite input alphabet and $\Sigma = Vta \cup Vp$
   - δ is a mapping from K X V into K
   - $q_0 \in K$ is the initial state
   - $F \subseteq K$ is the set of final states

The following restrictions hold for $L(\mathcal{A})$:

1) $L(\mathcal{A})$ should be finite (this is known to be a decidable question)
2) Each word $x \in L(\mathcal{A})$ should be of one of the following forms:
   1) either $x = A\alpha$ with $A \in Vp$ and $\alpha \in Va^{\star}$

In this case the path of transitions leading to the acceptance of x is called a nondepending path.

2) or $x = a A \alpha$ with $a \in Va$ (possibly λ), $A \in Vp$ and $\alpha \in Va^\star$. In this case the path of transitions leading to the acceptance of x is called a depending path.

5) $A \subseteq F \times Va$ is the *association relation.*

6) $AX \subseteq Va$ is the *axiomset.*

Definition 9.

1) A configuration s_i is a pair $\langle x,y \rangle$ with $x \in V^\star$ and $y \in K^\star$
(x represents the input tape and y the pushdownstore)

2) Let $a_1, \ldots, a_n \in V$, and $q_1, \ldots, q_m \in K$, $n,m \geq 0$ and s_1 and s_2 configurations where
$s_1 = \langle a_1 a_2 \ldots a_n, q_1 q_2 \ldots q_m \rangle$. We say that
s_1 *directly derives* s_2 denoted as $s_1 \vdash s_2$ if one of the following holds:

(a) TRANSITION

$$s_2 = \langle a_2 \ldots a_n, q_1' q_2 \ldots q_m \rangle \text{ where } q_1' \in \delta(a_1, q_1)$$

(b) PUSH

$$s_2 = \langle a_2 \ldots a_n, q' q_1 q_2 \ldots q_m \rangle \quad q' \in \delta\langle a_1, q_o \rangle$$

(c) POPUP

$$s_2 = \langle a' a_1 \ldots a_n, q_2 \ldots q_m \rangle \text{ iff } q_1 \in F \text{ and } \langle q_1, a' \rangle \in A$$

In all other cases s_2 is undefined

3) Furthermore let $\overset{\star}{\vdash}$ denote the reflexive and transitive closure of $\vdash$.

Definition 10.

Let $R = \langle Va, Vp, Vta, \mathcal{Q}, A, AX \rangle$ be a completion automaton

1) The language of R denoted as L(R) is defined by

$$L(R) = \{x \mid \langle x, q_o \rangle \overset{\star}{\vdash} \langle a, \lambda \rangle \; ; \text{ with } a \in AX,\ x \in (Vp \cup Vta)^\star\}$$

2) We say that $R = \langle Va, Vp, Vta, \mathcal{Q}, A, AX \rangle$ is a closed completion automaton iff

$$(\forall x \in L(\mathcal{Q}))\ (x = A\alpha,\ A \in Vp \text{ and } \alpha \in Va^\star)$$

3) We say that $R = \langle Va, Vp, Vta, \mathcal{Q}, A, AX \rangle$ is an open completion automaton iff

$$(\forall x \in L(\mathcal{Q}))\ (x = a_1 A \alpha,\ a_1 \in Va,\ A \in Vp \text{ and } \alpha \in Va^\star)$$

4) CA, CCA, OCA denotes the class of completion automata, closed and open respect.

5) $\mathcal{L}_{CA} = \{L(R) \mid R \in CA\}$, $\mathcal{L}_{OCA} = \{L(R) \mid R \in OCA\}$,

$\mathcal{L}_{CCA} = \{L(R) \mid R \in CCA\}$

Example 12.

Let R = ⟨ Va, Vp, Vta, $\mathcal{Q}$, A, AX ⟩ be a closed completion automaton where

1) Va = $\{a,b\}$
2) Vp = $\{A\}$
3) Vta = $\{b\}$
4) $\mathcal{Q}$ = ⟨ K, V, δ , q_o, E ⟩

$K = \{q_o, q_2, q_3, q_4, q_5\}$

$V = \{A, a , b\}$

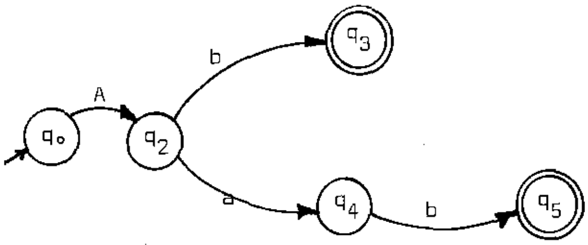
$\delta(q_o,A) = q_2$ $\delta(q_2,b) = q_3$

$\delta(q_2,b) = q_3$ $\delta(q_2,a) = q_4$

$\delta(q_4,b) = q_5$

5) A = $\{\langle q_3,a\rangle, \langle q_5,a\rangle\}$
6) AX = $\{a\}$

as a transition diagram:



Clearly the language accepted is $\{A^n b^n \ ; \ n \geq 1\}$

We try some strings:

Let x = A A b b

$\langle A\,A\,b\,b, q_o\rangle \vdash \langle A\,b\,b, q_2\rangle \vdash \langle b\,b, q_2q_2\rangle \vdash \langle b, q_3q_2\rangle \vdash \langle a\,b, q_2\rangle$

$\vdash \langle b, q_4\rangle \vdash \langle \lambda, q_5\rangle \vdash \langle a, \lambda\rangle$

Let x = A b b

$\langle A\ b\ b\ ,\ q_0\rangle \vdash \langle\ b\ b\ , q_2\ \rangle \vdash \langle\ b\ , q_3\ \rangle \vdash \langle a\ b,\ \lambda\ \rangle$

The automaton halts but the word is not accepted

Let x = A A b

$\langle A\ A\ b,\ q_0\rangle \vdash \langle\ A\ b, q_2\ \rangle \vdash \langle b, q_2 q_2\rangle \vdash \langle \lambda, q_3 q_2\rangle \vdash \langle a, q_2\rangle \vdash \langle \lambda, q_4\rangle$

