UNIVERSITEIT ANTWERPEN

UNIVERSITAIRE INSTELLING ANTWERPEN

DEPARTEMENT GERMAANSE FILOLOGIE

ASPECTS OF A MODULAR THEORY OF LANGUAGE

VOL I.

Proefschrift ter verkrijging van de graad van Doctor
in de Letteren en Wijsbegeerte aan de Universitaire
Instelling Antwerpen te verdedigen door *Luc STEELS*

Promotor:
H. Brandt-Corstius

Wilrijk,1977

PREFACE

In recent decades we have seen an enormous rise of investigations into the phenomenon of natural language. Each time it was shown that a certain aspect of this phenomenon was even more complex than previously thought or that a new aspect should be incorporated in the investigations. The present work has not the ambition of adding new subject areas to the expanding field of natural language study. We will not try to re-investigate certain areas of knowledge or apply existing theoretical models to unexplored language details. Instead we go back to the groundwork. We want to present and try out a new approach towards the investigation of language.

We consider it to be the task of linguistic theory to provide answers for the following questions: What sort of phenomena are used by natural languages to fulfil their task as a medium of communication ? How do each of these phenomena occur in a particular natural language ? What kind of systems are necessary to produce and perceive the linguistic phenomena ? How do these systems cooperate to understand or produce natural language ? How can we construct mechanisms that learn to cope with the phenomena found in natural languages ?

It is generally accepted that a grammar model constitutes the description of knowledge about language which is used by the speaker/hearer to produce/understand his language, the so called competence. We will accept this assumption. Normally however this viewpoint does not affect the theory of grammar itself. One constructs a grammar theory and then just hopes without further investigation that it represents the kind of knowledge necessary for a perceptual model. For our own research we decided to work the other way round. We tried to construct a perceptual model and studied what the implications are for the grammar theory itself.

By doing so, we found out that a fundamentally different linguistic theory is highly desirable. Not so much as regards the descriptive claims being made but more as regards the formal structure of the theory. The most important difference is that all knowledge sources are brought together in modules or specialists which can become active independently of each other.

In this work we present the basic principles of this new kind of theory. To illustrate them on the basis of the vast amount of language phenomena known today is an impossible task in a small amount of time. So we will pick out two basic aspects of language: grammatical function and case, and show how the theory formulates rules for them, and how the rules can be used in an empirical description. At the same time we will provide a perceptual model that 'consults' the knowledge represented in the grammar, to analyze and produce natural language, again basically concentrating on function and case.

Throughout the work, we try to satisfy strictly the requirements of exactness characteristic of scientific investigations. All models will be formally defined and for the perceptual models we will even present computer programs with which experiments can be performed to confirm the theories.

As a final remark we want to stress that the model to be presented here is not the final version of our theory nor an endpoint of our research. On the contrary, we feel a need for constant self-criticism, continuous revision and certainly further extensions (which may affect already existing parts of the theory). It is therefore better to call this work a progress rather than a final report.
Nevertheless we think that the general direction of the research is sufficiently clear and that the results so far obtained are sufficiently strong to justify the presentation now.

There is a Zen proverb saying 'a finger is needed to point at the moon but once the moon has been recognized we shouldn't bother about the finger'. In the same spirit we invite the reader to concentrate on discovering the ideas contained in the work and to forget about the deficiencies and errors which will undoubtely be present in the current presentation.

TABLE OF CONTENTS

INTRODUCTION

In this thesis we present the first approach of a new theory about the nature and mechanics of natural languages. This theory contains two parts:

(i) A description theory dealing with the problem how knowledge about the language systematics can be formalized. We will do this by introducing a set of independently consultable modules where each module explicates the relation of a certain factor and the language phenomena used to signal the factor. This explication is neutral in the sense that there is no bias towards generation or analysis.

(ii) A process theory showing how the linguistic knowledge is used to analyse or produce natural language.

As a whole the work is organized as follows. In a first volume the reader will find a chapter on foundations and a chapter on the description theory. The chapter on foundations contains all terms, concepts and systems which form the mathematical basis for the theories to be discussed later. Also we will discuss some metatheoretical assumptions. The chapter on the description theory deals with a detailed and formal description of the modular grammar theory which forms one of the main contributions of this work. In a second volume the reader will find a chapter on the process theory and on the implementation of this process theory on a computer. As regards the process theory we will in particular be engaged in a detailed presentation of a parsing system for natural language on the basis of the modular linguistic theory of chapter one. A system for language production also based of this linguistic theory will be presented only on an intuitive level. The chapter on the implementation contains a detailed and fully explicit definition of a parsing system for natural language as described theoretically in the chapter preceeding it.
The third volume is devoted to experiments and examples. Here we will discuss numerous examples for different languages and perform experiments with the system to illustrate the various points of the theory.
The final volume contains the conclusions of our work, the index and the bibliography.

On the whole this is a theoretical work which implies that the empirical interpretation will be restricted to what we need for the examples illustrating the theory. We will even at different points give different grammars or present facts which do not necessarily hold for the language in general. We invite the reader to take the same free position as regards empirical interpretation and we hope that our presentation will stimulate him/her to use the formalism in a creative way.

References to other work will be scarce. This is a consequence of our method of absorbing scientific information by discussions, lectures, personal communications, in other words by oral rather than written communication. This happened especially at the Summer School for Mathematical and computational linguistics in Pisa, at the Tutorial on Computational semantics in the Institute for semantics and cognitive studies in Lugano , the Tutorial on Montague grammar in Amsterdam and at the various conferences (especially the AI.conference in Edinburgh and the Computational Linguistics Conference in Ottawa) and seminars which we were able to attend due to generous support from our department.

We apologize for all the errors either due to incompetent usage of the English language or lack of care in the formal details. We hope that they will form no fatal obstacle for understanding our ideas.

We are well aware that the processing of such a large piece of work as the present one is a hard and time/energy consuming job. Let us hope that the ideas contained in the work will stimulate the reader in his own research efforts and that he will gain some new ideas for his own problems.

Antwerp, May, 1977

§ O. FOUNDATIONS

In this chapter we introduce a number of auxiliary notions from set theory and recursive function theory that will be used to define the theory under discussion in the other chapters. More in particular we will define several representation constructs such as atoms, n-tuples, strings, sets, relations and languages and present a graphical format and an implementation representation for each of these. Also we will introduce the reader to the intuitions behind the notion of computation and define some abstract systems for performing computations,in particular finite state machines and recursive transition networks.

Another topic of this chpater is a short discussion on some metatheoretical considerations. Here we will discuss the metatheoretical structure of the theory, the status especially as regards falsification and completeness and the experimental method.

CHAPTER O.

§ O.1. INTRODUCTION TO THE THEORY OF REPRESENTATION

§ O.2. INTRODUCTION TO THE THEORY OF COMPUTATION

§ O.3. METATHEORETICAL CONSIDERATIONS

0.1. Introduction to the theory of representation

In this section, the type of objects used on all the various levels of a linguistic theory are introduced and discussed. The study of these objects goes under the heading of the theory of representation. This is so because each object in the theory (e.g. a structural description) represents linguistic information about another object on another theoretical level to which it is related (e.g. a natural language sentence). In general, let us call an object defined by a theory of representation a representation construct. What sort of information is represented in a construct will be discussed in following sections. Here we concentrate on the type of constructs used. The mathematical foundations for the present investigation are provided by set theory.

A definition of representation constructs involves three aspects:

(i) A formal definition in which the logical or set theoretic aspects of the representation construct become apparent, we call such a representation the original or basic representation.

(ii) A formal definition of the graphical representation which is used for didactic purpose. It is obvious that there should be a homomorphical mapping between the original and the graphical representation.

(iii) (In computational linguistics) a formal definition of the implementation representation, i.e. the way in which the representation is physically stored in terms of machine manipulatable entities. It is again obvious that we want an homomorphical correspondence between the original and the implementation representation. Instead of remaining close to the physical storage formats, we will present as implementation representation a symbolic representation which can be processed by a machine.

It turns out that we can distinguish a hierarchy of types of representations. Within the hierarchy there are two levels: The first level consists of essentially finite basic representation constructs, such as atoms, n-tuples and strings. The second level consists of generalizations over these basic representation constructs in that now sets of basic constructs are represented. In this way we generalize from atoms to sets, from n-tuples to relations and functions and from strings to languages.

Schematically:

	type 1	type 2	type 3
level 1	atoms	n-tuples	strings
level 2	sets	relations	languages

In the following subsections, we will define for each type the representation constructs on each level. Also we will give some comments on the interaction of the various types of representation and their respective power.

TYPE 1.

Level 1: ATOMS

Definition

An atom is a finite sequence of characters considered to stand for a nondivisible primitive representation construct.

Example

21, ATOM, NOUN are atoms

Definition

Two atoms are equal if they have the same outlook.

Definition

NIL is the 'null' atom

Definition

An occurrence of an atom is the actual appearance of the sequence of characters in space/time.

Comment: The same atom can occur at several distinct times/places and it may be that this time/place relation is important. Note that the atom vs. occurrence of atom distinction is equal to the type vs. token distinction in linguistics.

Level 2: sets

Now we generalize over atoms by considering collections of atoms.

Definition

A set is a well-defined collection of atoms. If the atom a is an element of the set S, thenwe say $a \in S$, if it is not, we say that $a \notin S$.
A set is defined either by listing all its members, separated by commas and enclosed in brackets $\{\ \}$, or by specifying a characteristic property which is true for all members in the set and false for those not in the set. Let P(x) be such a characteristic property, then the set is defined by $S = \{ x \mid P(x)$, i.e. the set of x such that P(x) is true.$\}$

Example

$S = \{1,2,3\}$ is a set . $2 \in S$ is true.
$S'' = \{ x \mid x$ is an even number$\}$ is a set. $3 \in S''$ is true.

Definition

$\emptyset$ is the set containing no elements, the null set, or empty set. A finite set is a set containing a finite number of elements. An infinite set contains an infinite number of elements.
The number of elements in a set S is denoted as $\# S$.

Definition

A set A is equal to a set B, denoted as A = B, if and only if every element in A is also in B and vice-versa.

Example

$S = \{1,2,3\} = \{1 , 1, 2, 3\} = \{3,2,1\} = \{3,3,2,2,1,1\}$, etc;
Note that neither the ordering nor the occurrence plays a role. Some concepts as regards sets that we will need further on:

Definition

Let A and B be two sets then if $x \in A$ implies that $x \in B$, we say that A is a subset of B, denoted as $A \subseteq B$. Furthermore is there is an $x \in B$ which is not in A, then A is a proper subset of B, denoted as $A \subsetneq B$.

Example

Let $A = \{1,2,3\}$ and $B = \{2,3\}$ then $B \subsetneq A$. $A \subseteq A$.

Definition

Operations over sets: Let A and B be two sets, then the union of A and B denoted as $A \cup B$ contains those elements which belong to A or to B or to both; the intersection of A and B, denoted as $A \cap B$, contains all elements which belong to A and to B; the difference of A and B, denoted as A - B is the set of elements which belong to A but not to B; finally, the complement of A as regards the universe U, denoted as A', is the set of

elements which belong to U but not to A'.
If $A \cap B = \emptyset$, we say that A and B are disjoint sets.

Example

Let $A = \{1,2,3\}$ and $B = \{2,3,4\}$ then $A \cup B = \{1,2,3,4\}$
$A \cap B = \{2,3\}$, $A - B = \{1\}$ and with $U = \{1,2,3,4\}$
$A' = \{4\}$
A and B are not disjoint.

The only way of structuring that occurs with sets is by letting a set be an element of another set.

Definition

A family of sets or a class of sets, is a set of which the members are sets themselves.
The powerset of S denoted as $\mathcal{P}(S)$ is the family of all subsets of the set S: $\mathcal{P}(S) = \{A \mid A \subset S\}$.
In general: $\# \mathcal{P}(S) = 2^{\# S}$.

Example

$S = \{\{1\}, \{2,3\}, \{4\}\}$ is a family of sets. Let S be $\{1,2,3\}$ then
$\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{1,3\}, \{1,2,3\}\}$

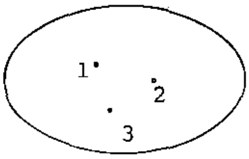
So far we discussed only the basic representation, now we turn to the other format: graphical representation.

Graphical representation

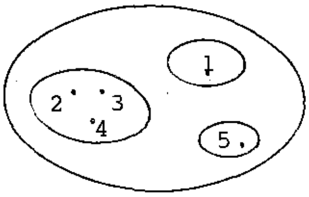
The graphical representation of a set (known as Venn-diagrams) works as follows: A set is represented by a circular plane area and the atoms in the set are written within the area, with one dot for each atom.

Example

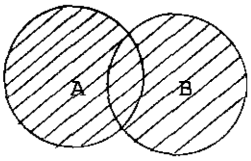
Let $S = \{1,2,3\}$ then the graphical representation is:



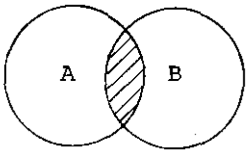
Let $S = \{\{1\}, \{2,3,4\}, \{5\}\}$ then the graphical representation is:



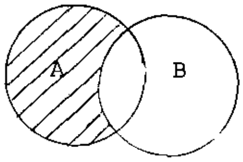
Some diagrams for operations:



A ∪ B



A ∩ B



A - B

Implementation representation

Atoms are usually stored in a coded form as strings of symbols. For (finite) sets, one normally uses list structures
to be discussed later. Infinite sets, which are recursively enumerable, can be represented by procedures that enumerate all members of the set.

Comments on the use of representation constructs of type 1.

ATOMS are used on every level of a linguistic theory where nondivisible entities are needed. This happens in two cases: (a) to represent theoretical entities, i.e. the terms of the theory, and (b) to represent observational entities , i.e. the linguistic objects from where an investigation starts. Depending on the level on which the observations take place, words, morphemes, phonemes, characters, etc; are considered to be atoms.

SETS are used to define the grouping of theoretical or observational entities into various classes. However, to represent in an interesting way nontrivial linguistic insights, more complex representation constructs will be needed. Nevertheless set theory forms the ultimate basis for all structures that we will discuss, even the most complicated ones.

Further references

There is a lot more to say about sets, but we will not give any more details, partly because we assume them well known, partly because all details can be found in the mathematical literature about set theory. The reader interested in a more tutorial account is referred to Lipschutz (1964). For a more linguistically oriented introduction, see Wall (1972). An axiomatic treatment of set theory can e.g. be found in Fraenkel, et. al. (1973).

TYPE 2

Level 1: n-tuples

Suppose now that we take a number of atoms and arrange them in a linear order. What we obtain then is an ordered pair, triple, quadruple, etc; or in general an n-tuple, with n the number of atoms.

Definition

An n-tuple (or array of length n) is an ordered sequence of n atoms. Notation: Let $a_1, \ldots, a_n$ $(n \geqslant 0)$ be atoms, then we say that $\langle a_1, \ldots, a_n \rangle$ is an n-tuple.

Definition

The zero-tuple is an n-tuple with no atoms.

Definition

Two n-tuples $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_m \rangle$ are equal if and only if $n = m$ and $a_1 = b_1, \ldots, a_m = b_m$.

Recall that for sets, an element could itself be a set; in the same manner we now introduce n-tuples of which one of the atoms is itself an n-tuple.

Definition

An m-dimensional array of length n is an m-tuple where each element is an n-tuple.

Level 2: Relations and functions

Now we will discuss several methods of defining sets of n-tuples, and of defining various types of sets of n-tuples. This subject is treated in great detail by set theory. We will therefore indicate only those aspects that are relevant for our purpose. Besides, we will pay great attention to graphs, being the graphical representation of constructs on this level, and lists which form the basis for the implementation representation.

The most general definition of a set of n-tuples is by the so-called Cartesian product:

Definition

Let S1 and S2 be two sets, then the Cartesian product (or product set) of S1 and S2, denoted as S1 X S2, is the set of all pairs $\langle$ x1,x2 $\rangle$ with

$x1 \in S1$ and $x2 \in S2$. The Cartesian product is generalized over n sets in the obvious way.

Example

Let $S1 = \{1,3,2\}$ and $S2 = \{2,3,4\}$ then

$$S1 \times S2 = \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 2,2\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 3,3\rangle, \langle 3,4\rangle\}$$

The first more restricted notion is that of a relation.

Definition

A relation R from S1 to S2 is a subset of S1 x S2.

$D = \{x1 \mid x1 \in S1$ and $\langle x1,x2\rangle \in R\}$ is the domain, and

$R = \{x2 \mid x2 \in S2$ and $\langle x1,x2\rangle \in R\}$ is the range of the relation.

Example

For S1 and S2 from the previous example: $R = \{\langle 1,2\rangle, \langle 1,4\rangle, \langle 2,3\rangle, \langle 2,4\rangle\}$ is a relation.

$\{1,2\}$ is the domain and $\{2,4\}$ the range.

Definition

A relation $R1 \subset S1 \times S2$ is equal to a relation $R2 \subset S3 \times S4$ if and only if $S1 = S3$, $S2 = S4$ and $R1 = R2$.

Definition

A relation R is empty if and only if $R = \emptyset$.

Definition

$R^{-1} = \{\langle x,y\rangle \mid \langle y,x\rangle \in R\}$ is the inverse of R.