The word is not accepted

<u>Example 13.</u>

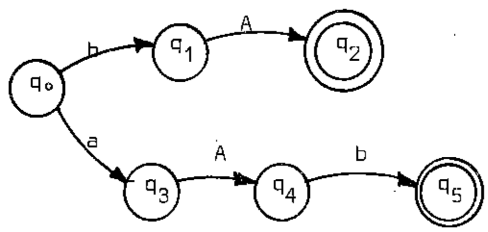
Let R = ⟨Va, Vp, Vta, 𝒬 , A, AX ⟩ be an open completion automaton where 𝒬, E, A, AX, Vta, Vp are exactly the same as in the previous example, except for the transition function δ :

$\delta(q_0,b) = q_1 \qquad \delta(q_0,a) = q_3$

$\delta(q_1,A) = q_2 \qquad \delta(q_3,A) = q_4$

$\delta(q_4,b) = q_5$

as a transition diagram:



Now $L(R) = \{b\ A\ (A\ b)^n : n \geq 0\}$

We try some strings:

Let x = b A A b A b

$\langle b\ A\ A\ b\ A\ b\ , q_0\rangle \vdash \langle A\ A\ B\ A\ b\ ,\ q_1\ \rangle \vdash \langle A\ b\ A\ b\ , q_2\rangle \vdash \langle a\ A\ b\ A\ b,\ \lambda\ \rangle$

$\vdash \langle A\ b\ A\ b\ , q_3\ \rangle \vdash \langle b\ A\ b\ ,\ q_4\ \rangle \vdash \langle A\ b\ , q_5\ \rangle \vdash \langle a\ A\ b, \lambda\ \rangle \vdash \langle A\ b, q_3\rangle$

$\vdash \langle b, q_4\ \rangle \vdash \langle \lambda, q_5\ \rangle \vdash \langle\ a, \lambda\ \rangle$

the word is accepted

<u>Lemma 17.</u> $\mathcal{L}_{CG} \subset \mathcal{L}_{CA}$

Let G = ⟨Va, Vp, P, Vta, AX, K⟩ ∈ CG, then we construct the automaton R = ⟨Va, Vp, Vta, $\mathfrak{A}$, A, AX⟩ as follows:

1) Va, Vp, AX are as in G

2) $\mathfrak{A} = \langle K, \Sigma, \delta, q_0, F\rangle$ is defined as follows:

Let Σ = Va ∪ Vp and for each $\phi \in V^*$ with p = ⟨a, ϕ⟩ ∈ P and K(p) = n or i we start a chain of transitions from q_0 such that for each element of ϕ we create a transition with this element as condition for the transition to take place.

In addition if p = ⟨a, $\bar{\phi}$⟩ ∈ P and K(p) = d or i we start a chain of transitions from q_0 such that for each element of ϕ we create a transition with this element as condition for the transition to take place and a new state, where $\phi = a\ A\ \phi'$, and $\bar{\phi} = A\ a\ \phi'$, $a \in Va$, $\phi' \in Va^*$.
The last element of ϕ will make a transition to a final state.

Note that we can always do so because L($\mathfrak{A}$) is finite due to the fact that P is finite.
Note also that as a consequence of this construction process there corresponds a unique final state q_f with each $x \in L(\mathfrak{A})$. We say that $\mathfrak{A}$ accepts x in the final state q_f.

Let $\langle q_f, a'\rangle \in A$ iff there is a production ⟨a', ϕ⟩ in P where ϕ is accepted by $\mathfrak{A}$ in the final state q_f.

Clearly as a consequence of this construction L(G) = L(R)

<u>Example 15.</u>

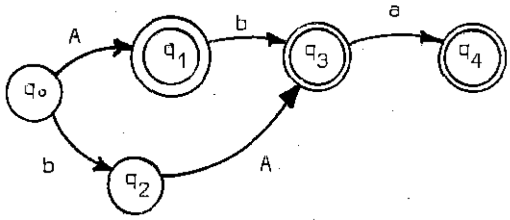
Let G = ⟨{a,b}, {A}, {a → A b a, a → A b, b → A}, {a}, ∅⟩ ∈ CG
then by applying the construction of the lemma we obtain:

R = ⟨Va, Vp, Vta, $\mathfrak{A}$, A, AX⟩ such that Va = {a,b}, Vp = {A} Vta = {a}
and $\mathfrak{A}$ = ⟨K, Σ, δ, q_0, F⟩ with $K = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{a, b, A\}$

$\delta(q_0, A) = q_1$ $\delta(q_0, b) = q_2$
$\delta(q_1, b) = q_3$ $\delta(q_2, A) = q_3$
$\delta(q_3, a) = q_4$

$F = \{q_1, q_3, q_4\}$

As a diagram



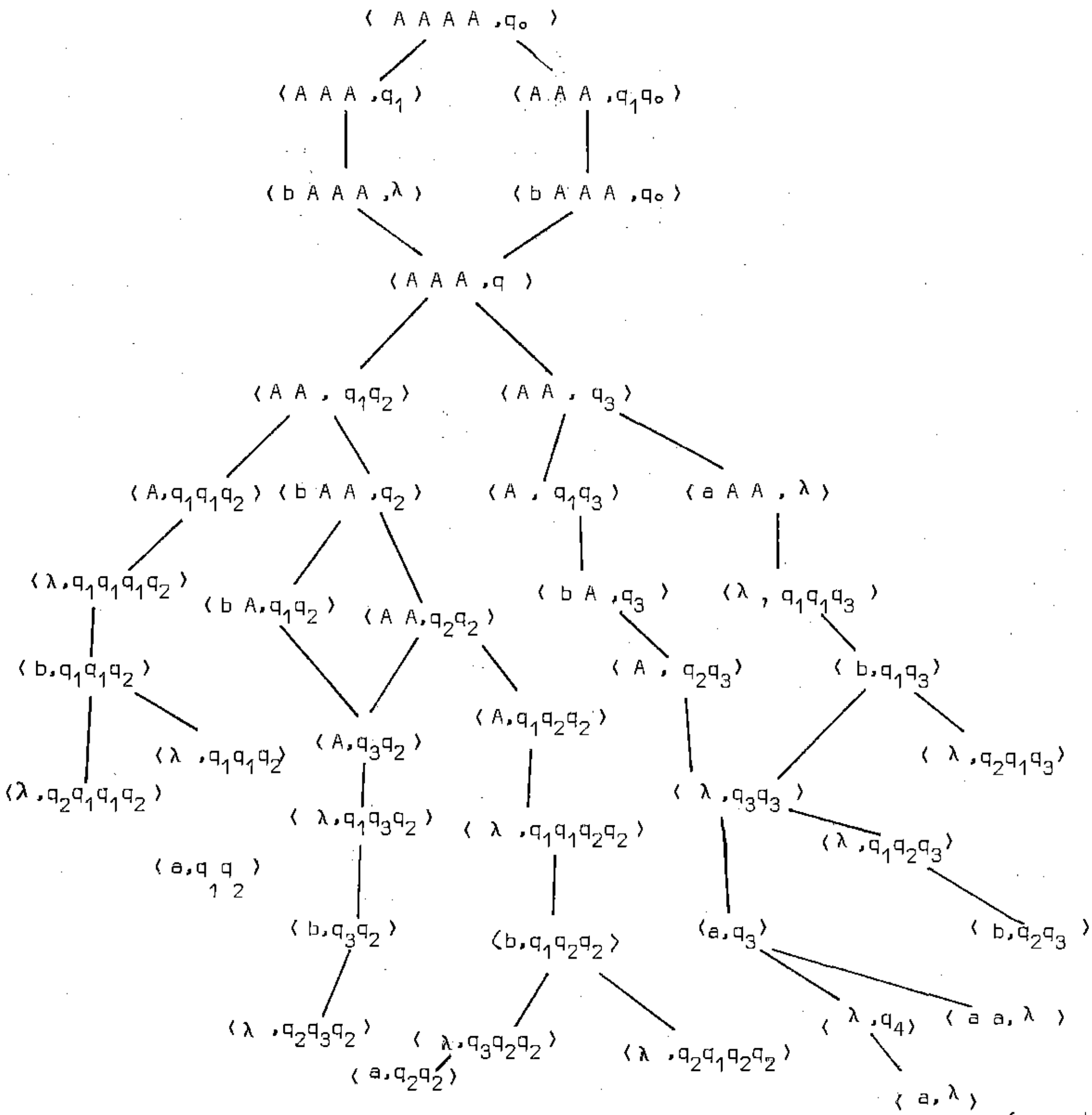
$AX = \{a\}$

$A = \{\langle q_1, b\rangle, \langle q_3, a\rangle, \langle q_4, a\rangle\}$

Clearly $L(R) = L(G) = \{A^{2n} : n \geqslant 1\}$

Let us give an example of a derivation, as is usual with nondeterministic processes we draw a tree to represent the parsing paths where a connection between two nodes means that the '$\vdash$' relation is present.

Let x = A A A A



(accepted)

Lemma 18. $\mathcal{L}_{CA} \subset \mathcal{L}_{CG}$

Let R = ⟨Va, Vp, Vta, $\mathfrak{A}$, A, AX ⟩∈ CA , then G = ⟨Va, Vp, P, Vta, AX, K⟩∈ CG is constructed as follows:

1) Let Va, Vp, Vta, AX be as in R

2) Let M1 = $\{x \in L(\mathfrak{A}) : x \text{ is of the form } A\phi \text{ , } A \in Vp, \phi \in Va^\star\}$

and

M2 = $\{x \in L(\mathfrak{A}) : x \text{ is of the form } a\,A\phi \text{ with } A \in Vp,\ a \in Va \cup \{\lambda\} \text{ and } \phi \in Va^\star\}$

Due to the restriction on L($\mathfrak{A}$) M1, M2 are finite and M1 ∪ M2 = L($\mathfrak{A}$)

3) P = $\{a \xrightarrow{n} \phi \mid \phi \in M1 \text{ and } \langle q_f, a\rangle \in A \text{ and } \phi \text{ is accepted by } \mathfrak{A} \text{ in the final state } q_f\}$

$\{a \xrightarrow{d} \phi \mid \phi \in M2 \text{ and } \overline{\phi} = a\,A\,\phi' \text{ , } \phi = A\,a\,\phi',\ \phi' \in Va^\star,\ a \in Va \cup \{\lambda\},\ \langle q_f, a\rangle \in A \text{ and } \overline{\phi} \text{ is accepted by } \mathfrak{A} \text{ in the final state } q_f\}$

The obvious proof then that L(G) = L(R) follows immediately.

Example 16.

Let R = ⟨Va, Vp, Vta, $\mathfrak{A}$, A, AX ⟩∈ CA be the automaton of the previous example. By applying the construction of the lemma we obtain:

M1 = {A, Ab, Aba}

M2 = {bA, bAa , A} and thus:

$$P = \{b \xrightarrow{n} A\ ,\ a \xrightarrow{n} A\ b\ ,\ a \xrightarrow{n} A\ b\ a\ ,\ a \xrightarrow{d} A\ b\ ,\ a \xrightarrow{d} A\ b\ a\ ,\ a \xrightarrow{d} A\}$$

We can clearly shorten P as follows.

$$P = \{b \longrightarrow A,\ a \longrightarrow A\ b\ ,\ a \longrightarrow A\ b\ a\}.$$

The grammar obtained is:

G = ⟨ {a,b} , {A} , P , {a} , K ⟩ and this is indeed the grammar we started with in the previous example.