Definition

Types of relations. Let R be relations on a set S, then

R is reflexive if $(\forall x)\ (\langle x,x\rangle \in R)$

R is symmetric if $(\forall x)\ (\langle x,y\rangle \in R \rightarrow \langle y,x\rangle \in R)$

R is transitive if $(\forall x)\ ((\langle x,y\rangle \in R$ and $\langle y,z\rangle \in R) \rightarrow \langle x,z\rangle \in R)$

R is an equivalence relation if R is reflexive, symmetric and transitive.

Convention: We often say that xRy if $\langle x,y\rangle \in R$.

Definition

Operations on Relations. The k-fold product of a relation R (on a set S) denoted as R^k is defined as follows:

(i) $\langle x,y\rangle \in R^1$ iff $\langle x,y\rangle \in R$

(ii) $\langle x,y\rangle \in R^i$ if there is a z in S such that $\langle x,z\rangle \in R$ and $\langle z,y\rangle \in R^{i-1}$ for $i \geqslant 1$.

In general, the transitive closure of a relation R on a set S, denoted as R^+ is defined iff xR^iy for some $i > 1$, and the reflexive and transitive closure is defined iff xRx and xR^+y for all x,y in S.

Example

The reflexive and transitive closure of a relation is of great importance in defining languages in a formal way. We will see examples later on.

Further restrictions bring us to the notion of a function.

Definition

A function f of S1 into S2 is a subset of S1 x S2 in which x1 $\in$ S1 appears in only one pair belonging to f, we say that f: S1 → S2 or that f(x1) = x2 for x1 $\in$ S1 and x2 $\in$S2

Example

For S1 and S2 from the previous example, we construct a function: f: S1 → S2, defined by the set: $\{\langle 1,2\rangle , \langle 2,4\rangle\}$.

Definition

A function f1: S1 → S2 is equal to a function f2: S3 → S4 if and only if, S1 = S3, S2 = S4, and for every element x1 $\in$ S1, $f_1 (x_1) = f_2(x_1)$.

Some more definitions:

Definition:

A function f is partial if there is at least one a $\in$ S1 for which f: S1 → S2 is undefined. If there is no such an a, f is total.
If f: S1 → S2 is a function and for each x $\in$ S2 there is at most one y $\in$ S1, such that f(y) = x, we say f is a one-to-one mapping.
If moreover f is a total function and f is a one-to-one mapping, we say that f is a one-to-one correspondence.

Instead of defining in more detail the various types of functions and relations, we now concentrate on the graphical representations of relations in terms of graphs.
Graphs, and a particular sort of graphs namely trees, have an important place in the theory of linguistic representations.

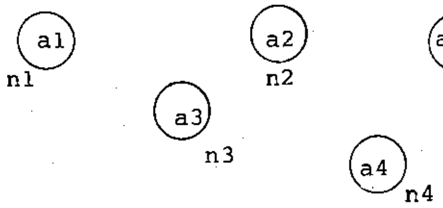
Graphical representation

Intuitive introduction:

Let R be a set of ordered pairs over a set V:

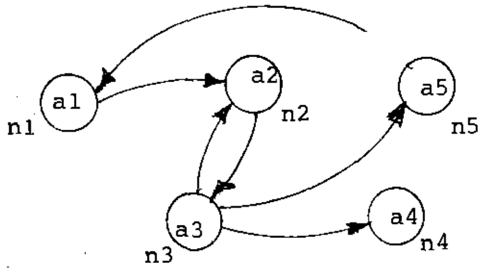
R = {⟨ a1, a2 ⟩ , ⟨ a3, a4 ⟩ , ⟨ a3, a2 ⟩ , ⟨ a3, a5 ⟩ , ⟨ a5, a1 ⟩ , ⟨ a2, a3 ⟩}

Let us now associated with each atom in V a node, draw circles for each node and put the atom in it;

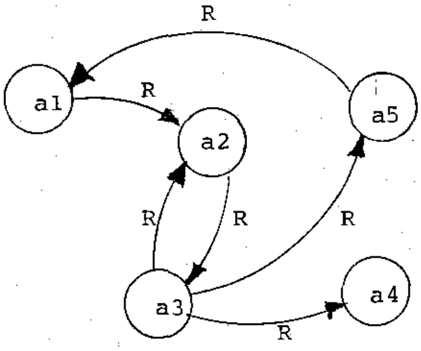


For ease of representation we will often omit n1, ..., n_n, and only represent the circles with the atoms in them.

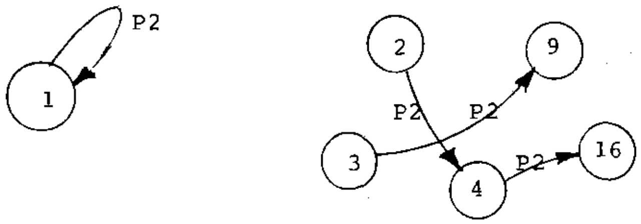
Now let us connect two nodes ni,nj with a directed arc if and only if the atoms ai,aj associated with ni,nj are in the set of ordered pairs. For the above example this results in the following:



Finally we associate labels with each arc denoting the relation, this is useful e.g. if more than one relation is represented in the same graph:

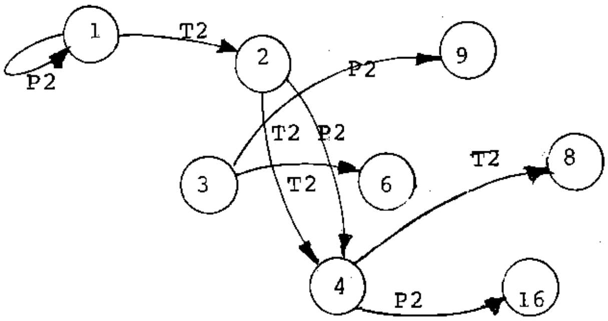


Here is another example:
Let us consider the relation 'to the second power' (denoted as P2) for the set of natural numbers from 1 to 4.
Let $\mathbb{N} = \{1,2, \ldots, 16\}$ then $P2 \subseteq \mathbb{N} \times \mathbb{N}$ and
$P2 = \{\langle 1,1 \rangle, \langle 2,4 \rangle, \langle 3,9 \rangle, \langle 4,16 \rangle\}$ because $1^2 = 1, 2^2 = 4, 3^2 = 9$; etc.
The graph of this relation (which is by the way a function because each atom on the left of the ordered pair occurs only once in such a position):



(We leave out all irrelevant nodes for other natural numbers)

Now let us consider a second relation, e.g. 'two times' (denoted as T2) for the same set $\mathbb{N}$. $T2 = \{\langle 1,2 \rangle, \langle 2,4 \rangle, \langle 3,6 \rangle, \langle 4,8 \rangle\}$
We represent T2 in the same graph and obtain:



The diagrams discussed in the previous paragraph are called directed labelled graphs.

Definition

A directed labelled graph (or DLG) is defined by a 6-tuple:
$G = \langle V,A,L,R,\phi,\psi \rangle$ and V is a finite set of nodes, A is a finite set of arcs, L is a finite set of labels for the nodes, R is a finite set of labels for the arcs, $\phi : V \times V \rightarrow A \times R$ and $\psi : V \rightarrow L$.

In language applications a directed labelled graph is normally called a network.

Some more concepts around graphs that we will need further on:

Definition

If an arc leaves a node n1 and enters another node n2, we say that n1 is the parent of n2, and n2 is the successor of n1.
If a node has no successors it is said to be terminal, else it is nonterminal.

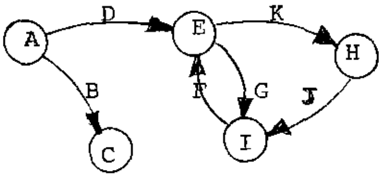
Definition

A subgraph of a graph is a subset of the nodes in the graph, together with the arcs between the nodes in the subset.

Definition

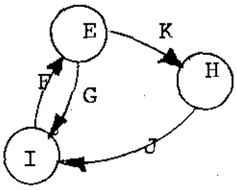
A sequence of arcs and nodes leaving from a given node A to a given node B is called a path from A to B.

Example



graph 1

A successor of the node with label A is E. The node with label A is a parent of the node with label E.
The node with label C is a terminal node.
The node with label E is a nonterminal node.



is a subgraph of graph 1

$E \xrightarrow{K} H \xrightarrow{J} I \xrightarrow{F} E \xrightarrow{G} I$ is a path

Definition

We say that a path is a circuit if the same node occurs more than once in the path.

Example

The path given in the previous example is a circuit.
Convention: We say that a graph has circuits if it is possible to construct circuits in the graph.

Now we discuss a graph of great importance in linguistic theory:

Definition

A tree is a graph with the following special properties:

(i) There is exactly one node in the tree which has no parents. This node is the root or topnode of the tree.

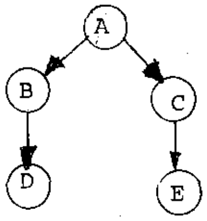
(ii) There is a path from this root to any other node in the tree.

(iii) The tree has no circuits.

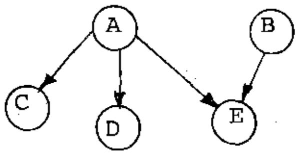
(iv) There are no arcs in the tree which cross each other.

Example

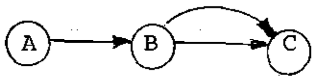
Example of a tree:



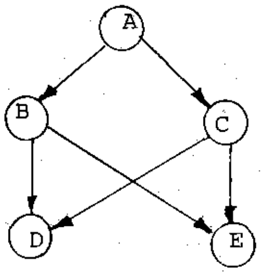
Examples of graphs which are not trees



This example violates property 1 because there is no unique topnode, also it violates property 2.



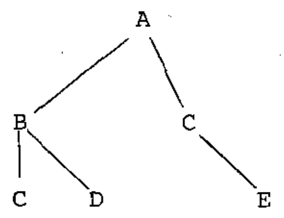
This example violates property 3, because one can construct a circuit where B occurs more than once.



This example violates property 4.

Normally the circles are left out in a tree representation, and also the labels on the arcs if only one relation is represented.
If the labels for the arcs are not left out, they are often called selectors.

Example of a tree without circles and labels:



Additional convention: terminal nodes in a tree are often called the leaves of the tree.

Now we discuss two kinds of extensions:

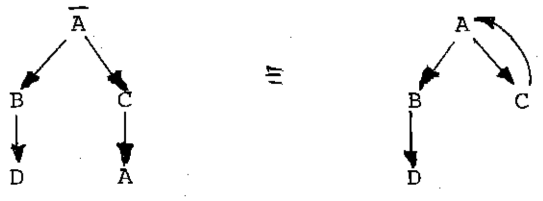
(a) an extension of trees in the sense that 'variable nodes' are introduced which stand for whole trees

(b) an extension of graphs in the sense that relations are introduced which are themselves complete graphs.

Extending tree representations

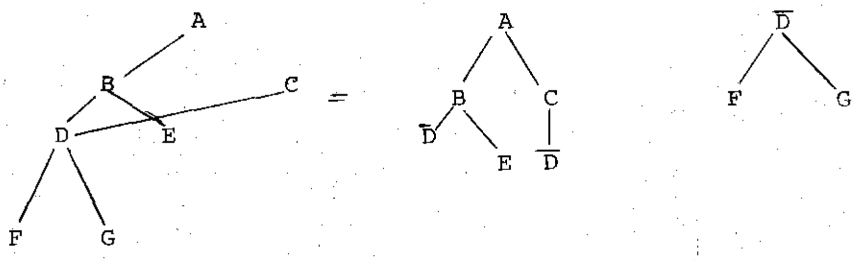
Now we define an extension of trees in such a way that we can represent circuits in a tree and that we can somehow use the idea of disconnected graphs to obtain more economic representations. The extension consists in the introduction of nodes which are given the status of variables by the fact that they represent the whole tree depending from the node. We denote a variable node by putting a bar on the label.

Example

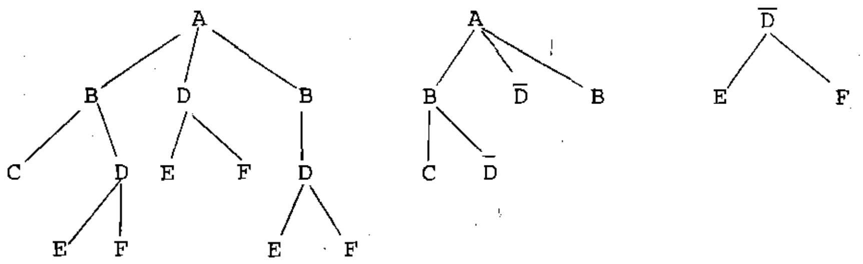


It should be clear that by this mechanism we can represent a graph as a finite tree, where it would otherwise be impossible.

Another use of variable nodes lies in isolating a subtree which occurs more than once in the representation construct. In this way we can construct subtrees which would normally not be accepted because the arcs crossing.



Also we can use variable nodes to obtain a more economical representation:



Note that we have qua representation a disconnected graph, but that there is theoretically a connection due to the variable node D.

We conclude with a definition of recursive trees.

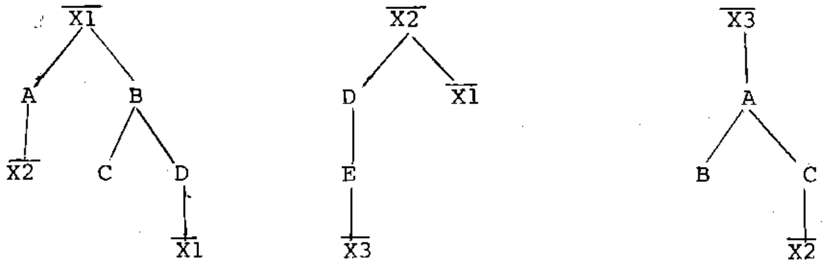
Definition

A recursive tree is a collection of trees where some nodes, called the variable nodes are used to denote the tree depending from this node. A variable node occurs either on the top of the tree, in this case the variable node is given the value of the tree of which it is a topnode, or as a terminal node in a tree, in this case it is to be replaced by its value.

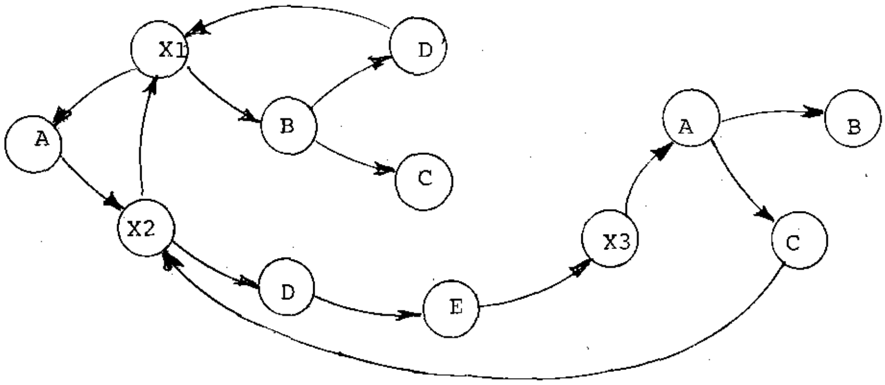
Remark:
In a well-formed recursive tree, every variable node has no more than one but at least one depending tree.

To see clearly that recursive trees have the power of graphs, we give a final example:

Example



equivalent graph:



Extending graphs

There exist several possibilities of extending networks or directed labelled graphs. Only one extension will be of interest for our purposes: recursive graphs or recursive networks.

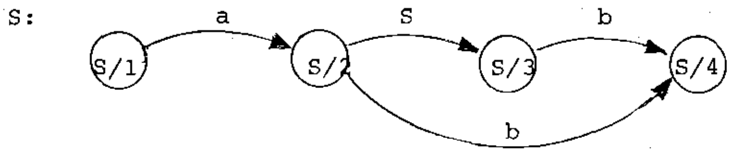
We have seen that a graph is a representation construct representating a complex of relations between several atoms. Suppose now that we consider such a whole representation construct as one complex relation which may act itself as a label on an arc in a graph.

We represent this by introducing variables for a whole graph (or network).

Definition

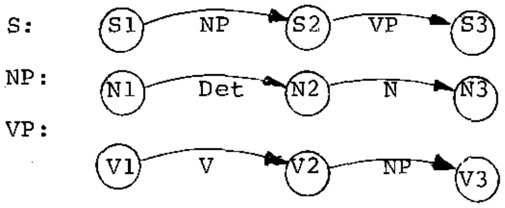
A recursive graph is a set of directed labelled graphs, with a label for each graph. The label may act itself as the label of an arc in the same or another graph.

Example



Note that S occurs itself as label of the arc from S/2 to S/3.

Here is another example



So far we discussed the basic representation of type 2 constructs and their graphical representation. Now we turn to the third aspect: the implementation representation. This implementation representation is of great importance, we therefore introduce the subject in considerable detail.

Implementation representation

introduction: list structures

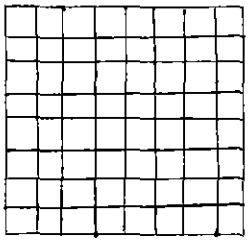
Definition

A data structure is (i) a set of cells, which can contain a certain datum and (ii) a relation among the cells: a way of organizing them.