Theorem 4. $\mathcal{L}_{CA} = \mathcal{L}_{CG}$

Proof:

The proof follows immediately from the lemma's.

§ 4. THE PRECEDENCE RELATION RECONSIDERED

In this section we extend completion automata such that they accept all possible combinations of a word which is normally accepted by a completion automaton after careful application of the rules. First we define the language that is to be defined and then define the extended completion automaton.

Note that we do not change the definition of the components of the automaton, only the way in which he operates, in particular we introduce an additional stack.

First we define the extended language

Definition 11.

Let G = ⟨Va, Vp, P, Ax, Vta, K⟩ ∈ CG

Let Ψ be the Parikh mapping, then define

$$c\,(L\,(G)) = \Psi^{-1}\,\Psi\,(L(G))$$

Definition 12.

An extended completion automaton $\overline{R}$, denoted as ECA , is a T-tuple

$\overline{R}$ = ⟨Va, Vp, Vta, $\mathfrak{G}$, A, AX ⟩ is as an ordinary CA.

A configuration is a triple:

$\langle x,y,z \rangle$ where $x \in V^*$, $y \in K^*$, $z \in V^*$

$$s_1 \vdash s_2 \quad \text{iff: } s_1 = \langle a_1a_2 \dots a_n, q_1q_2 \dots q_m, a_ia_{i+1} \dots a_{i+k} \rangle \qquad k,n,m \geq 0$$

and

TRANSITION

$$s_2 = \langle a_2 \dots a_n, q_1'q_2 \dots q_m, a_i \dots a_{i+k} \rangle \qquad \text{iff} \quad q_1' \in \delta(a_1,q_1)$$

PUSH 1

$$s_2 = \langle a_2 \dots a_n, q'\,q_1 \dots q_m, a_i \dots a_{i+k} \rangle \qquad \text{iff } q' \in \delta(a_1,q_0)$$

PUSH 2

$$s_2 = \langle a_2 \dots a_n, q_1 \dots q_m, a_1a_i \dots a_{i+k} \rangle \qquad \text{in any case}$$

PUSH 3 (called the emergency push)

$s_2 = \langle a_i a_2 \dots a_n, q_1 \dots q_m, a_{i+1} \dots a_{i+k} \rangle$ in any case

POPUP 1

$s_2 = \langle a' a_1 \dots a_n, q_2 \dots q_m, a_i \dots a_{i+k} \rangle$ iff $q_1 \in F$ and $\langle q_1, a' \rangle \in A$

POPUP 2

$s_2 = \langle a_1 \dots a_n, q_1' q_2 \dots q_m, a_{i+1} \dots a_{i+k} \rangle$ iff $q_1' \in \delta(q_1, a_i)$

Definition 14.

$L(R) = \{ w \mid \langle w, q_o, \lambda \rangle \vdash \langle a, \lambda, \lambda \rangle \quad a \in AX \}$

Example 17.

Let $R \in$ CCA with

$\langle \{a,b\}, \{A\}, \{b\}, \mathcal{Q}, \{\langle q_3, a\rangle \ \langle q_5, a\rangle\}, \{a\} \rangle$ and

$\mathcal{Q} = \langle K, V, \delta, q_o, E \rangle$ where

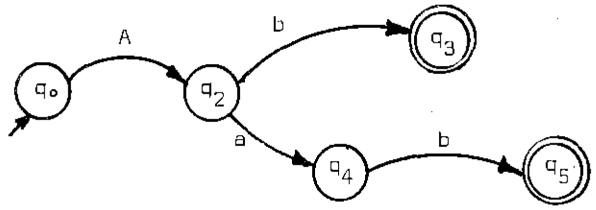
$K = \{q_o, q_2, q_3, q_4, q_5\}$

$V = \{A, a, b, A\}$

$\delta(q_o, a) = q_2$ $\delta(q_2, b) = q_3$

$\delta(q_2, b) = q_3$ $\delta(q_2, a) = q_4$

$\delta(q_4, b) = q_5$



We know already that $L(R) = \{A^n b^n : n \geq 1\}$ if CCA is consider to be not extended.

Let R' $\in$ECA then L(R') = c(L(R))

some examples (we only give the path leading to a final state)

Let x = A A b b

$\langle A\,A\,b\,b\,,\ q_o, \lambda\,\rangle \vdash \langle A\,b\,b\,,\ q_2, \lambda\,\rangle \vdash \langle b\,b\,,\ q_2 q_2\,, \lambda\,\rangle \vdash \langle b, q_3 q_2, \lambda\,\rangle$

$\vdash \langle a\,b\,,\ q_2, \lambda\,\rangle \vdash \langle b\,, q_4, \lambda\rangle \vdash \langle a\,, \lambda\,, \lambda\,\rangle$

(note that we did not use the additional stack)

Let x = b b A A

$\langle b\,b\,A\,A\,,\ q_o, \lambda\rangle \vdash \langle b\,A\,A\,,\ q_o, b\rangle \vdash \langle A\,A\,, q_o,\ b\,b\rangle \vdash \langle A\,,\ q_2,\ b\,b\rangle$

$\vdash \langle A\,,\ q_3,\ b\rangle \vdash \langle a\,A\,,\ \lambda\,,\ b\rangle \vdash \langle A\,,\ \lambda\ \ a\,b\rangle \vdash \langle \lambda\,,\ q_2, a\,b\rangle$

$\vdash \langle \lambda\,,\ q_4, b\rangle \vdash \langle \lambda\,,\ q_5, \lambda\rangle \vdash \langle a, q_5 \lambda\,\rangle$

Let x = A b A b

$\langle A\,b\,A\,b\,,\ q_o, \lambda\rangle \vdash \langle b\,A\,b\,,\ q_2,\ \rangle \vdash \langle A\,b\,, q_3\, \lambda\rangle \vdash \langle a\,A\,b\,, \lambda\,, \lambda\rangle \vdash \langle A\,b\,, \lambda\,,\ a\,\rangle$

$\vdash \langle b, q_2\,,\ a\rangle \vdash \langle b\,, q_4, \lambda\rangle \vdash \langle \lambda\,, q_5, \lambda\rangle \vdash \langle a, q_5, \lambda\rangle$

Example 16.

Let R be OCA with

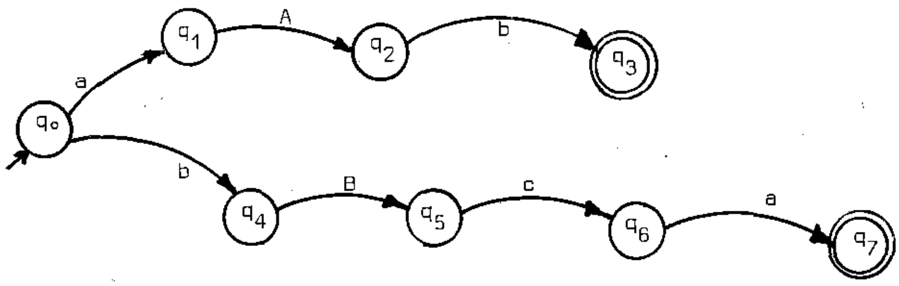
$R = \langle \{a,b,c\}\ ,\{A, B\}\ ,\ \{a,b,c\}\ ,\ \mathcal{Q}\ ,\{\langle q_3,a\rangle\ ,\langle q_7,b\rangle\}\ ,\{a,\ b\}\ \rangle$

$\mathcal{Q} = \langle K, V,\ \delta\,,\ q_o,\ E\ \rangle$ i.e.

$K = \{q_o,\ q_1,\ q_3,\ q_2,\ q_4,\ q_5,\ q_6,\ q_7\}$

$\delta\,(q_o\ ,a) = q_1$	$\delta\,(q_o\ ,\ b) = q_4$
$\delta\,(q_1,A) = q_2$	$\delta\,(q_4,B) = q_5$
$\delta\,(q_2,b) = q_3$	$\delta\,(q_5,c) = q_6$
	$\delta\,(q_6,a) = q_7$

$E = \{q_3,q_7\}$



If R is considered to be ECA then:

x = b B c a A b B c a

$\langle$ b B c a A b B c a , $q_o, \lambda\rangle \vdash \langle$ B c a A b B c a , $q_4, \lambda\rangle \vdash$

$\langle$ c a A b B c a , $q_5, \lambda\rangle \vdash \langle$ a A b B c a , $q_6, \lambda\rangle$

$\vdash \langle$ A b B c a , $q_7, \lambda\rangle \vdash \langle$ a A b B c a , $\lambda , \lambda\rangle$

$\vdash \langle$ A b B c a , $q_1, \lambda\rangle \vdash \langle$ b B c a , $q_2, \lambda\rangle$

$\vdash \langle$ b B c a , $q_o q_2, \lambda\rangle \vdash \langle$ B c a , $q_4 q_2 , \lambda\rangle \vdash \langle$ c a , $q_5 q_2 , \lambda\rangle$

$\vdash \langle$ a , $q_6 q_2 , \lambda\rangle \vdash \langle \lambda, q_7 q_2, \lambda\rangle \vdash \langle$ b , $q_2, \lambda\rangle \vdash \langle \lambda, q_3, \lambda\rangle$

$\vdash \langle$ a, $\lambda , \lambda\rangle$

The reader should now try the very nice example of x = b b B c a A B c a for himself.

<u>Theorem 5.</u> $\mathcal{L}_{ECA} = c(\mathcal{L}_{CG})$

The standard proof of this theorem is left to the reader.

Given the above results one can introduce the following concepts:

<u>Definition 15.</u>

For a given R $\in$ CA the ungrammaticality of a word $w \in L(R)$ denoted as $u_R(w)$ is defined by MIN (MAX $|s_i(3)|$) $1 \leq i \leq n$ where a configuration is a triple $\langle s_i(1), s_i(2), s_i(3)\rangle$ and s_1 is the initial configuration and

$s_1 \vdash \dots \vdash s_n$ is a derivation accepting w, i.e. w = s_n

Example 19.

For the system in example 17.

(i) $w = A\ A\ B\ b$ then $u_R(w) = 0$

(ii) $w = b\ b\ A\ A$ then $u_R(w) = 2$

(iii) $w = A\ b\ A\ b$ then $u_R(w) = 1$

For the system in example 18.

(i) $w = b\ B\ c\ a\ A\ b\ B\ c\ a$
then $u_R(w) = 0$

(ii) $w = a\ c\ B\ b\ a\ c\ B\ b\ A$
then $u_R(w) = 7$

Definition 16.

For a given $R \in ECA$ the complexity of a word $w \in L(R)$ denoted as $com_R(w)$ is defined by

$$MIN\ (\ MAX\ (|s_i(2)|)) \quad 1 \leqslant i \leqslant n$$

(Note that this definition does also apply to not extended completion automata)

Example 20

if $w = A\ A\ b\ b$
then $com_R(w) = 2$

if $w = b\ A\ A\ b\ A\ b$
then $com_R(w) = 1$

if $w = b\ A\ A$
then $com_R(w) = 1$

Very interesting results are obtained by comparing the complexity degrees of $\mathcal{L}_{OCG}$ and $\mathcal{L}_{CCG}$.

These and other results, with special reference to the application to natural languages, can be found in Steels & Vermeir (1976).

§ 5 APPLICATIONS

Although this paper concentrates on the formal properties of completion grammars, in this final chapter we deal as briefly as possible with the view on language that we wanted to model with completion grammars.

The information in this section is essentially contained in Steels (1976a).

5.1. *Completion grammars as a means of formalizing case systems*

There exist several proposals for case systems. These proposals differ mostly in the cases that are adopted and in the level of 'deepness' that is aimed at. What they all seem to have in common are the following ideas:

(i) A case is a (binary) relation between a predicate and one of its arguments. We call the name for the case the case indicator or simply indicator. Examples of commonly found indicators are agent, object, instrument, source, goal, range, etc... .