Example

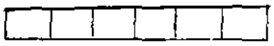
Some data structures are

a table:



or

a linear array

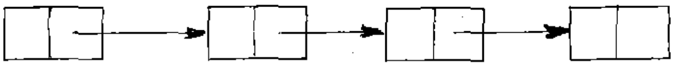


In these two data structures the location of the different cells of the data structure is defined in an implicit way, namely on the basis of the horizontal or vertical order. We can retrieve a value in one of the cells by addressing the position the cell takes in the data structure.

Suppose now that we make the structure explicit by drawing arrows if two cells are linked with each other:

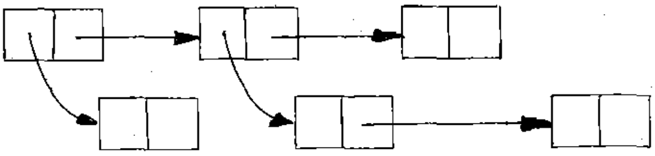
E.g.:

a)



or

b)

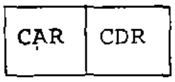


Definition

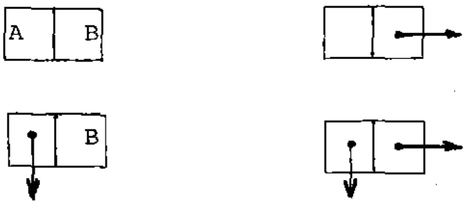
A data structure where the relations between the cells are made explicit by drawing arrows between them is a list structure.

To locate a data cell in the structure, we have to 'walk through it', until we come to the desired place.

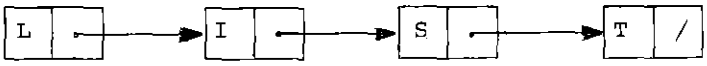
From the diagrams it could be seen that in an explicitly linked data structure a cell contains two parts. These two parts are known as the CAR and the CDR (pronouns 'cudder') of the cell:



A CDR- or CAR-field contains either a data item or another pointer, i.e. a link to another cell.
Compare:



Datums are considered to be nondivisible entities, in other words they are atoms. A list is a number of cells linked onto each other by their respective pointers.
E.g.:



Note the slash at the end, it denotes the end of the list. Another name for the slash is NIL, denoting that there is nothing in that part of the cell. If a list contains no elements at all then it is also represented as NIL. In this case NIL is called the null list. So, if NIL is placed in a CAR- or CDR-field, then we may assume that a list without any elements is attached to this field.
(Note that NIL = null list = null-atom)

Some more concepts:

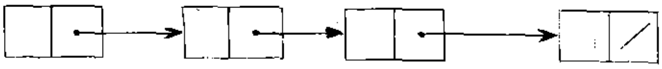
<u>Definition</u>

A list structure where every cell is linked only to its successor is a <u>one way list</u>.
A list structure where for each cell there is a link both to its successor and its predecessor, is a <u>two way list</u>.

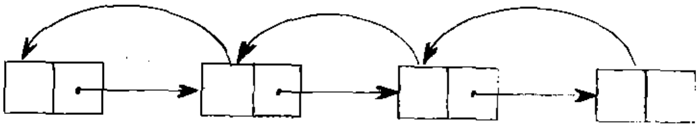
From now on we will only deal with one way lists.

Example

a) one way list



b) two way list

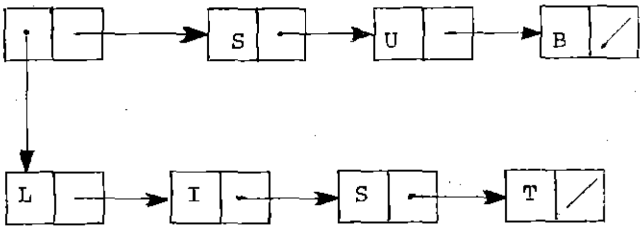


Definition

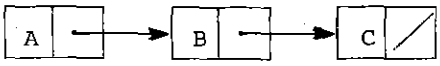
If a list contains a pointer in one of its CAR-fields, then the list starting from such a CAR-field is a sublist.
A list with sublists is called a branched list, a list without sublists is called a linear list.

Example

branched list:



linear list:



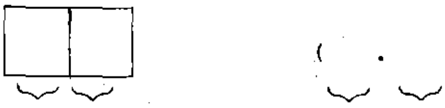
The representation problem

To have a successful data structure it is not sufficient to have a graphical representation. One must be able to write down the graphical representation in a linear way, i.e. algebraically. For tables and vectors, we do this by naming the whole data structure with a symbol (say X), and the different cells of the data structure are addressed by subscripts, e.g. X(1,2) denotes the first cell of the second column in a table called X.

For list structures the solution to the representation problem is not so easy, simply because cells cannot be addressed by subscripts on the basis of their location, i.e. by referring to lines and columns. The problem is solved by the introduction of S-expressions with two particular formats: dot-notation and list-notation.

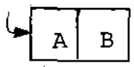
dot-notation

The dot-notation of a list structure is a direct mirror of its graphical representation. For each cell we introduce two brackets and one dot:

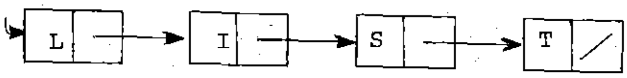


On the right side of the dot we write the CAR and on the left side the CDR. If the CAR or CDR contains a pointer, then we replace this pointer by the whole sublist depending from this pointer written in dot-notation.

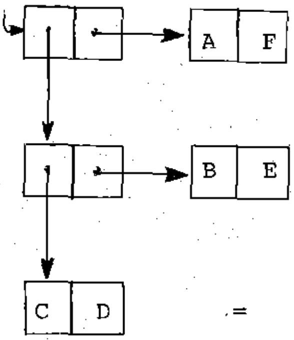
Examples



= (A . B)



= (L . (I . (S . (T . NIL))))



= (((C . D) . (B . E)) . (A . F))

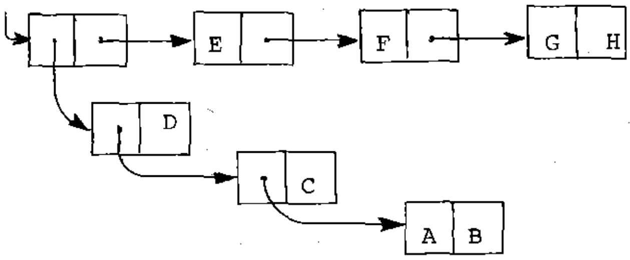
The following strategy can be followed for the construction of dot-notation from graphical structures:

(i) Consider a list to be linear and whenever a pointer appears, introduce a variable name for the sublist depending from this pointer.

(ii) Similarly construct for each sublist on the same basis a linear list with variables when necessary.

(iii) Replace all variables by their respective dot-representations.

Example:

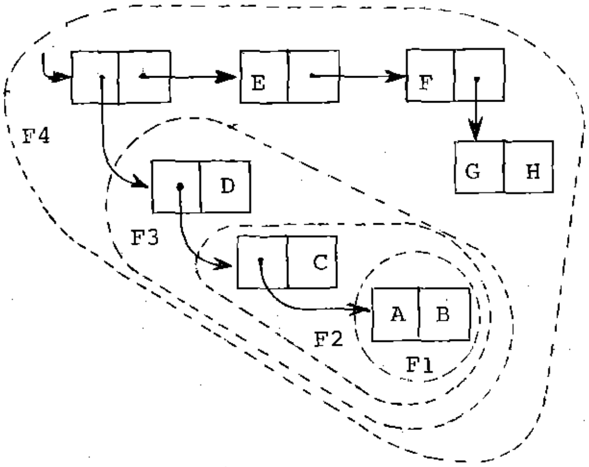


with F1 = (A . B) , F2 = ((A . B) . C)

F3 = (((A . B) . C) . D)

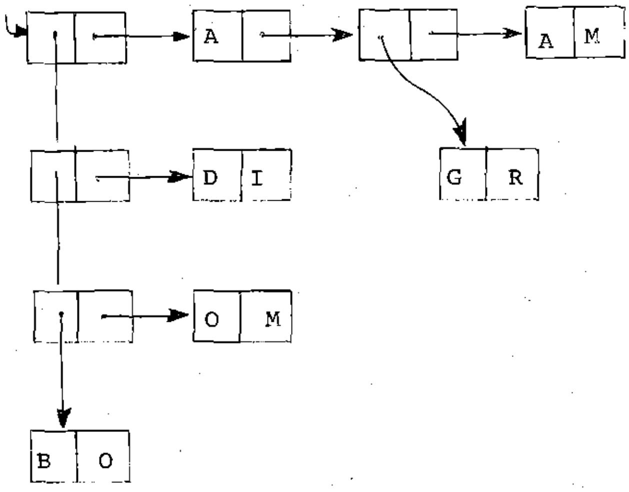
F4 = (F3 . (E . (F . (G . H))))

Graphically we had the following sublists:



The final result is: ((((A . B) . C) . D) . (E . (F . (G . H))))

Here is another example:



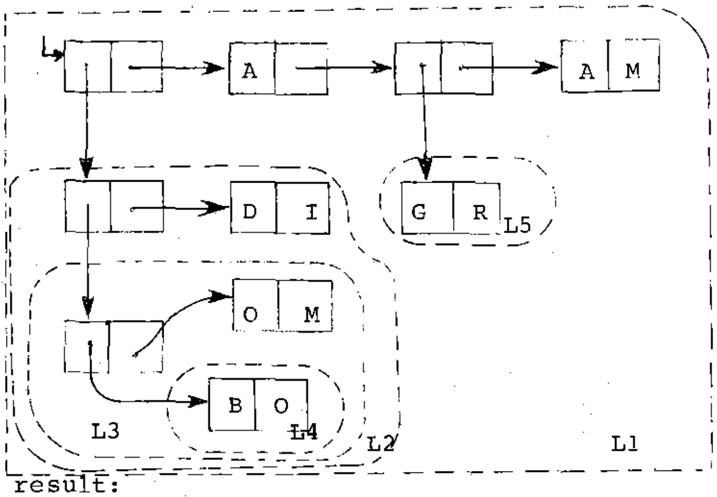
We have the following sublists:

```
L1 = ( L2  . ( A . ( L5 . ( A . M ) ) ) )
L2 = ( L3 . ( D . I ) )
L3 = (L4 . ( O . M ) )
L4 = ( B . O )
L5 = ( G . R )
```

Replacing all the variables yields:

```
L3 = ( ( B . O ) . ( O . M ) )      (L4 in L3)
L2 = ( ( ( B . O ) . ( O . M ) ) . ( D . I ) )   ( L3 in L2)
L1 = ((( ( B . O ) . ( O . M ) ) . ( D . I ) ) . ( A . ( L5 . ( A . M ) )
```

Graphically:



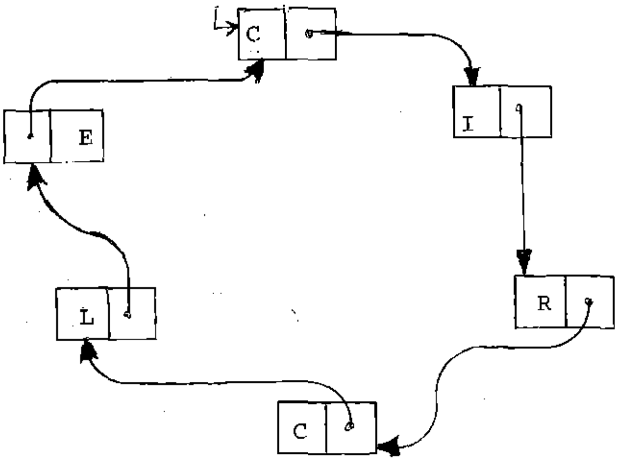
Final result:

(((B . O) . (O . M)) . (D . I)) . (A .
((G . R) . (A . M))))
(L5 in L1)

Although dot-notation is an immediate reflection of a graphical structure, there is already one sort of list structures that cannot be expressed namely a circular list.

A circular list is a list where a pointer in some field points to a previous cell of the list.

Example:



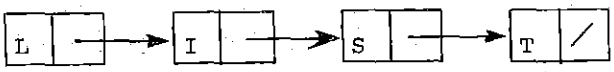
Clearly a dot-notation of the graph would never come to an end. The fact that circular lists cannot be expressed is however seldom felt as a drawback, certainly not in linguistic practice.

list-notation

Although the dot-notation of lists is a very nice way of writing graphs into a linear format, it soon becomes extraordinary complex when the list structures themselves grow. Therefore another representation has been designed: list-notation. This goes as follows

(i) A linear list is transferred by writing all the elements of the respective CAR-fields right after each other.

E.g.:



or

(L . (I . (S . (T . NIL)))) (dot-notation)

or

(L I S T) (list-notation)

(Note that nothing is provided if there is an atom in the CDR-field)

(ii) As for dot-notation , as soon as there appears a pointer to a sublist in one of the CAR-fields, construct the list-notation for this sublist and replace the pointer by this sublist.

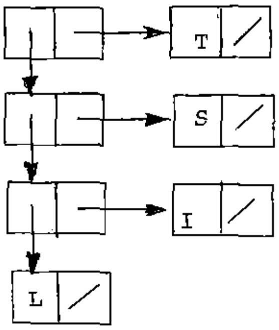
Example:

dot-notation

((((L . NIL) . (I . NIL)) . (S . NIL)) . (T . NIL))

lists

i.e.



in list-notation: ((((L) I) S) T)

The technique for constructing dot-notation from graphical structures can also be used here to construct list-notation from graphical structures:

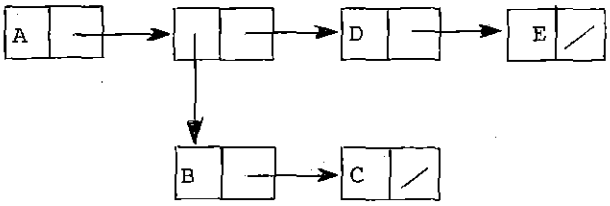
(i) Consider a list to be linear and whenever a pointer appears, introduce a variable name for the sublist depending from this pointer.

(ii) Similarly construct for each sublist on the same basis a linear list with variables when necessary.

(iii) Replace all variables by their respective list representations.

Example:

(i)



```
Let L0 =  ( A L1 D E )
    L1 =  ( B C )
then L1 in L0 yields:    ( A ( B C ) D E )
```

(ii) Given the following list (in dot-notation):

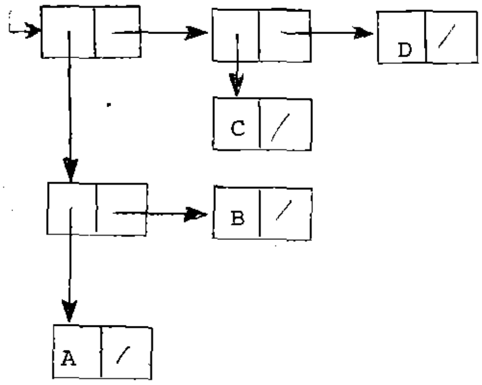
```
( ( ( A . NIL ) . ( B . NIL ) ) . (( C . NIL ) . ( D . NIL ) ) )
```

We have the following steps:

```
L5 =  (L1 L2 D)
L1 =  ( L3 B)
L2 = ( C )
L3 =  ( A )
```

Finally: L5 = (((A) B) (C) D)

L5:



Restrictions on list-notation:

(a) It is not possible to represent circular lists

(b) Whenever an atom appears in a CDR-field we have to use dot-notation.

list-notation of trees

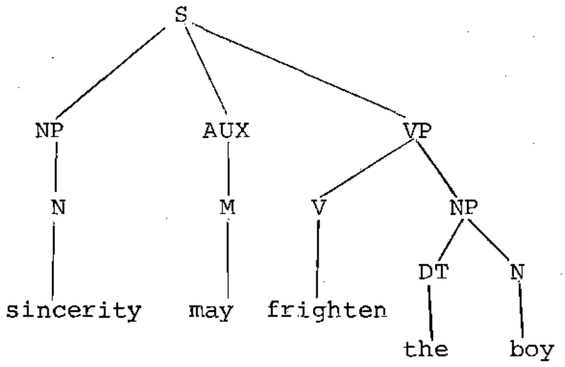
List structures are widely used in any sort of linguistic applications. There are two reasons for this

(1) easy input/output of strings and easy processing of alphanumeric data;

(2) (and this is even more important) easy representation of structures.

The latter point will be illustrated more explicitly in this section. We will discuss the standard means of representing tree structures in terms of list structures. The reader should not proceed without being thoroughly familiar with this. The representation of linguistic information will depend crucially on it.

A typical tree looks as follows:



An alternative linear representation of the same information is the so called labelled bracketing:

$$(\ (\ (\ \text{SINCERITY})_{N})_{NP}\quad (\ (\ \text{MAY})_{M}\ )_{AUX}$$

$$(\ (\ \text{FRIGHTEN})_{V}\ (\ (\ \text{THE})_{DT}\ (\text{BOY})_{N}\ )_{NP})_{VP})_{S}$$

lists

OR

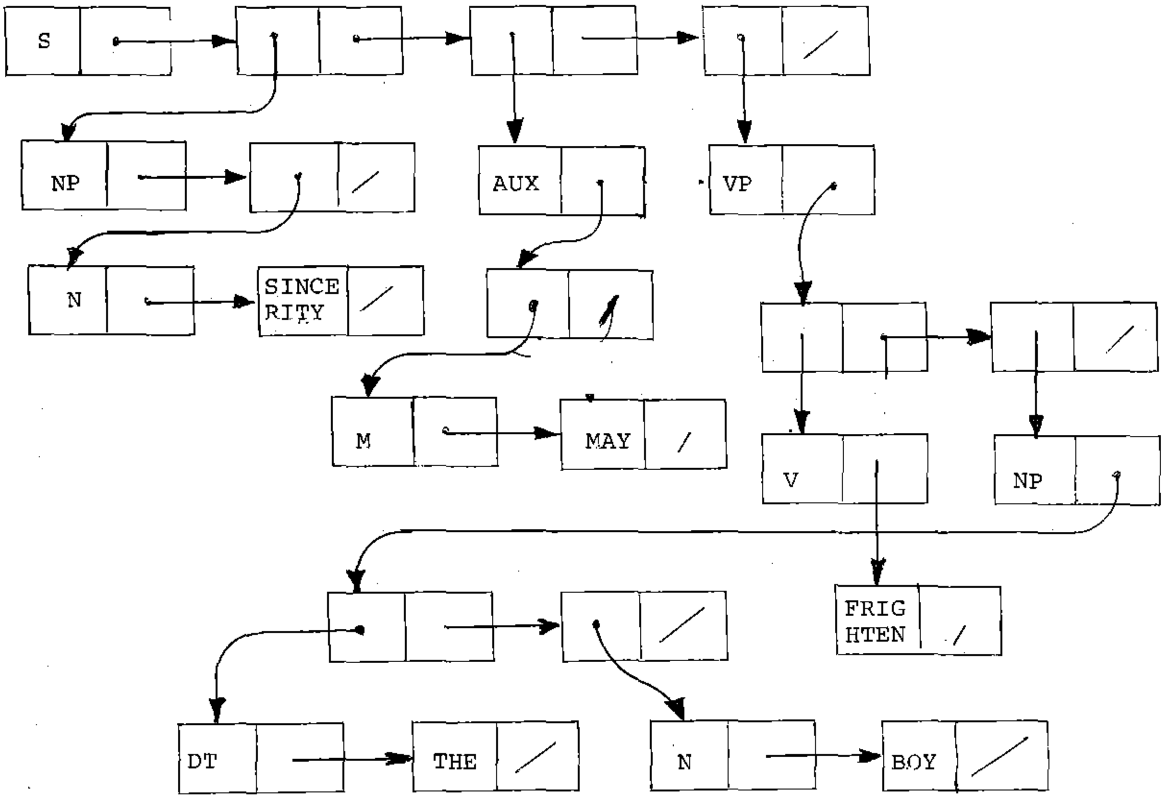
$(_S (_{NP} (_N$ SINCERITY)) ($_{AUX}$ $(_M$ MAY))

$(_{VP}$ ($_V$ FRIGHTEN) ($_{NP} (_{DT}$ THE) $(_N$ BOY))))

respectively called right labelled bracketing and left labelled bracketing. Now, if we take the left labelled bracketing and write all symbols on one line we obtain:

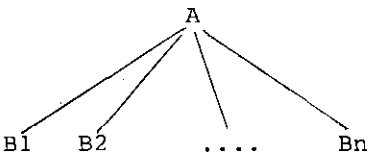
(S (NP (N SINCERITY)) (AUX (M MAY)) (VP (V FRIGHTEN) (NP (DT THE) (N BOY))))

and this is nothing else but the list-notation of a list structure, the graphical representation of it being:



Because of the importance of the relation between trees in graphical and list-notation we define explicitly the relationships between the wo:

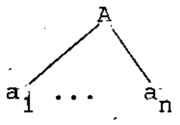
(i) Given a tree structure



with **A**,B1, ..., Bn nonterminal nodes, then the equivalent list-notation is:

(A (B1 ...) (B2 ...) ... (Bn ...))

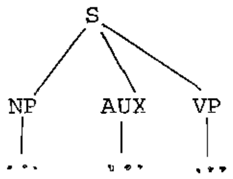
(ii) Given a tree structure



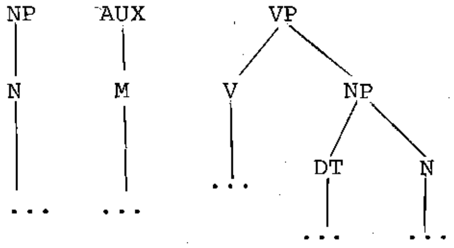
with A a nonterminal node and a_1, ... , a_n atoms

then the equivalent list-notation is (A a_1 ... a_n)

Example:



(S (NP ...) (AUX ...) (VP ...))



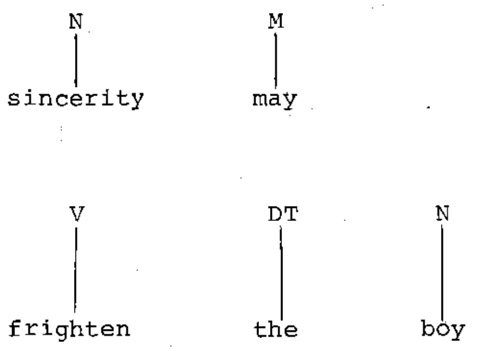
(S (NP (N ...)) (AUX (M ...))

(VP (V ...) (NP (DT ...)

(N ...))))

lists

finally:



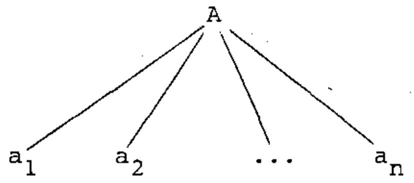
(S (NP (N sincerity) (AUX (M may))
(VP (V may) (NP (DT the) (N boy))))

reverse:

Given a list (A a_1 a_2 ... a_n)

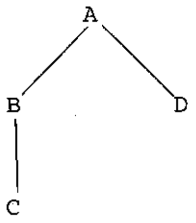
with a_1, ... ,a_n sublists or atoms, then

the equivalent tree is



Example:

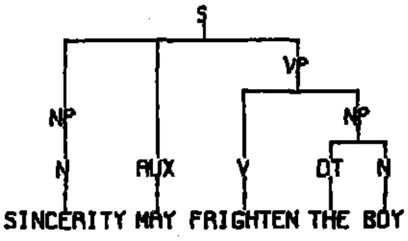
Given (A (B C) D) the equivalent tree is:



lists -

Comment

We stress the importance of the relation between trees (= graphs) and the equivalent list-notations. Due to this importance the author (in coopreation with P. Reypens) took the pain of constructing computer programs that given a list-notation of a tree, automatically plots the graph structure of it. The output has the following format:



lists

Comments on the use of representation constructs of type 2

N-TUPLES are used in the definition of formal systems. All components of the system are given a name and these names are grouped in an n-tuple. The definition of a formal grammar is an obvious example of this, see e.g. also the definition of a directed labelled graph, already given.

RELATIONS are used for two purposes:

(a) The representation of linguistic information structures which are produced or processed by the language systems, examples are structural descriptions in the form of a tree as result of a syntactic analysis, semantic structures, etc.

(b) The representation of linguistic data upon which the language systems operate. Examples are semantic networks, recursive transition networks, etc.

Further references

For the mathematical aspects of relations and functions, we refer to the relevant mathematical literature about the subject. A tutorial account is found in Lipschutz (1964). A more linguistically oriented introduction in Wall (1972). A nice introduction to graphstructures can be found in Gavare (1972). An introduction to list structures can be found in any textbook on the programming language LISP. E.g. Weissman (1968). A formal treatment of list structures is presented in Guha and Yeh (1976).

TYPE 3

Level 1: Strings

Another representation construct that is of considerable interest in linguistics is the concept of a string.

Definition

A string is a finite sequence of occurrences of atoms.

Notation: Let a1,a2,a3 ... be atoms, then a1a2a3 ... is a string.

Definition

The null string, denoted as λ , is a string without any elements.

A useful operation is that of concatenation.

Definition

Let α and β be strings with $\alpha = a_1 \ldots a_n$ and $\beta = b_1 \ldots b_m$ then the concatenation of α and β , denoted as $\alpha\beta$ (or $\alpha . \beta$ or $\alpha \frown \beta$) is $a_1 \ldots a_n b_1 \ldots b_m$.

Definition

A string α is a substring of a string β , if $\beta = \gamma\alpha\delta$ for γ, δ possibly empty strings.

Example

Let abc be a string then $\{\lambda$, a , b , c , ab , bc , abc , ac$\}$ is the set of all substrings of abc.

Definition

The reversal of a string σ , denoted as σ^R is a string in reverse order, i.e. let $\sigma = a_1a_2 \ldots a_n$ then $\sigma^R = a_n \ldots a_2a_1$.

Definition

The length of a string σ , denoted as $|\sigma|$ is the number of atoms in σ .

Note that n-tuples as elements of an n-tuple are in comparison to substrings of a string what sets as elements of a set are compared to subsets of a set. This is exactly the difference between n-tuples and strings.

Level 2: Languages

We now generalize over strings by considering ways of definings sets of strings, called languages.
The most general way of doing so, is by considering a language to be a subset of the set of all strings over a given alphabet V.

Definition

Let V be a finite set of atoms, called an alphabet, then V^+ is the set of all strings over V, and $V^\star = V^+ \cup \{\lambda\}$

The statement that a language L is a subset of the set of all strings over its alphabet is a rather trivial statement. We want to have ways to define more exactly what elements there are in the language. As languages tend to be infinite, we should find a finite representation of this infiniteness. The solution to this problem is a system called a formal grammar.

Definition

A formal grammar is defined by a quadruple G $=\langle Vn, Vt, P, S \rangle$

where Vn, Vt are finite , nonempty disjoint sets of nonterminals and terminal symbols respectively, $V = Vn \cup Vt$.

$P \subseteq V^\star \times V^\star$ is the set of rewriting rules. We say that $\alpha \rightarrow \beta$ if $\langle \alpha , \beta \rangle \in P$

$S \in Vn$ is the start symbol or initial symbol.

Definition

The derivation relation denoted as $\Rightarrow$ holds between two strings α , β if and only if $\alpha \rightarrow \beta \in P$

$\overset{\star}{\Rightarrow}$ is the reflexive and transitive closure over $\Rightarrow$.

Definition

A language defined by a grammar G, denoted as L(G) is defined as the set $\{x \mid S \overset{\star}{\Rightarrow} x, x \in Vt\}$

Conventions

As we will use formal grammars sometimes, it is important to keep the following notational conventions for productions in mind:

1. Nonterminals will always be written between angular brackets $\langle \rangle$
2. When two possibilities occur at the right of the arrow, we write braces: $\{ \}$

e.g. $\langle a \rangle \rightarrow \begin{Bmatrix} b \\ c \end{Bmatrix}$

3. When a substring on the right of the arrow is optional, we will write it between straight brackets:

e.g. $\langle a \rangle \rightarrow a[b\ c]d$

is a shorter version of

$$\langle a \rangle \rightarrow \left\{ \begin{array}{l} a\ d \\ a\ b\ c\ d \end{array} \right\}$$

Example

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

Some derivations:

$\langle S \rangle \Rightarrow a \langle S \rangle b \Rightarrow a\ a \langle S \rangle b\ b \Rightarrow a\ a\ a\ b\ b\ b$

$\langle S \rangle \Rightarrow a$

The language generated is $\{a^n b^n \ , \ n \geqslant 1\}$

It is well known that there exists a hierarchy of grammars (the Chomsky hierarchy) which is defined on the basis of the formal outlook of the rules:

Definition

type	form of rule
3 (regular)	$\langle S \rangle \rightarrow a$ or $\langle S \rangle \rightarrow a \langle S \rangle$ for $a \in Vt$, $S \in Vn$
2 (context-free)	$S \rightarrow \gamma$ for $S \in Vn$, $\gamma \in (Vn \cup Vt)^+$
1 (context-sensitive)	$\beta \rightarrow \gamma$ $\beta = ..\langle A \rangle.. \in (Vn \cup Vt)^+$, $\gamma \in (Vn \cup Vt)^+$
0 (unrestricted)	$\beta \rightarrow \gamma$ for $\beta, \gamma \in (Vn \cup Vt)^\star$

It is equally well known that a different sort of power (i.e. type of language) corresponds to each type of grammar and that each type of language has certain distinct mathematical properties. Details and other mathematical results can be found in any textbook on formal language theory.

Contrary to what may be assumed, we will NOT use formal grammars of the kind just defined in the definition or recognition of natural languages. The reason for this is that the formalism is toosimple to account for the complexity of natural language. We will however use formal grammars to define representation languages on the various level of a linguistic theory, in particular we will use type 2 grammars because the languages generated by these grammars are sufficiently complex but at the same time not so complex that a recognition and processing of expressions from the language becomes difficult.

Formal grammars as definition of representation languages

Already right from the moment of conception of phrase structure grammars, it was felt that they should not only be used for the definition of linear languages but also for the definition of structures, i.e. representation languages. The two usages are given different names: strong and weak generative capacity.

Definition

The weak generative capacity is the set of terminal strings generated by the grammar. The strong generative capacity is the set of structural descriptions assigned by the grammar to these strings.

For completeness, we indicate here briefly the well known method for assigning structural descriptions to strings.

Algorithm:

Let $G = \langle Vn, Vt, P, \langle S \rangle \rangle$ be a cf-grammar, then

(i) We introduce a node for $\langle S \rangle$, the topnode of the tree

(ii) Whenever we use the rule $\langle A \rangle \rightarrow \gamma$ in a derivation with $\langle A \rangle \in Vn$ and $\gamma \in (Vn \cup Vt)^{+}$, $\gamma = a_1 \ldots a_n$ $n \geqslant 1$. We introduce new nodes for each a_i, $1 \leqslant i \leqslant n$ in γ and connect these new nodes with a line from $\langle A \rangle$ to a_i.

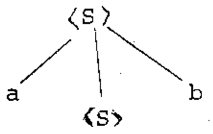
Example

Let $G = \langle \{\langle S \rangle, a, b\} \{\langle S \rangle \rightarrow a\ b, \langle S \rangle \rightarrow a\ \langle S \rangle\ b\}, \langle S \rangle \rangle$ be a cf-grammar

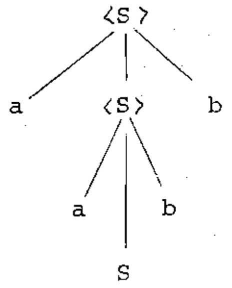
derivation tree

$\langle S \rangle$ $\langle S \rangle$

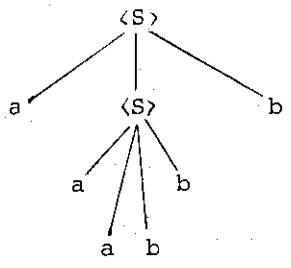
$\langle S \rangle \Rightarrow a\ \langle S \rangle\ b$



$a \langle S \rangle b \Rightarrow a\ a \langle S \rangle b\ b$



$a\ a \langle S \rangle b\ b \Rightarrow a\ a\ a\ b\ b\ b$

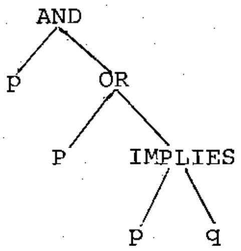


It should be clear that the relation between derivations and trees is not a 1 - 1 mapping, because the information about the order of rule application is lost in a tree representation

This method of definition, which we will call derivationally controlled tree construction, is widespread in linguistics. It is actually the only method being used to define representation languages. However there are some strong restrictions on its use. Let us consider them briefly and after that try to find a new method of defining representation languages with formal grammars.

The main 'bad' point about the derivationally controlled method of tree construction is that 'open' situations are only allowed to appear at the terminal nodes of the tree.

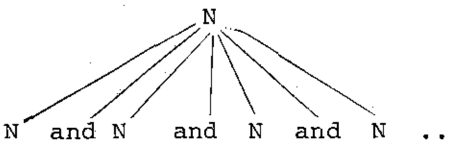
Suppose the following representation for a simple propositional calculus expression:



It is easy to see that there corresponds no straightforward context-free grammar that would generate these structures. The reason for this is that the operators form open classes, i.e. classes where there is more than one member but where all members have the same possible position and function in the expression. These operators occur in the nonterminal part of the tree and therefore we cannot generate them.