(ii) Case relations affect in two ways language:

(a) They are expressed (or recognized) by means of surface signals such as prepositions, case affixes, word order, intonation, etc... . These signals are called case markers.

(b) The system is further refined by the idea that semantic properties of the unit under consideration act as selection restrictions.

(iii) A case structure or case frame for a particular predicate is a set of case relations that occur with that predicate, TOGETHER WITH information how they are expressed (or recognized) in language, i.e. the case markers and selection restrictions involved.

(iv) Parsing a language expression with the guidance of a case system (where a case system is just a set of case structures for a language) involves the discovery of the case structure, that means a decision on which units in the sentence fill in which slots in the structures; this decision is being made on the basis of the case markers and selection restrictions of the unit under consideration.

Producing a language expression with the guidance of a case system involves filling in the appropriate concepts in the places where the selection restrictions permit this, and translating the case markers in their surface structure equivalents.

Given the above statements on the nature of case systems, we will now try to relate this to the formal model that was studied in this paper.

At the center of the case structure we find a predicate, the rest of the structure contains arguments. So, a first step will be to make a basic distinction between a set of predicates (Vp) and a set of arguments (Va).

In connection with a procedural semantics, predicates can be considered as the names of procedures and Vp is therefore also called the set of procedure names.

One of our basic insights is a fundamental distinction between predicates in prefix-position, predicates in infix-position and predicates which appear in both. (There are languages where still other positions are possible, e.g. postfix, it should be clear however how the model can be extended to cope with these). The case frame, and thus a production in the grammar, should reflect this basic opposition. We call rules which contain predicates which should give rise to an expression where they stand in prefix position, infix position, or both, nondepending productions, depending productions and indifferent productions respectively.

In a generative viewpoint, the order of the expressions in a language is defined by the derivation relation. Hence we will have a derivation mechanism realizing prefix-position, this is called a closed derivation, and one realizing infix notation, this is called an open derivation.

It may seem superfluous that both prefix and infix predicates appear in the same language, it can be shown however on the basis of cognitive arguments (in particular memory limitations) that it is necessary for humans to have both types of predicates. It would lead us too far to deal with this point here.

To make the model more precise we now consider the status of the arguments.

An argument is said to have three levels:

(i) argument type, being the semantic restrictions and the case markers

(ii) argument name, being the case indicator

(iii) argument value, being the particular object involved (this can be a pointer to a place in the data base for example).

It is easy to see that when studying the formal properties of case systems only level (i) is important, indeed the argument names are nothing else but mnemonic labels for relations which are positionally defined in the structure, and argument values are only important for the actual semantic interpretation, not for the parsing. So we are left with the argument type. And also here some further reduction is still necessary.

The argument type contains as we said semantic properties of the argument that act as selection restrictions, and case markers. Arbitrary arguments can however contain a very large number of properties but there are of course only a certain set of properties relevant as a selection restriction for a particular case structure. Hence what we will actually find in our grammar rules is a set of relevant properties and for a particular argument to match, the set of relevant properties should be a subset of the set of all properties of that argument. Va therefore contains sets of relevant properties of arguments.

Also it is necessary to delimit a subset of Va, the so called nonterminal arguments, being those arguments which cannot appear in a language expression itself but are to be realized further.
E.g. if a certain argument type contains the indicator that a case affix should be present, then the argument is terminal if and only if this case affix is indeed added to the word form of the argument. In the grammar definition we will indicate the terminal arguments, and the nonterminal arguments are all the other arguments.

In addition we will incorporate the <u>result</u> of semantic interpretation , of which at least the type information is known in advance, in our definition of a case frame. This will guarantee among other things the recursiveness which is necessary to cope with the infiniteness of language.

With the above explanations the reader should understand all the components of a completion grammar G = $\langle$Va, Vp, P, AX, Vta, K$\rangle$ and the derivation processes $\boldsymbol{\Rightarrow}$, $\Rrightarrow$, and $\Longrightarrow$.

5.2. Some examples

example (a) A subset of the FORTRAN IV programming language

Although completion grammars were designed to cope with natural languages, they are equally well applicable to programming languages. These languages are (syntactically) simpler than the natural languages, and therefore not all aspects of the model can be illustrated. It is our hope that completion grammars will once form a tool in the development of a more human oriebted outlook of programming languages.

Let G = $\langle$Va, Vp, P, AX, Vta , K$\rangle$ be a completion grammar with
Va = {statem , num , log , rightpar}
Vp = { = , + , - , / , x , (, GOTO, IF, AND, OR, NOT, END, GT, EQ, LE , LT }
AX = {statem}
Vta = {num , log} (i.e. <u>num</u>erical variable or constant and <u>log</u>ical variable or constant)
and P:

1. statem $\xrightarrow{d}$ = num num
2. num $\xrightarrow{d}$ $\left\{\begin{array}{c} + \\ - \\ / \\ \times \end{array}\right\}$ num num
3. num $\xrightarrow{n}$ (num rightpar

4. statem $\xrightarrow{n}$ IF log statem

5. statem $\xrightarrow{n}$ GOTO num

6. log $\xrightarrow{d}$ $\left\{\begin{matrix}\text{GT}\\ \text{LT}\\ \text{LE}\\ \text{EQ}\\ \text{NE}\end{matrix}\right\}$ num num

7. log $\xrightarrow{d}$ $\left\{\begin{matrix}\text{AND}\\ \text{OR}\end{matrix}\right\}$ log log

8. log $\xrightarrow{n}$ NOT log

9. statem $\longrightarrow$ END

10. rightpar $\longrightarrow$)

11. log $\xrightarrow{n}$ (rightpar

(Note that we always put arguments between square brackets)

Some derivations:

(i) statem $\overset{1}{\Longrightarrow}$ num = num (E.g.: I = 1)