Another illustration of the restrictedness of the strong generative capacity of cf-grammars is the representation of coordination (see Lyons,1968,221):

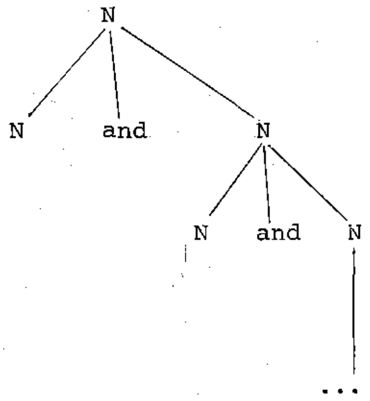
A structure as



where the number of (AND N) nodes is open, cannot be generated by a cf-grammar. The only way to obtain a similar structure is by the recursive rule:

N → N [and N]

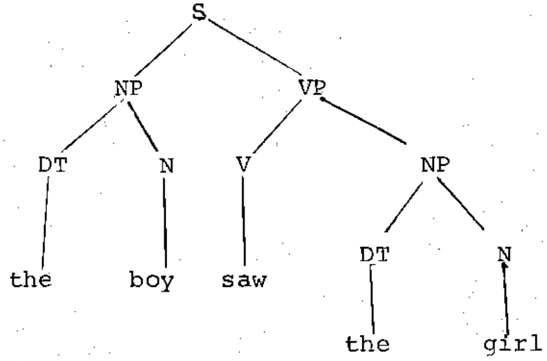
This however leads to



which is not quite the representation of coordination that we wanted to have.
Again a cf-grammar cannot represent this kind of structures because an open situation occurs in the nonterminal part of the structure.

There is however another means of defining representation constructs by formal grammars: The method consists in (i) defining an equivalent symbolic expression for each tree, (ii) defining a grammar which generates such symbolic expressions, (iii) converting the generated expression into a tree. We call this method the <u>indirect tree construction method</u>.

As symbolic equivalent of a tree we will use the list-notation (already discussed), e.g. given the tree:



we have

```
        (S    (NP     (DT  the)  (N boy) )
              (VP     (V   saw)   (NP ( DT the) (N girl))))
```

as list-notation (or left labelled bracketing)

Let us now define a grammar for the propositional calculus example.

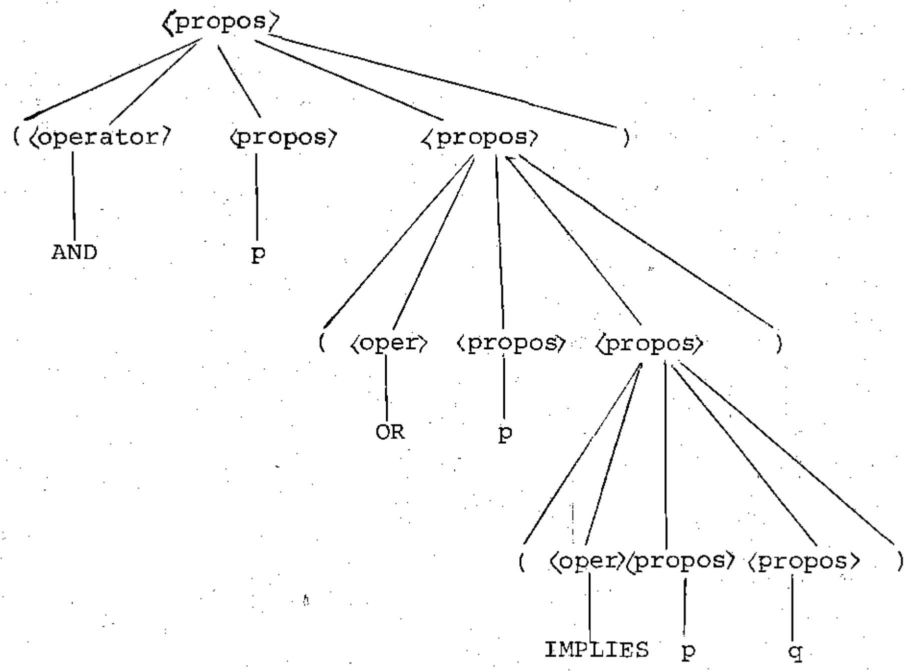
⟨propos⟩ → (⟨operator⟩ ⟨propos⟩ ⟨propos⟩)

⟨operator⟩ → AND, OR, ...

⟨propos⟩ → p, q, ...

(note that the brackets are terminal symbols !!!)

Structure from the derivation:

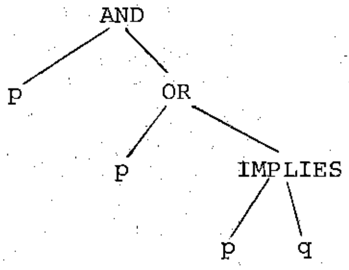

⟨propos⟩
(⟨operator⟩ ⟨propos⟩ ⟨propos⟩)
AND
p
(⟨oper⟩ ⟨propos⟩ ⟨propos⟩)
OR
p
(⟨oper⟩⟨propos⟩ ⟨propos⟩)
IMPLIES
p
q

(structure 1)

the resulting terminal string:

(AND p (OR p (IMPLIES p q)))

if we consider this as a list we get the following tree:



(structure 2)

and this is exactly what we wanted to have.

All the syntactic load in structure 1 is absent from structure 2 but we do have the structure on which the interpretation can work.

Now we discuss the grammar for the coordination example:

⟨nomen⟩ → ⟨N⟩ ⟨co-N's⟩

⟨co-N's⟩ → ⟨N⟩

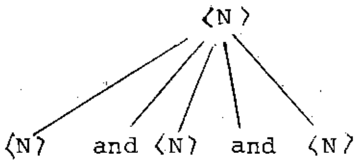
⟨co-N's⟩ → ⟨N⟩ and ⟨co-N's⟩

derivation

⟨nomen⟩ ⇒ (⟨N⟩ and ⟨co-N's⟩) ⇒ (⟨ N⟩ and ⟨N⟩and ⟨co-N's⟩)

⇒ (⟨N⟩ and⟨ N⟩ and ⟨N⟩ and ⟨ N⟩)

This final string represent the following structure:



It is easy to see that we could have as many N's as necessary while preserving the structure in which the coordination should be represented.

Summarizing, formal grammars can be used to define representation languages in two ways (i) either directly by a straightforward mapping from the derivation sequence to a tree structure, (ii) or indirectly by considering the language generated by the grammar as symbolic linear representations of tree structures. The first method has the disadvantage that certain types of structures cannot be defined. The second method is unrestricted qua representational power. It has the additional advantage that we can define a structure of the representation construct, outside the construct itself. For these two reasons <u>we will define representation languages always by means of the second method</u>.

Discussion

In the previous subsections we have presented several representation constructs:

type 1:
    level 1: atoms: the primitive objects of the representation
    level 2: sets : an unordered collection of atoms.
type 2:
    level 1: n-tuples and lists: an ordered collection of atoms
    level 2: sets of n-tuples: relations, functions, graphs and trees
type 3:
    level 1: strings: ordered sequences of occurrences of atoms
    level 2: languages: sets of strings.

It is obvious that for each type, representation constructs on level 1 are more powerful than those on level 2. The relation between type 1 and type 2 is such that an n-tuple$\langle a_1, a_2, \ldots , a_n\rangle$ is per definitionem equivalent with the set $\{\{a_1\},\{a_1,a_2\}, .. \{a_1, \ldots a_n\}\}$.
So we see that there is no theoretical difference between the two. It is clear however that a set representation of n-tuples is more cumbersome and therefore inconvenient.

The relation between type 2 and type 3 is such that any n-tuple can be considered to be a string (but not vice-versa !) and similarly any set of n-tuples can be considered to be a language. This is an interesting observation of which we make extensive use in the indirect tree construction method for defining representation languages.

Comments on the use of representation constructs of type 3.

As we already said before formal grammars will be used to define representation constructs on every level of the theory. Note that we will always use the indirect tree construction method.
The formal grammars of the type discussed in this section have only a theoretical significance: they are a representation construct representing classes of structures. How these structures are computed and processed is another matter.

Further references

Since the formalization of p.s. grammars by Chomsky, the concept of a grammar has been fully investigated mathematically. Textbooks on the subject are Hopcroft and Ullman(1969) and Salomaa (1973). For all details about the mathematical foundations of formal grammars, the reader is invited to consult these references.

§ 0.2. Introduction to the theory of computation

As we assume no knowledge or experience with computers, we start this section by giving a very broad and intuitive introduction to the main ideas behind the operation of machines that are able to compute. We address ourselves to the question: how is it possible to design machines that are capable of doing symbolic manipulations as required for linguistic tasks. After that we define some basic concepts of computation theory such as procedure, algorithm and the like. The first part can be skipped by someone who knows about computer programming.

Intuitive introduction

(i) Coding and storing

The first principle that underlies the design and operation of a computer is that of coding. It runs as follows. First define different representations in such a way that information in one representational format can be translated into another format while the meaning (= interpretation) of the first is equal to that of the second. Then define the reverse of this relation. E.g. we have a message in natural language, we design a code for each letter of the alphabet and then we can translate each sentence in a code representation and back. Thus we can store any sort of information we want to.

Now suppose you can construct a machine that can perform on command certain operations (however simple), i.e. that can change x into y if you tell it to do so. Then you can consider the objects that the machine is capable of changing as the 'zero-language' and design a coding such that your information (in whatever representational format) is transformed into this zero-language and back. Doing so, you can instruct the machine to manipulate information via the coding.

As an illustration consider a calculator; the operations that can actually be performed are very simple: change of a state from one to zero (how this is realized does not concern us here). As 0 and 1 are the objects that can be manipulated, we construct a code with 0 and 1 for the information, which are normally numbers for a calculator. The code is called the binary code. The actions necessary to do a computation can be described as follows:

(a) Translate the input numbers in terms of 0 and 1 and store this information, i.e. change a piece in the machine in such a way that it now reflects the codings of the number,

(b) change the 0's and 1's in a particular way (= computation)

(c) and translate the result of the change back into the 'user's language' which is the decimal form.

(ii) Programming

We now have a way to represent the information inside the machine and a way to perform a simple operation over that representation. Now comes the second step. Suppose we code the operation that is to be performed also. That means a particular way of carrying out changes is given a code (in 0's and 1's), and this code is also translatable. Instead of pushing a particular button, say, for a certain operation, one can make the machine react to that code. Clearly what we have then is not only a way to represent the data (that upon which the change is being made) but also a way to represent the operation (a name for the change).

Given this idea we can even go further. As we can keep data in some way stored in our machine, we can also store the names for the operations that are to be performed. We need then a mechanism 'reading' operation names and carrying them out after each other.

This soon leads to fascinating computer power because we can construct operations that change the flow of carrying out operations on the basis of a condition, e.g. given a list of operations: oper 1, oper2,oper 3,oper 4 we could let operation3 be such that it goes back to execute operation 2 unless some condition is satisfied.

```
E.g. operation 1: store number 0 in register 17;
     operation 2: add 1 to the contents of register 17;
     operation 3: if the contents of     register 17 are less
                  than 5 go back to operation 2, else proceed;
     operation 4: stop.
```

What happens is that the machine will count until 5 and then stop. It is important to note that one starts from the first operation and takes each time the next one (this is known as a sequential manner of executing operations), except when the normal flow is changed due a control statement, such as operation 3.

A sequence of operations is called a program, the language in which the human instructor writes programs is called a programming language and this is translated into a code the machine can read (the machine or object language).

The task of the user should be clear now: he has to specify all the actions that he wants the computer to perform, and the way in which the actions are organized. In other words he has to write a program. Then he gives this program to a coding device which delivers a program in machine readable form. This then is read and executed by a machine with (hopefully) the required result. So, a program can be considered as a sophisticated way of pushing buttons that lead to machine action.

The rest of the story is one of ever growing complexity based on these main principles. An important step was to let the computer itself do the process of translating the program stated in some programming language into a program stated in the machine code. Such a process is also directed by a program and this program is called the compiler or interpreter.

Currently the result is (i) that any information whatsoever can be represented within the memory of a computer if properly coded, (ii) that any process when sufficiently explicit (that is when every single step is made clear) can be programmed and executed.

These powerful tools (powerful especially if one considers that a very high degree of complexity is allowed) are the basic means that are used in order to bring about the process of language understanding and producing and they are regarded to be sufficient for that purpose. Moreover signals from acoustic and visual analysers can be processed, again after properly being coded, which makes spoken and written language analysis possible. Also signals can be issued from the computer to acoustic synthezisers leading to speech synthesis. The normal means for input/output are however type writing.

Having presented in an intuitive manner the basic principles of doing computations by machines, we now turn to some fundamental terms used to talk about computation.

Fundamental terms

Definition

A procedure is a finite sequence of instructions which can be executed mechanically in a finite amount of time.
Normally a procedure takes some input and returns some output after executing the sequence of instructions.

Example

Input: any natural number
Output: 0 if the number is divisible by 2, else 1.

```
Procedure: Let N be the input number
   step 1: if N is 1 or 0, output N and stop, else do step 2.
   step 2: set N equal to N - 2, proceed with step 1.
```

Example of operation:

Let N be 7 then the result should be 1, which was the code for nondivisible by 2.

```
    step 1: as 7 ≠ 1 or 0, we do step 2
    step 2: N becomes 5 and we do step 1 again
    step 1 as 5 ≠ 1 or 0, we do step 2
    step 2: N becomes 3 and we do step 1 again
    step 1: as 3 ≠ 1 or 0, we do step 2
    step 2: N becomes 1 and we do step 1 again
    step 1: N = 1, so the result is 1, and that is what we
            expected.
```

Definition

A procedure is an algorithm if and only if for each input only a finite amount of instructions are carried out.

This implies at least that somewhere there must be an instruction saying 'halt' or 'stop' or 'end'. (In the example this was the case in step 1). It also implies that this instruction must be reached in a finite amount of time.

Example

The following procedure is not an algorithm

Input: Any natural number
Output: ?

Procedure:

```
    step 1: whatever the input, do step 1 again ;
    step 2: halt.
```

It is obvious that we will never reach step 2 and therefore will never stop.

At first sight it seems silly to design machines that do not necessarily stop. It was one of the great discoveries of this century however, that there is a class of problems for which there is no algorithm. The best thing we can do in such a situation is to use a procedure.

Obviously a procedure defines a function from its input to its output.

Definition

The function defined by a proecure is a partial recursive function. The function defined by an algorithm is a total recursive function.

Important for our purposes is also the idea to relate the notion of procedure and algorithm to sets and therefore to languages. We do this by means of the notion of a characteristic function: given a set S, then when we apply this characteristic function to all members of S, the function yields true, and when to all elements which are not a member of S, the function yields false. This brings us to the following definition:

Definition

If the characteristic function of a set is necessarily a partial recursive function, i.e. if there exists no algorithm for that function, only a procedure, then the set is said to be recursively enumerable or undecidable.

And if the characteristic function of a set is a recursive function, then the set is said to be recursive or decidable.

Because languages can be considered as infinite sets, we talk about recursively enumerable or undecidable (and recursive or decidable languages) if there is not (or there is) an algorithm to decide for an arbitrary sentence whether it is in the language or not.

In the theory of formal languages and abstract automata several systems have been defined to represent procedures: Türing machines, register machines, Chomsky phrase structure grammars, Post systems, Markov algorithms and of course programming languages. Normally we use a natural language description of a procedure and give formal definitions in a programming language, because then it can be demonstrated that the procedures are working by objective experimentation: we simply execute the procedure by some machine. It has been shown that all these systems are notational variants, i.e. the one has no more computational power than the other.

In addition the so called Church-Türing thesis is accepted which states that any explicit process that is completely understood, can be simulated on a Türing machine. In other words, if you are able to explicate the working principles of a process, then you will be able to express these principles into a procedure. The problem is of course to discover the working principles.

The theory of formal languages not only provides us with a way to write down procedures in an accurate way, in addition a hierarchy of notation systems has been discovered, which provides a finer distinction than that between algorithms and procedures.

The best known example is the so called Chomsky hierarchy already discussed earlier, and the hierarchy of abstract automata equivalent to it:

type 3: finite state machines
type 2: pushdown automata
type 1: linear bounded automata
type 0: Türing machines

An important result of studying sets in their relation to these systems (via a characteristic function) is that type 0 systems define recursively enumerable sets/languages whereas type 1-2-3 systems define recursive sets/languages.

We now introduce members of the class of automata on two levels of the hierarchy. The reason for picking out these systems and no other ones (e.g. Markov algorithms) is that they have a special place in the theory of natural language processes. So e.g. we will introduce recursive transition networks on level 2 instead of pushdown automata because the former are used substantially where the latter not at all. Much more detail about automata theory can be found in the references.

(1) LEVEL 3

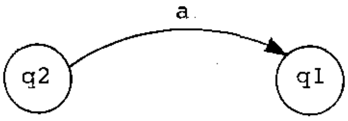
The least powerful automaton is a finite state automaton.

Definition

A finite state automaton FA is a 5-tuple $= \langle Q, \Sigma, \delta, q0, F \rangle$ with

1. Q a finite nonempty set of states
2. Σ a finite nonempty set of symbols
3. δ: $Q \times \Sigma \rightarrow \wp(Q)$ is the transition function
4. $q0 \in Q$ is the initial state
5. $F \subsetneq Q$ is the set of final states

δ is represented graphically as follows, if $q1 \in \delta(q2,a)$ with $q1,q2 \in Q$ and $a \in \Sigma^{*}$ then



A complete graphical representation for δ is called a transition diagram.