(ii) statem $\overset{4}{\Longrightarrow}$ IF log statem $\overset{11}{\Longrightarrow}$ IF (log rightpar statem

$\overset{10}{\Longrightarrow}$ IF (log) statem $\overset{7}{\Longrightarrow}$ IF (log OR log) statem

$\overset{6}{\Longrightarrow}$ IF (num EQ num OR log) statem

$\overset{3,10}{\Longrightarrow}$ IF (num EQ num OR (log)) statem

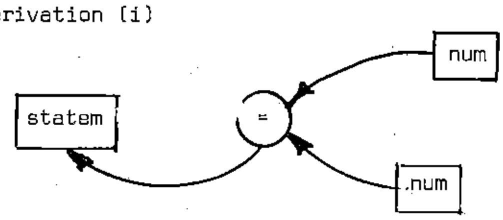
$\overset{6}{\Longrightarrow}$ IF (num EQ num OR (num GT num)) statem

$\overset{4}{\Longrightarrow}$ IF (num EQ num OR (num GT num)) GOTO num

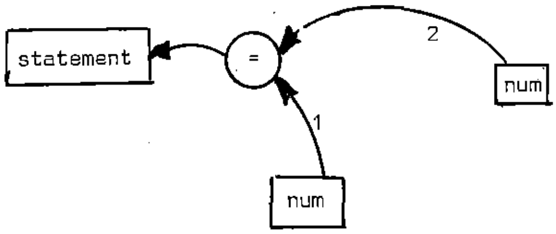
(E.g.: If (I EQ 1 OR (I GT J)) GOTO 16)

As is usual with grammars, one can define structures assigned by the grammar to a particular string of the language. The structures assigned by completion grammars are called relation strucutres. They work as follows: for a procedure draw a circle and for an argument a square. Argument squares can be divided into two sublevels representing the level for the argument type and one for the argument value. Input relations are represented by directed lines leaving a circle and entering a square.

Example for derivation (i)



If there is any need to specify names or labels for the case relations (i.e. case indicators), we write these on the directed lines:

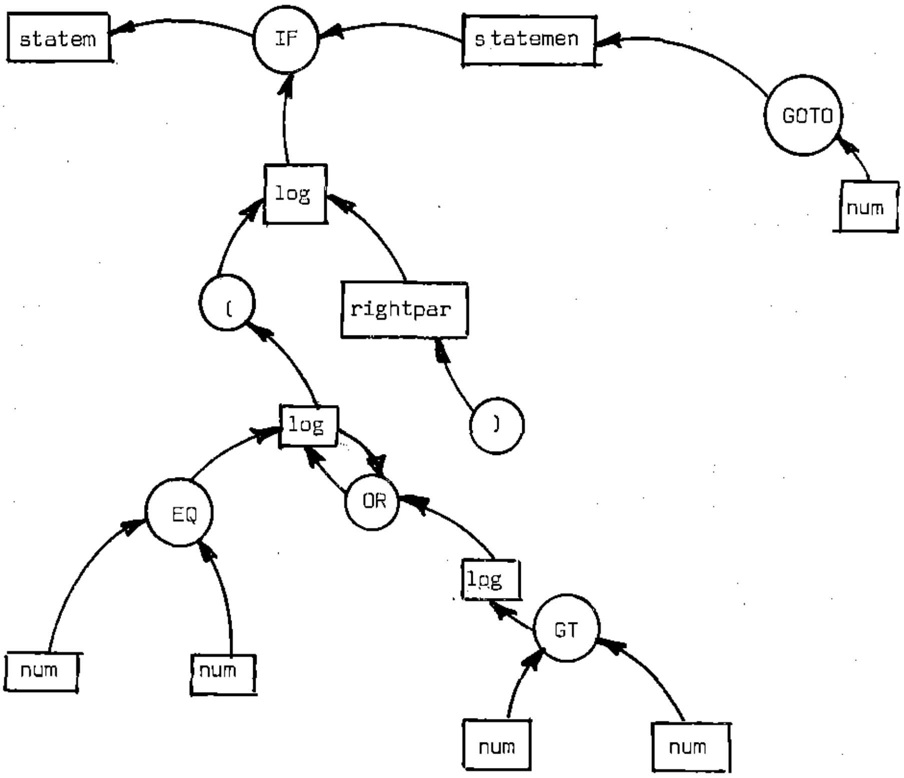


Where '1' denotes 'first input argument' and '2' denotes the second input argument'.

If we want to specify these indicators in the grammar we write the case indicator as a subscript:

$$\text{statem} \longrightarrow = \text{num}_1 \quad \text{num}_2$$

The relation structure for derivation (ii). (Note that in a strict open derivation output and first input argument are identical, therefore we get a symmetric relation in the structure)



Example:

Let us now give another grammar which contains case frames for some arithmetic procedure names. (It is the smallest 'natural language' example we could think of).

Let G = $\langle$ Va, Vp, P, AX, Vta, K $\rangle$ be a completion grammar with Va = $\{$ num , num,prep:of , num,prep:by $\}$ Vp = $\{$ divided, division, is, of, by , the $\}$, AX = $\{$num$\}$ and P contains the following productions

1. num $\xrightarrow{n}$ division num,prep:of num,prep:by
2. num $\xrightarrow{d}$ divided num num,prep:by
3. num,prep:by $\xrightarrow{n}$ by num
4. num,prep:of $\xrightarrow{n}$ of num
5. num $\xrightarrow{d}$ is num num
6. num $\xrightarrow{n}$ the num

A derivation:

num $\overset{5}{\Longrightarrow}$ num is num $\overset{6}{\Longrightarrow}$ the num is num

$\overset{1}{\Longrightarrow}$ the division num,prep: of num,prep:by is num

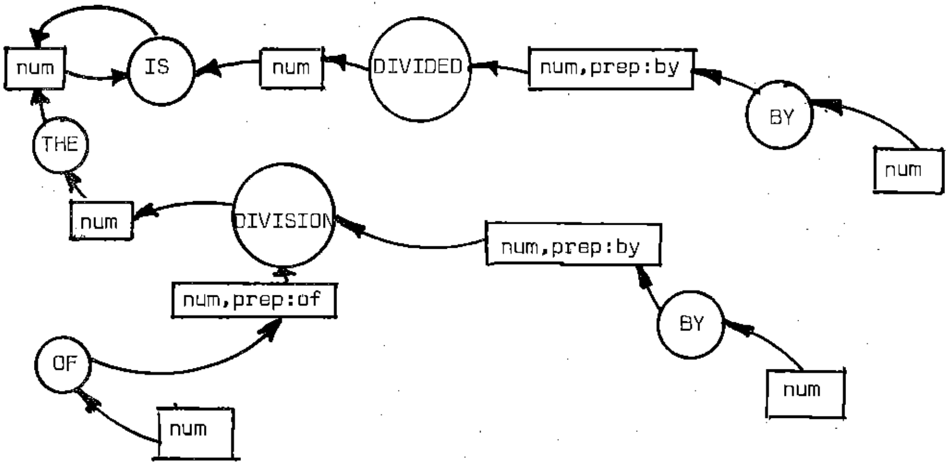
$\overset{4}{\Longrightarrow}$ the division of num is num

$\overset{3}{\Longrightarrow}$ the division of num by num is num

$\overset{2}{\Longrightarrow}$ the division of num by num is num divided num,prep:by

$\overset{3}{\Longrightarrow}$ the division of num by num is num divided by num

A possible realization of which is 'the division of 4 by 2 is 4 divided by 2'.
The relation structure for this derivation:



This example shows clearly how prepositions are expressed and recognized in their function as case markers: prepositions are considered as predicates which add simply the marker that the prepositions is present to the list of properties in the argument type of the result. (Note it can be that prepositions do more than a mere syntactic functioning, this fact only makes the formalization even more valid.)

Normally nouns, adjectives (in front position), prepositions, a.o. are defined by nondepending productions, whereas verbs, adjectives (in post-noun positions) participles, etc... are defined in depending productions.

Another aspect, namely the case affixes are treated in a similar fashion. In particular, the case suffixes, which are added at the end of a word form, are simply depending procedures whereas case prefixes, which are added in front of a word form are nondepending procedures. Although the consequences for morphology should be further investigated, this way of dealing with them guarantees a more 'semantically' oriented treatment of morphology and a unifying approach to surface analysis. Let us now deal with a third aspect: order.

For phrase structure grammars and similar systems 'being grammatical' means that elements of certain syntactic categories are present in a particular order, i.e. these grammars define a dominance relation (x belongs to the category y) and a precedence relation (x comes before y). With completion grammars we define first of all functional relations among arguments and predicates, and hence we can have a freeer attitude towards word order. The need to have systems which are not so strict bound to a linear order has often been mentioned and various attempts have been made to construct grammars which are free from word order (see Levelt (1973,104)).

A completion grammar defines a weaker precedence relation. Weaker because order CAN be an element in the decision about which relations exist and therefore a completely free word order will not do. The notion of a weaker precedence relation order is introduced by considering the order imposed by a strict application of the grammar rules as a preferential order. If certain cases , i.e. arguments, are missing in an expression this expression is said to be incomplete. Between an incomplete expression and a preferentially ordered one, it is possible to define a gradually increasing degree of grammaticality (see the section on order in this paper). We have also seen that it is possible to increase the power of completion automata by precise methods such that they accept not only preferentially ordered expressions but also not preferentially ordered ones, and when order IS a means of decision making, the preferential order has a higher priority.

The term completion grammar/automaton (which is due to M.A.M. Verreckt) is reflecting the fact that patterns are being described, and during analysis the parser looks for such patterns to be completed. When the elements that complete the pattern will occur is of lesser importance than their occurrence.

A more detailed treatment of applications for natural languages is given in Steels (1975b). In this paper the reader can find implemented parsing systems for completion grammars and some experiments in language understanding systems.
We also started to apply the model on a large scale in a project for the construction of an automatic translation system for notes used on catalog cards in libraries from and into the different languages in the EEC.

We are also considering a possible extension of the theory by the introduction of rule production mechanisms. These rules would construct completion grammar type rules on the basis of (i) the information to be expressed and (ii) information on how this information should be expressed, and (iii) information on how this should be done in a given language.

References

Levelt, W.J.M. (1975) Formal grammars in linguistics and psycholinguistics, II. Mouton, Den Haag.

Salomaa, A. (1973) Formal languages, Academic Press, New York.

Steels, L. (1975a) Recent research results in computational semantics. In: Nickel, A. (ed) Proceedings of the fourth international congress of applied linguistics. Forthcoming.

Steels, L. (1975b)Completion grammars and their applications. Antwerp papers in Linguistics. Nr. 3. U.I.A.

Steels, L. (1976a) On formalizing case systems. Paper submitted to the Artificial Intelligence and simulation of behavior summer conference 1976 Edinburgh.

Steels, L. and D. Vermeir (1976) Some results on the relation of word order to grammatical complexity. Forthcoming.

Woods, W.A. (1970) Transition network grammars for natural language analysis. Communications of the ACM 13 (10), pp. 591 - 606.

ISSUES OF ANTWERP PAPERS IN LINGUISTICS

1 - 1975 Luc Steels : Parsing Systems for Regular and Context-free Languages.

2 - 1975 Johan Van der Auwera : Semantic and Pragmatic Presupposition.

3 - 1975 Luc Steels : Completion Grammars and their Applications.

4 - 1976 Georges De Schutter & Eddy Kockx : Meervoudsvorming en Vervoeging in het Nederlands. Een morfofonologische Proeve.

5 - 1976 Luc Steels & Dirk Vermeir: On the formal Properties of Completion Grammars and their related Automata.