Definition

Let $\Theta = \langle Q, \Sigma, \delta, q0, F \rangle$ be a finite automaton then a configuration of Θ is a pair $\langle q, \sigma \rangle$ in $Q \times \Sigma^{*}$

The reduction relation denoted as $\vdash$ holds between two configurations α and β if $\alpha = \langle q, a\sigma \rangle$ and $\beta = \langle q', \sigma \rangle$ where $a \in \Sigma$, $\sigma \in \Sigma^{*}$, $q,q' \in Q$ and $q' \in \delta(q,a)$

Let $\overset{*}{\vdash}$ denote the reflexive and transitive closure of $\vdash$.

Definition

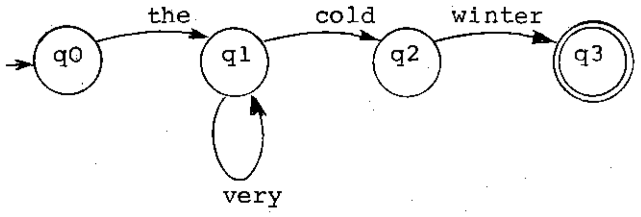
The language defined by Θ denoted as $L(\Theta) = \{\sigma \mid \sigma \in \Sigma^{*}$ and $\langle q0, \sigma \rangle \overset{*}{\vdash} \langle q, \lambda \rangle \}$ for some $q \in F$

Example

We construct an automaton for the sentences "the $(\text{very})^{n}$ cold winter", with $n \geqslant 0$.

Let $\Theta = \langle Q, \Sigma, \delta, q0, F \rangle$ and
$Q = \{q0,q1,q2,q3\}$, $F = \{q3\}$, $\Sigma = \{\text{the,very,cold,winter}\}$

the transition diagram:



Examples of operation:

(i) "the cold winter"

$\langle q0, \text{"the cold winter"}\rangle \vdash \langle q1, \text{"cold winter"}\rangle \vdash \langle q2, \text{"winter"}\rangle$

$\vdash \langle q3, \lambda\rangle$

(ii) "the very very cold winter"

$\langle q0, \text{"the very very cold winter"}\rangle \vdash \langle q1, \text{"very very cold winter"}\rangle$

$\vdash \langle q1, \text{"very cold winter"}\rangle \vdash \langle q1, \text{"cold winter"}\rangle$

$\vdash \langle q2, \text{"winter"}\rangle \vdash \langle q3, \lambda\rangle$

(2)LEVEL 2

A second class of systems is more powerful in the sense that they admit embedding. The example we will discuss on this level is a system called <u>recursive transition network automaton</u>. It came out of computational linguistics studies as a process model for context-free grammars and as an alternative for pushdown automata.

Although recursive transition nets are widely used (since ± 1970) they seem not yet to have reached the textbooks on formal language theory. We introduce here a formalism for recursive transition networks that is new by lack of standard notations.

Definition

A recursive transition network automaton is a 6-tuple
$R = \langle Q, V, \delta, Q_O, F, Fc \rangle$
with

1. Q a finite nonempty set of states
2. V a finite nonempty set of symbols $Q \cap V = \emptyset$
3. $\delta : Q \times (V \cup Q) \rightarrow \mathcal{P}(Q)$ is the transition function
4. Q_O is the initial state
5. $F \subseteq Q$ is the set of final states
6. $Fc \subseteq F$ is the set of completely final states.

δ is represented graphically just as for finite automata.

Note that the only differences between finite automata and recursive transition nets so far are that (i) we distinguish a subset in the set of final states
and (ii) a 'condition' for a transition cannot only be a symbol from the alphabet but a state as well. The motivation for including the latter will become clear in next definition.

Definition

Let $R = \langle Q, V, \delta, q_o, F, Fc \rangle$ be a recursive transition network, then a configuration β of R is a pair $\langle \gamma, \sigma \rangle$ in $Q^\star \times V^\star$

The reduction relation denoted as $\vdash$ holds between two configurations α and β if $\alpha = \langle q\gamma, a\sigma \rangle$ with $q \in Q$, $\gamma \in Q^\star$, $a \in V$, $\sigma \in V^\star$
and

(i) TRANSITION $\alpha \vdash_{\text{TR}} \beta$ with

$\beta = \langle q'\gamma, \sigma \rangle$ if $q' \in \delta(q,a)$

(ii) PUSH: $\alpha \vdash_{\text{PUSH}} \beta$ with

$\beta = \langle q'q''\gamma, a\sigma \rangle$ for $q', q'' \in Q$ and $q'' \in \delta(q,q')$

(iii) POPUP: $\alpha \vdash_{\overline{POPUP}} \beta$ with

$\beta = \langle \gamma \ , a\sigma \rangle$ if $q \in F$

$\vdash = \vdash_{\overline{TR}} \cup \vdash_{\overline{PUSH}} \cup \vdash_{\overline{POPUP}}$ and $\overset{*}{\vdash}$ is the reflexive and transitive closure of $\vdash$.

Comment: In a configuration $\langle \alpha \ \gamma \rangle$ γ is called a <u>pushdownstack</u> because information is placed on top of it and the latest added information is first consumed.

<u>Definition</u>

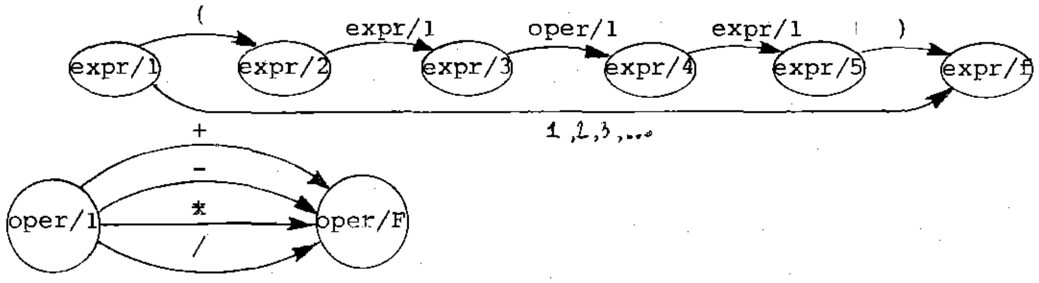
The <u>language</u> defined by R, denoted as $L(R) = \{\sigma \mid \sigma \in V^*$ and $\langle q0 \ , \ \sigma \rangle \overset{*}{\vdash} \langle q, \lambda \rangle$ (for some $q \in Fc) \}$

<u>Example</u>

Let us define a language for bracketed numerical expressions such as ((1 + 1) x (2 + 3)) or (((1 + 1) + 1) + 1)

R = ⟨Q,V, δ, expr/1 ,F,Fc ⟩ with
Q = {expr/1,expr/2,expr/3,expr/4,expr/5,expr/f,oper/1,oper/F}
V = {1,2,3,..., +,x,-,/,(,)} F = {expr/f,oper/F}
Fc = {expr/f}

The transition diagram:



Example of operation

Let = (((1 + 2) + 1) + 3)

⟨expr/1,"(((1 + 2) + 1) +3)"⟩ $\vdash_{\text{TR}}$ ⟨expr/2,"((1+2)+1)+3"⟩

$\vdash_{\text{PUSH}}$ ⟨expr/1 expr/3,"((1+2)+1)+3)"⟩ $\vdash_{\text{TR}}$ ⟨expr/2 expr/3,"(1+2)+1)+3)"⟩

$\vdash_{\text{PUSH}}$ ⟨expr/1 expr/3 expr/3,"(1+2)+1)+3)"⟩

$\vdash_{\text{TR}}$ ⟨expr/2 expr/3 expr/3,"1+2)+1)+3)"⟩

$\vdash_{\text{PUSH}}$ ⟨expr/1 expr/3 expr/3 expr/3 , " 1 + 2) + 1) + 3)"⟩

$\vdash_{\text{TR}}$ ⟨expr/f expr/3 expr/3 expr/3 , " + 2) + 1) + 3)"⟩

$\vdash_{\text{POPUP}}$ ⟨expr/3 expr/3 expr/3 " + 2) + 1) + 3) "⟩

$\vdash_{\text{PUSH}}$ ⟨oper/1 expr/4 expr/3 expr/3 " + 2) + 1) + 3) "⟩

$\vdash_{\text{TR}}$ ⟨oper/F expr/4 expr/3 expr/3 " 2) + 1) + 3) "⟩

$\vdash_{\text{POPUP}}$ ⟨expr/4 expr/3 expr/3 , " 2) + 1) + 3) "⟩

$\vdash_{\text{PUSH}}$ ⟨expr/1 expr/5 expr/3 expr/3 , " 2) + 1) + 3) "⟩

$\vdash_{\text{TR}}$ ⟨expr/f expr/5 expr/3 expr/3 , ") + 1) + 3) "⟩

$\vdash_{\text{POPUP}}$ ⟨expr/5 expr/3 expr/3 , ") + 1) + 3) "⟩

$\vdash_{\text{TR}}$ ⟨expr/f expr/3 expr/3, " + 1) + 3) "⟩

$\vdash_{\text{POPUP}}$ ⟨expr/3 expr/3 expr/3 , " + 1) + 3) "⟩

$\vdash_{\text{PUSH}}$ ⟨oper/1 expr/4 expr/3 , " + 1) + 3) "⟩

$\vdash_{TR}$ ⟨oper/F expr/4 expr/3 , " 1) + 3) "⟩

$\vdash_{POPUP}$ ⟨expr/4 expr/3 , " 1) + 3) " ⟩

$\vdash_{PUSH}$ ⟨expr/1 expr/5 expr/3 , " 1) + 3) " ⟩

$\vdash_{TR}$ ⟨expr/f expr/5 expr/3 , ") + 3) " ⟩

$\vdash_{POPUP}$ ⟨expr/5 expr/3 , ") + 3)"⟩ $\vdash_{TR}$ ⟨ expr/f expr/3, " + 3)" ⟩

$\vdash_{POPUP}$ ⟨expr/3, " + 3) "⟩ $\vdash_{PUSH}$ ⟨ oper/1 expr/4, " + 3) "⟩

$\vdash_{TR}$ ⟨oper/F expr/4, " 3) "⟩ $\vdash_{POPUP}$ ⟨ expr/4, " 3) " ⟩

$\vdash_{PUSH}$ ⟨expr/1 expr/5, " 3)" ⟩ $\vdash_{TR}$ ⟨ expr/f expr/5, ")" ⟩

$\vdash_{POPUP}$ ⟨expr/5, ")" ⟩ $\vdash_{TR}$ ⟨expr/f, λ ⟩

An extension of recursive transition networks up to the level of type 1 and even type 0 can be obtained by introducing (1) arbitrary conditions for the transition to take place, (ii) registers in which additional information can be stored and (iii) actions each time a transition is made to change the contents of these registers.

This type of systems is called augmented transition networks and they arewidely used for natural language processing because it is an interesting and powerful process model for transformational grammars. But for this very reason we will not be concerned here any further with this type of systems. It is shown elsewhere that by superposing on each other type 2 systems instead of integrating all the rules in one system, the linguistic facts for which you need the augmentation can be dealt with without increasing the power of the grammar.

Discussion

After some intuitive explanations about computation by machines, we presented some relevant aspects of computation theory. Important for our purpose here are the following points.

1. If we know explicitly how natural language processes work we will be able to design a procedure defining this process according to the Church-Türing hypothesis.

2. Some care is necessary however. We must find not only a characterization of the process in terms of a procedure but one in terms of an algorithm. The reason is that with a procedure it is not necessarily guaranteed that the process is finite. In other words, suppose we construct a program for understanding natural language, then if we give it a sentence, it would not be guaranteed that the program will ever stop ! Clearly we do not want that. It is counterintuitive in comparison to human language use and it is impracticle.
The theory of computation warns us for this situation, a warning which is not wholy unnecessary because transformational grammars for example are type 0 systems.

Further references

There are a great number of works on the general theory of computation and on the theory of abstract automata available. We mention especially Minsky(1967), Arbib (1969) and Engeler (1973) and the textbooks on formal languages already referenced when introducing formal grammars.

§ 0.3. Metatheoretical considerations

In this section we try to formulate an answer to the following questions: (i) what is the metatheoretical structure of the proposals to be presented in this work and (ii) what is the scientific status of the components in the structure.

First we state briefly some assumptions underlying the present discussion (1). Arguments for these assumptions can be found in the works cited in the references at the end of this section. Then we deal with the structure of the theory (2) and with the status of each aspect in the structure (3). In a final subsection we will discuss in some more detail the experimental method we are going to follow (4).

(1) ASSUMPTIONS

1. There are empirical sciences where the theories have a relation to some part of an observable reality, and non-empirical sciences which do not have such a relation.
Linguistics is an empirical science.

2. For a theory to be considered a scientific theory, it is necessary that the theory has at least the following properties:

(a) All aspects of it should be fully explicit (i.e. the theory should be exact). A good test for this is to construct computer programs based on the theory.

(b) The theory should be internally consistent. The construction of computer programs is an equally valid test for this purpose:

(c) It should be possible to falsify the theory, i.e. it must be clear what claims are being made by the theory and how each of these claims can be refuted.

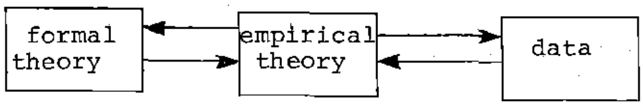
(d) A theory dealing with everything all at once is outside the scope of the present knowledge. It follows that the domain of the theory is somehow restricted. It should be possible to see the restrictions being made and it should be obvious in what directions further developments may extend the domain.

This list is not meant to be exhaustive. E.g. we did not go into the purely'semantic' properties of what it means to be a theory (see Achenstein,1968, for such a discussion).

In the following discussion aspects (c) and (d) will be of particular interest to us.

(2) Structure

We now start by presenting the 'normal' structure of a linguistic theory. The structure has three compartments:



(i) The formal theory

A formal linguistic theory, often called a universal grammar (although we are here only thinking about "formal universals") is the definition of a language in which descriptions of a natural language can be expressed. An example of such a formal theory is the transformational theory in which it is specified e.g. that linguistic structures take the form of trees, that rewrite rules are to be used as a means to define regularities, that a special sort of rewrite rules, namely transformation rules, are a necessary component, etc.

(ii) The empirical theory

The empirical theory is the actual description of the data in a form specified by the formal theory. We often say that the empirical theory is an interpretation of the formal theory.

An example of such an empirical theory would be a transformational grammar for a particular language.

(iii) The data

The third component of the structure consists of the data for which it all started.

Between each component there are certain well defined relationships and it must be possible to prove that the relationships hold, otherwise the whole construct collapses. In particular:

(i) It mustbe possible to decide whether an arbitrary empirical theory is a member of the class of possible empirical theories characterized by the formal theory and

(ii) it must be possible to decide whether an arbitrary sentence is a member of the class of sentences characterized by the empirical theory.

In an optimal situation, both decision processes should be algorithms, although (from a purely theoretical point of view) it is not such a great harm that there is no algorithm and that creative intellects are necessary to prove the relation between the various levels.

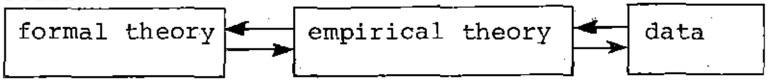
For the transformational theory the relation between the formal theory and the empirical theory is usually obvious, albeit that this relation is seldom explicitly proved. The relation between the empirical theory and the data is taken care of by the derivation relation inherent in generative grammars. By means of this derivation relation it is possible to prove exactly that a certain piece of data falls indeed under the empirical theory. On the other hand it is known that there is no algorithm that given an arbitrary piece of data tells us whether it is defined by a particular transformational grammar or not.

Having presented very briefly the commonly accepted structure of a linguistic theory, and an example of it, namely transformational grammar,we now turn to a metatheoretical investigation of the linguistic theory presented in this work. The most important results of this investigation are:

(i) The relation between the 'grammar' and the 'data' can no longer be proven directly but we have other means available, and

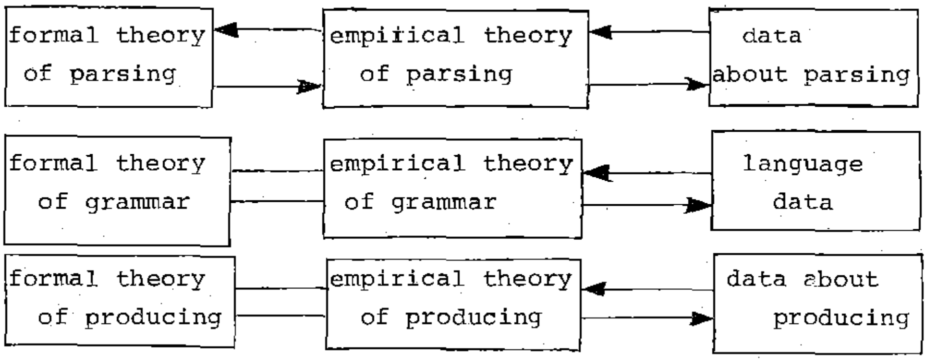
(ii) the formal completeness is no longer guaranteed.

The first thing of importance is that the straightforward structure



should be extended simply because the subject matter of the theory itself has been enlarged. We are dealing with a theory about parsing, a theory about production and a theory about the knowledge used in both: the grammar.

So as a first approach we get


formal theory of parsing
empirical theory of parsing
data about parsing
formal theory of grammar
empirical theory of grammar
language data
formal theory of producing
empirical theory of producing
data about producing

But this is not quite the scheme we want to have, simply because that is not the way we proceed. In particular the parsing/producing systems are in this work not studied as empirically observable entities, therefore the structure of an empirical science does not apply to these investigations. Note that there do exit parsing and production systems in reality: every human has one. Still we do not apply an empirical approach to them. This is a work of linguistics, the only empirical reality we are dealing with is language.

Instead we have the following structure.

We construct a formal theory with two components:

(a) A formal theory of grammar defining ways to represent linguistic knowledge, we call this the <u>description theory</u>

(b) A theory of parsing which defines the set of parsing processes for all sentences defined by a possible grammar defined by the description theory. That means the process that occurs if a sentence defined by a possible grammar is analysed into the structures defined in the description theory. And a theory of producing which defines the set of production processes for all sentences defined by a possible grammar defined by the description theory, that means the process that occurs if a structure defined in the description theory is converted into a natural language sentence based on a possible grammar.

We call the theory of parsing and producing the <u>process theory</u>.

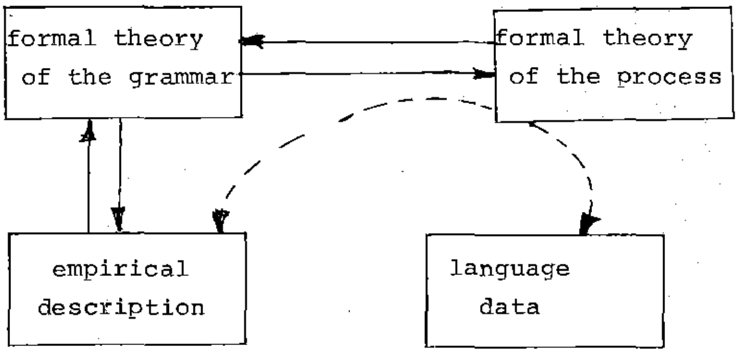
There is a clear relation between the process theory and the description theory in the sense that for each formal rule in the description theory there is a predicate in the process theory (as the reader will see). On the other hand the process theory involves more than the description theory because knowledge due to the process itself is available.

Now the next question is, what is the relation to the language data themselves. It must be obvious from the presentation that due to its formal properties it is not possible to 'generate' in some way language sentences on the basis of the grammar alone. The grammar tells us what factors are responsible for what language phenomena but not what the interrelationships of the phenomena itself are, this is so because the grammar is organized in a modular fashion. This brings us however in the unhappy position that the relation between the language data and the description system cannot be proven. Fortunately we have a theory of parsing and producing and by means of this theory we can indeed provide the necessary relation.

How is this going. Consider a language sentence and a particular grammar which specifies all information for this sentence in a format prescribed by the description theory. How can we know that the grammar specifies indeed the information for that sentence ? For this purpose we introduce the parser which is actually a function taking as arguments (i) the sentence itself, and (ii) the grammar under discussion, and produces as result the structure assigned by the grammar to the sentence. If no structure is produced the language defined by the grammar does not include the sentence.

To complete the proof, we take the structure computed by the parser and hand it over to the producing system. This producing system is again a function taking as arguments (i) the structure and (ii) the grammar under discussion, and produces the natural language sentence again.

We can summarize the results of the discussion in the following diagram depicting the structure of the theories presented here:



We come now to our second topic: the status of the components in the structure. We first investigate the 'normal' type of structure.

(2) Status

Let us investigate the question of falsification and incompleteness.

(i) Falsification

A formal linguistic theory of the 'normal' type is falsified if certain description modes which are defined in the formal theory are superfluous for the formulation of empirical theories.

An empirical linguistic theory is falsified if it defines language phenomena which do not occur.
It suffices to find such a phenomenon and the empirical theory is falsified.

(ii) Incompleteness

A formal linguistic theory is incomplete iff certain systematic aspects of a natural language cannot be expressed.

An empirical theory is incomplete iff certain phenomena which occur in the data are missing in the description.

It is of interest to note that

(i) A transformational theory is always complete because the transformational grammars are type O systems and therefore any computation process can be defined in terms of transformational grammars (according to the Church-Türing thesis)
(ii) The price to be paid for completeness is however that the formal transformational theory is much too broad. In this perspective the attempts to restrict the power of transformational grammars become extremely important.
(iii) We think it is fair to state that there is at the moment no complete empirical theory for any language based on a transformational model (and there is no such a theory for any other model).

(iv) On the other hand if the empirical theory defines phenomena which do not occur, this is mostly due to carelessness of the grammar writer. Although it must be said that transformational grammar is not an easy or perspicuous way of representing systematic aspects.

We study now the conditions for falsification and incompleteness for the theory under discussion in this work.

(i) Falsification

The description theory is falsified if certain description modes which are defined in the formal theory are superfluous for the formulation of empirical theories.

The empirical description is falsified if it defines language phenomena which do not occur.

The process theory is falsified if the rules expressed in the description theory do not lead to the predicted results.

(ii) Completeness

The description theory is incomplete if there are phenomena occurring in natural languages which cannot be expressed by the formalism provided by the theory.

The empirical theory is incomplete if there are phenomena which occur in the data but are missing in the description.

The process theory is incomplete if there are rules in the description theory for which the corresponding processes have not been defined.

Notes

(a) It is known that the formal description theory to be presented here is incomplete (a proof of this incompleteness is provided in the text). Also it is known that no description example to be given is complete, far from it.

(b) It will become obvious that it is trivial to prove that the process theory is complete as regards the formal description theory.

(c) As regards falsification, we claim that the formal description theory so far will be hard to falsify. It suffices to show that a certain aspect is superfluous but we do not think that this is the case. It is rather too weak than too powerful.

In this context we want to make the following remark. It is highly probable that is is impossible to construct a complete empirical interpretation of a language because of the complexity involved. What we need therefore is a learning system that is able to extend its knowledge on the basis of its own observations. Although some work is going on in this area (cf. Sikklossy (1972),Anderson (1975)) it is generally accepted that we are not yet far enough to construct such systems. Meanwhile our own attempts to write empirical interpretations constitute a sort of learning process. Each time we add new words or types of constructions we extend or change the description of the language.

(d) We consider the process theory strongly confirmed by the computer programs we constructed for it and the experiments being done: One could say that the theory of parsing and producing defines a way of experimentation by which an empirical description theory can be tested. As we were able to construct computer programs for simulating the parsing and production processes the experiments can be performed fully automatically and in a completely objective way. Notice that the highest standards of experimentation as regards exactness, repeatibility, etc. are all met with. We think that the ability to perform experiments is a very important aspect of our work. In next paragraphs we provide some more detail about the way in which they are performed.

(4) The experimental method.

As we said the test whether a language sentence is properly treated by the linguistic theory (in all its aspects) can be performed by means of experiments. The performance of experiments is something unusual in linguistic theories. We therefore study the conditions under which we do these experiments in some detail now. At the same time this will enable us to reflect on the nature of the linguistic argumentation being used.

The need for experiments

Whenever human beings deal with complex problems they try to develop means to control this complexity. In science this is done by introducing machines that gather data automatically or which perform the calculations involved in complex computations, etc.
It need not be said that natural language is an example of an extremely complex problem . The software needed to process a spoken natural language sentence is comparable to that needed to send a manned rocket to the moon !
Due to this complexity it is simply necessary to use machines which assist us in testing the theories.

The preparation

In order to execute experiments, we use general purpose machines, i.e. computers, although it is perfectly possible to construct linguistic systems directly in hardware. The main preparation necessary is the construction of computer programs that reflect in full detail the proposals made by the formal linguistic theories.

At least the exactness and consistency requirement should be met with if these programs are to be successful. The construction of the programs requires some technical knowledge from the part of the experimenter, but every experimentation involves a technological background and there is no reason why it should not be part of the basic training of the linguist.

Once the program is ready, the empirical theory to be tested is prepared for consultation by the program. Finally we give an input sentence and the result comes out. It need not be said that the preparation of an experiment needs the utmost care up to the finest detail.

The evaluation

Now comes the important part of the discussion here;the evaluation of the outcome of an experiment. If the outcome is as was expected (i.e. predicted) all components of the metatheoretical structure are confirmed. But if there is not outcome or not the outcome we wanted to have, the following method of reasoning comes into action:

(a) The preparation.

First we critically examine the way in which the experiment was performed: Whether no errors occured in the construction of the programs, whether the data were entered in the format of the programs, etc.

The remedy to improve the preparation is simply to improve the program or to improve its data.

(b) Empirical theory

If the performance of the experiment itself is allright, we examine the empirical description. Maybe wrong facts were included or inappropriate facts. One can (and we did) design the experiments in such a way that it becomes obvious in what way the empirical theory is false or incomplete. The remedy here is to extend the description or change it.

(c) The formal description theory

Suppose however that we try to deal with a certain fact and we cannot express it in the format that is provided, then it becomes necessary to extend the description theory itself. This is normally a far reaching activity. Not only will it be necessary to extend the process theory, but moreover the experimental setting itself will need a revision. (This is not necessary if the empirical theory fails)

(d) The process theory

The process theory as such is constructed in direct relation to the description theory and will therefore be reworked as soon as the description theory is reworked. Besides these considerations it may be that the bad outcome of an experiment is due to a badly conceived process theory .

In such a case we work on the process theory and subsequently change the experimental setup itself.

Discussion and further references

There is a growing literature about the metatheoretical foundations of linguistics, especially in the German language (see e.g. Wunderlich,1974, Van de Velde,1975). The reader is referred to these texts for a characterization of linguistics as an empirical science and for the deductive structure of generative theories.

For the problem of falsification as a method of investigating the scientific status of a theory, see Popper (1974). The problem of incompleteness is unfortunately not very much on the foreground in the philosophy of science.

The use of computer simulation as a method for proving the operational feasibility of a linguistic theory is not yet very widespread in linguistics. Although a very fine example exists for transformational grammars (Friedman,1968). Normally work with computers is placed in an 'applied linguistics' corner, but we think this is an underestimation of the power obtained by having machines to assist you in the testing and development of linguistic theories. We feel that for the heuristics involved in the present investigation, the use of computers proved to be irreplacable.
See for the metatheoretical foundations of computer simulation in general Harbordt (1972).

§ 1. THE THEORY OF MODULAR GRAMMAR

In this chapter we introduce a grammar theory which was designed with the parsing and production problem in mind. This grammar theory is a linguistic theory in the usual sense: A formal model for the representation of the systematics in language.

At the same time we will provide some examples of an empirical interpretation of this formal model for some natural languages. These examples are incorporated to illustrate the approach, they are by no means meant to constitute a complete description of the natural language being discussed.

To make clear the distinction between the formal model and the empirical interpretation of it, the reader can keep in mind that every statement with the label definition is part of the theory and every statement with the label example is part of the empirical use of the theory. All the rest are intuitive explanations.

After each subpart of the text we insert discussions and further references which bring our ideas in the perspective of existing linguistic theories. On the whole the reader will find this perspective more in accordance with the European traditions of language study than with recent American approaches, except maybe for the exactness in formalism we are striving for. These discussions can be skipped at first reading.

§ 1. THE THEORY OF MODULAR GRAMMAR

1.0. INTRODUCTION TO MODULAR GRAMMAR

The model we are introducing here will be called a modular grammar because the major deviation from other theoretical approaches is that instead of striving for an integration of all linguistic knowledge into one compact single system, we decompose the grammar in several independent modules.

In a way you could say that any theory of language is 'modular' in the sense that various components (morphology syntax, semantics) are distinguished and in each component still further subcomponents (e.g. in transformational grammar you could say that the lexicalisation transformations, the various cycles, the postcyclic transformations are each different modules of the subcomponents, you could even say that each transformation is in fact a module on its own !). But that is not the way in which we want to use the term module.

In Webster's dictionary we find that module means

(a) 'any of a set of units (...) designed to be arranged or joined in a variety of ways;

(b) a detachable section, compartment, or unit with a specific purpose or function, as in a space craft;

(c) in electronics; a compact assembly functioning as a component of a larger unit.'

We here envisage especially meanings (a) and (b), for (a) in particular that it is possible to arrange or join modules in a variety of ways (b) that each module has a specific purpose or function. When we say modular we mean that the various rules of the grammar are seen as independently consultable sources of knowledge which can be joined in a parallel fashion with other modules to accomplish the task of producing or understanding natural language.

Now how do we get all those modules ? From observation it is clear that natural languages use a number of devices such as the ordering from left to right of the words, the use of concord or agreement, the use of morphological affixes to signal certain relationships,etc. In an integrationistic grammar all these phenomena are stated in the same type of rules, e.g. rewrite rules, and each rule operates on the result of the application of other rules. In other words the grammar rules specify explicitly the interaction necessary to obtain the whole language sentence.

It turns out that there are some good reasons why it is advisable not to proceed in this way. These reasons stem from purely theoretical considerations (e.g. efficiency of representation) and especially from the problem of designing a whole language system.

The alternative to an integrationistic conception of grammar which we will present here, is to see the grammar as a set of specialists: One specialist is competent in word order, another one is competent in agreement rules, and he knows exactly in what situations they are applicable and how it should be done, another one is competent in morphological affixes for the signalling of case relations, etc;.
The specialists on their own cannot cause the analysis or production of a natural language sentence, to that purpose processes outside the scope of the specialists are neeessary.

Let us call a specialist a MODULE. It is a body of knowledge concerned with a specific aspect of language. As the grammar consists of a set of modules, it follows that a linguistic theory should investigate the knowledge contained in the modules. This investigation has three main aspects: first what kind of knowledge is involved, second how should we represent this knowledge and third how should we use the knowledge.

The third problem will be treated in next chapter when we come to a discussion of the whole language system in operation. In this chapter we will further concentrate on the first two aspects.

As regards the problem of how the knowledge should be represented, we point out that on some occasions this representation will be very straight forward, on other ones we will have to introduce quite complicated representation constructs to realize our goals. In particular we anounce the introduction of a new class of automata and a new representation construct for feature complexes.

As regards the problem of what kind of knowledge is involved we may already point out that there are two main things that will be discussed in this context: First there is a situation in the language sentence that is of interest, second (and even more important) there is a reason for the situation to be there. Let us call the situation a language phenomenon and its reason a factor. The factors themselves are anchored in the higher level process of semantic structuring as we will see. Examples of situations are word x comes before word y, word x takes certain features of word y, etc. Examples of factors are word x has a particular grammatical function as regards word y, word x is stressed, word x fills a certain case slot in the frame of word y, etc.

In each module one phenomenon and one factor are brought together. During analysis the module will be asked what factor is responsible for a particular phenomenon, during synthesis the module will be asked what phenomenon should be used to signal a certain factor. The bare information i.e. the relation factor/phenomenon is stated in a rule which forms the core of each module.

So we arrive at the following notion of grammar:

Definition

A modular grammar is a set of modules where each module contains a rule.
A rule is a function (in the mathematical sense). The function defines a relation between a language phenomenon and the factor(s) determining it. This implies that the formal structure of a rule r is

$$r(f) = p$$

with f the factor and p the phenomenon.

In the following sections we will make a start with investigating what kind of modules are necessary to represent the linguistic knowledge used by natural languages.

In particular we will investigate two important factors: grammatical function and case. We know that there are (probably many) other factors such as the type of speech act, the use of coordination, various sorts of 'pragmatic' factors (e.g. stress on particular aspects of the utterance), but one must start somewhere and it is impossible to cover everything all at once in a short amount of time. Moreover grammatical function and case appear to be very basic factors in the functioning of language and we think it is therefore simply necessary to start with them.

The rest of the text contains two main parts. In the first one we introduce the notion of grammatical function and the modules centered around grammatical function. In the second one we introduce the notion of case and the modules using case (and grammatical function). After that we will discuss some other topics, such as the relation to semantic structuring and some further problem areas.

Discussion and further references

In this first part we presented the first central assumption of our theory, namely that a linguistic description system should be organized in a modular fashion, rather than in an integrated one. This first central assumption is at the same time the first distinctive assumption. If we look at the grammar constructs being used at the moment, we see that they are all organized in an integrated way. Indeed, one could say that the idea to have such an integrated description system has been growing gradually from the early traditions of structuralism to culminate in the conception of a transformational grammar (Van de Velde, personal communication). On the whole the more traditionalistic a grammar the more it is modularly organized ! (E.g. Zandvoort (1945) treats word order, concord, functional interrelationships, etc. in different chapters of this grammar; another example of a grammar with a modular flavour is Jespersen (1961))

Although the idea to have this modular organization of a grammar is in direct opposition to the current trendin linguistics, in other areas relevant to the subject of natural language, modularity has already been recognized as being a very fruitful approach towards the organization of knowledge. We are here thinking about studies in artificial intelligence. Here modules are called demons (Charniak,1972), specialists or molecules (Rieger, 1975). Each time referring to a body of knowledge needed to perform a certain cognitive task (e.g. inference making). The necessity of having modular whole systems has become especially obvious when trying to design speech understanding systems which must be able to cope with unclear data (see Reddy(1973) for a discussion of the problem and Bruce and Nash-Webber(1976) for an example of a speech understanding system).

Although the idea of modularity is obviously present in artificial intelligence, it has never been applied to the design of grammars itself. An augmented transition network e.g. (cf. Woods, 1972) is a typical integrated system.

1.1 GRAMMATICAL FUNCTION

In the introduction to this section, we mentioned that we will be investigating two factors: grammatical function and case. In this subsection we present some modules concerned with grammatical function. First we introduce the concept itself in some detail.

1.1.0. Introduction to grammatical function

Definition

Let us consider a finite nonempty set of words W of a language, then the functional relations over W denoted as FR is a relation in the set theoretic sense $FR \subseteq W \times W$

If $\langle w1,w2 \rangle \in FR$, then we say that a grammatical relation holds from w1 to w2

We can furthermore distinguish subsets in FR where each subset defines a particular sort of grammatical relation.

If a particular grammatical relation, say $F \subseteq FR$,holds from w1 to w2 then we say that w1 has the grammatical function F as regards w2; w2 is called the head and w1 the subordinate of the relationpair $\langle w1,w2 \rangle$

If $\langle w1,w2 \rangle \notin FR$ then we say that w1 has the grammatical function NIL as regards w2, i.e. NIL is the empty grammatical function

If a word w occurs as the subordinate of at least one $F \subseteq FR$ then we say that F is a possible grammatical function of W.

Example

Let adjunct be a grammatical function then in "young boy" a grammatical relation holds from "young" to "boy". We say that "young" is the subordinate and "boy" the head, and that "young" has the function adjunct as regards "boy". Adjunct is a possible grammatical function of "young".

Additional conventions

1. It is well known that one single form of a word may have different functions and meanings. This is a serious problem in the design of natural language processing systems and we will see what we can do about it. Theoretically we will consider such a word form as being more than one word form: for each function or meaning then we could say that we are dealing with another word. This greatly simplifies our definitions.

2. Although the relational character is lost, we will often say that a word w1 is an F if there is a word w2 and w1 has the grammatical function F as regards w2. This is in accordance with existing habits.

We now bring the notion of grammatical function in relation to a sequence of words.

Definition

Let $w_1 \ldots w_n$ be a sequence of words, then the functional structure for $w_1 \ldots w_n$ is defined as follows:

- if $n = 1$ the functional structure of the sequence is the possible grammatical function of the only word occurring in that sequence;
- if n is greater than 1 the functional structure is the set of all pairs $\langle w_k, w_{k+1} \rangle$ such that

(i) a grammatical relation holds from w_k to w_{k+1};

(ii) except for one w_j each w_i $1 \leqslant i \leqslant n$ is the subordinate of at least one but no more than one relation pair where the head of this relation pair is w_k $1 \leqslant k \leqslant n$ and $i \neq k$. In other words each word in the sequence has at least one but no more than one grammatical function as regards another word of the sequence;

(iii)a path in a functional structure is a sequence of relation pairs where the head of one relation pair is the subordinate in the next relation pair in the sequence. A path is a circuit if the same relation pair occurs more than once in at least one path in the functional structure. There should never be circuits in a functional structure.

The word w_j which is not occurring as the subordinate of any relation pair is called the top of the functional structure. The top has of course a possible grammatical function.

Example

For "the edited translation of a text", the functional structure contains the following relations:

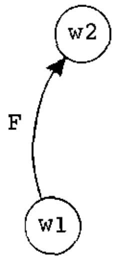
"the" has the function determiner as regards "translation"
"edited" has the function adjunct as regards "translation"
"translation" has the possible function object
"of" has the function casesign as regards "text"
"a" has the function determiner as regards "text"
"translation" is the top of the structure.

We are now faced with the task of defining a graphical representation construct for functional structures. The main requirement of this representation is that it should reflect the functional relations for a sentence in an explicit and perspicuous way.

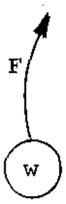
The solution we will adopt here goes as follows:
Use the standard mathematical way of drawing graphs for relations. The graph thus obtained is the representation we are looking for. More explicitly:

Convention:

If w1 has the grammatical function F as regards w2, then we draw a node for w1 and w2 labeled with w1 and w2 respectively. Then we draw a directed line from w1 to w2 and label the line with F:

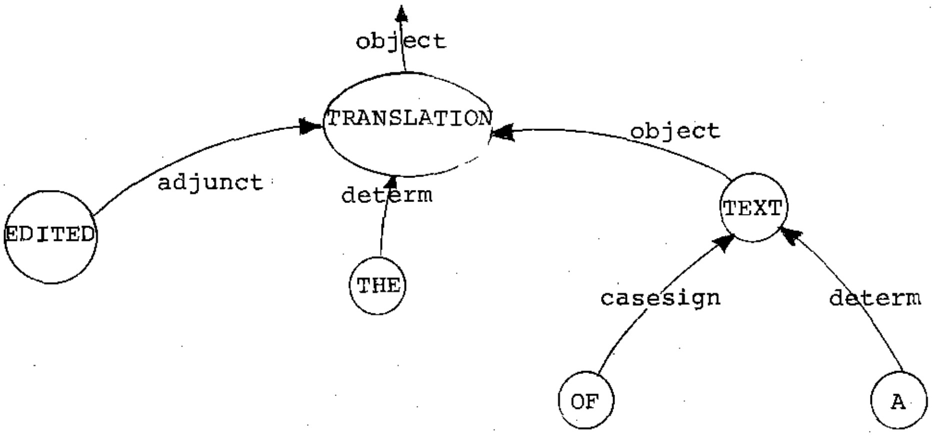


But if w has only a possible grammatical function F we draw a node labeld w and draw a line from it with label F



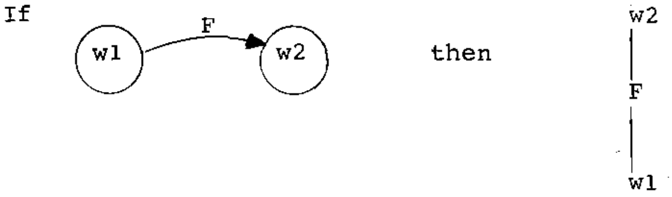
Example

For "the edited translation of a text":



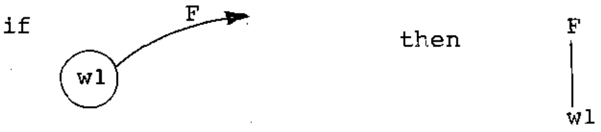
To simplify the representation we can turn those graphs into trees by the following convention:

If



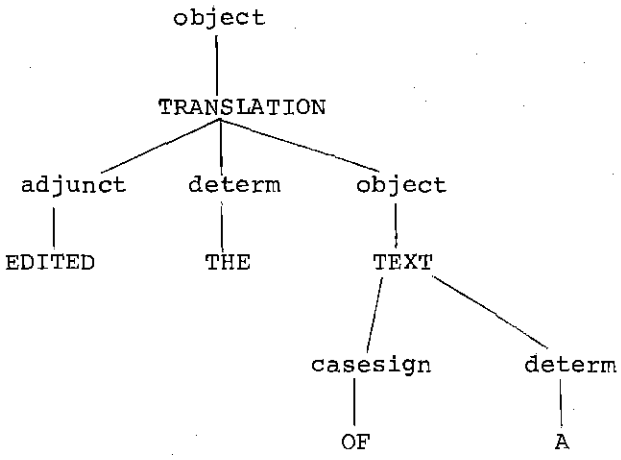
and

if



Example

For "the edited translation of a text":



It is important to keep the unsimplified graph representation in mind when studying functional structures.

Two questions can be asked in connection to these functional structures:

(1) Will the convention of turning graphs into trees always work ? (The question raises because a graph is a more powerful representation construct than a tree.)
Then answer is yes. The proof follows from the definition of functional structures.
A tree has the following properties (i) there is one topnode, (ii) this topnode is reachable form all other nodes, and (iii) there are no circuits.
Condition (i) is always satisfied because there is one w_j which is not the subordinate of any relation pair.
Condition (ii) is always satisfied because each word is connected to the graph via another word. Condition (iii)was a condition of functional structures per definitionem.

(2) Is it possible to construct a generative grammar which derives a functional structure just as a phrase structure grammar grammar derives constituent structure trees ?
(The question is important because it influences our choice of grammar type)
The answer is no. The proof follows from the method of constructing trees on the basis of the derivation relation as defined in formal language theory for phrase structure grammars. A consequence of this definition is that a node can only occur as dominating another one if its label occurs on the left of a rule. But this implies that the label is a nonterminal. Because the words of the sentence which are terminal symbols occur higher up in the tree, they should be nonterminals. But in a generative grammar the set of terminals and nonterminals form disjoint sets hence it is not possible to do it.

(Notice however that it is possible indirectly by means of the indirect tree construction method defined earlier for generative grammars)

We now have a definition of the concept of grammatical function and a definition of functional structures to represent the grammatical relations holding in a certain sequence of words. We close this introduction to grammatical functions by discussing a typology for functions and by introducing the concept of an inference tree.

Typology

One of the main results of our investigations is that it is possible to distinguish between 3 classes of functions and to translate this distinction into the formal theory itself. The question is first on what ground such as typology should be built.

As we said in the introduction to this chapter knowledge about a specific aspect of language as contained in a rule involves two things: a factor and a phenomenon. The factor here under discussion is grammatical function. Recall that a factor has relevance for the process of semantic interpretation. It follows that a typology of functions can be based on the differences as regards semantic functioning. But due to the second aspect in a rule, the way in which the language phenomena are approached is an equally valid approach. It turns out that the typology we will be proposing is based on both grounds. First the semantic side.

We will see later in more detail that the functional structure of the input sentence is some sort of control structure for the creation of semantic representations: with each function a particular tree building action is associated and what the arguments of this action are is determined by the functional relations in the sentence.

The fundamental entities of a semantic structure are the predicates (which may be considered as bundles of properties or relations). Each of the predicates has a certain role in the communication, some introduce entities, others modify other predicates, qualify an already introduced entity, etc. Now let us associate with each of these roles a certain grammatical function.

Seen from this perspective it turns out that there are two main types of functions: objects (leading to predicates which introduce entities) and adjuncts (leading to predicates modifying other predicates qualifying another entity).
A third class comes in for words which carry no predicates themselves but act as additional instruments to signal certain aspects. These are the functionwords.

So we obtain three basic classes:

Definition

Let F be the set of all grammatical functions, then F-obj, F-adju,F-functw is the set of grammatical functions of the type object, adjunct and functionword respectively.

Comments:

(i) Objects:

Objects are words which denote an entity or a class of entities, that means they will lead to a semantic structure which represents an entity or a class of entities. An object stands in a dependency relation to either other objects (as in the father of John) or adjuncts (as in translated from a text).
Traditional grammars further distinguish subject, direct object, indirect object, prepositional object and other sorts of objects.
We will not make that distinction because the particular relation of one object to its 'head' is better explicated in terms of case relations as we will show later.

(ii) Adjuncts

Adjuncts are words which 'amplify' or 'modify' an object or another adjunct, that means they will lead to a semantic structure attached to an object or another adjunct in which new information is introduced. This happens e.g. by relating the object which is modified to another object. Traditional grammars distinguish several types of adjuncts: predicators (or verbs), attributive adjuncts, predicative adjuncts, adverbial adjuncts, etc.

(iii) Functionwords

Functionwords are words not introducing any semantic predicates in the sentence, they only add features and modifications to the words which act as heads of the function words.
Examples are determiners, casesigns, particles, a.o..

The typology discussed above on semantic grounds will find a further foundation in the differences in behaviour which exist on a mere surface level, especially between objects on the one hand and adjuncts/functionwords on the other. In particular we will center the information on surface phenomena for objects with the head of the relation pair and for adjuncts and functionwords with the subordinate of the relationpair. It will become obvious very soon what we mean by this and why we do it.

Functional inference tree

Of equal importance in the whole theory is the idea that you may organise the set of functions into groups which show a particular behaviour as regards a certain phenomenon We do this to capture certain regularities in the rules of the grammar which are otherwise treated by using nonterminals (which we will not use at all). It is e.g. necessary to make within the general class of objects a distinction between pronominal objects.and nominal objects, simply because words having these functions may vary considerably in their behaviour. But nevertheless we must keep the possibility to consider the class as a whole.

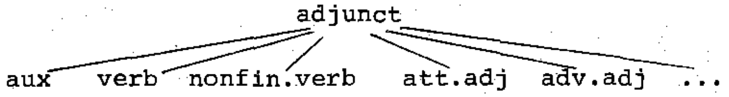
We solve this representational problem as follows: We define a hierarchy of functions which is represented in a tree. The tree will be called the functional inference tree (later on we will have inference trees for other theoretical objects). We will use our standard method of representing trees in list notation.

The idea is that given a (possibly sub)tree

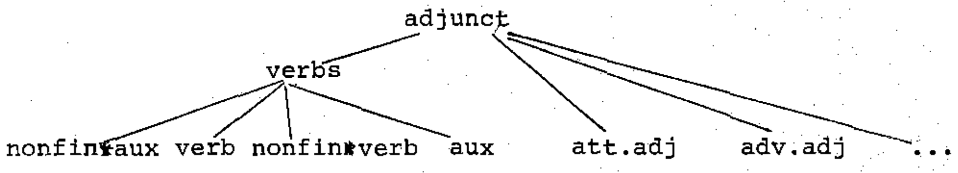
$(A \quad a_1 \ldots a_n)$ with $a_1, \ldots , a_n$ subtrees or functions

a property which is defined for node A also holds for every node in $a_1,\ldots,a_n$.

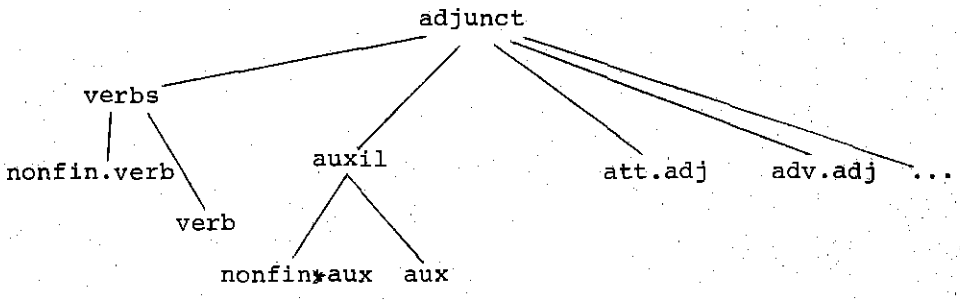
E.g. an adverbial adjunct may link with practically every other adjunct, so we define a tree for all adjuncts:



For another purpose it may be necessary to address only the verbs, then we make a subtree:



It may be necessary to still make a further subdivision, e.g. in order to address only the auxiliaries:



etc;

Discussion and further references

Already in traditional grammars the words of the language where classified according to their part of speech or syntactic category: noun, pronoun, adjective, verb, adverb, preposition, conjunction, interjection, etc.
In traditional gramars this classification was meant as an indication of the semantic function these words had in the communication, their 'mode of signifying' (Lyons, 1968,272) E.g. nouns are words naming entities, adjectives are words qualifying a noun by amplifying its meaning, etc.

In structural grammars the part of speech specification was more considered to be an indication of what structural properties the word having that part of speech could have. E.g. nouns are words showing a particular sort of behaviour on the morphological level, they occur only in certain combinations with other parts of speech, etc. Various methods were designed to classify the words via (structural) tests into distinct classes where each class was labeled with a particular category (or subcategory).

The two roles which are assigned to parts of speech by traditional grammarians and structural linguists respectively will in our grammar be related to grammatical functions (or functional categories) as they were called in traditional grammars).

The reason for doing so are as follows:
(i) To indicate the function of a word in the communication more precise characterizations are needed than the eight or ten parts of speech that were used in traditional grammars. This is so because (a) one single function (e.g. complement) can be realized by more than one part of speech (for complement: noun, pronoun, adjective, verb, adverb, etc.)and (b) one single part of speech (e.g. adjective) can have many different functions (for adjective:attributive adjunct, predicative adjunct, complement, etc;).

(ii) On the other hand many of the structural properties of a word are not determined by the part of speech it belongs to but by the grammatical function. We will see many examples of this in the sequel.

The step to make grammatical functions instead of categories or consituents the basis of the grammar is a very important one. And although phrase structure trees (and grammars) are a very powerful mechanism for dealing with a parts-of-speech analysis, they fail completely as regards the treatment of grammatical functions.

Because the grammatical functions are in a transformational grammar not represented explicitly in the deep structure, all surface phenomena which we will show to depend on grammatical functions and which are to be realized by transformations in such a grammar cannot in a clear way be related to these functions. Especially if the surface phenomena relate to so called 'derived' grammatical functions, such as attributive adjuncts which are obtained only after the application of a whole series of transformations. If the grammatical functions are represented by relations between dominance relations (as is normally assumed) the transformations will need extensive tree processing as a condition for their application.

The move towards deeper structures by the generative semanticists has nothing changed that would affect the criticism presented here. On the contrary, the fact that semantic structures are further away from the functional level results in even more obscurity as regards the role of grammatical functions and their effect on the surface structure.

The typology introduced above is strongly related to traditional ideas. E.g. the distinction of a special class of words not functioning in the semantic structure as predicate (the functionwords) is close to that of introducing a class of words having only grammatical significance or structural meaning vs. words which have not only a grammatical but also a lexical effect (Lyons,1968,435).

The idea of using a functional inference tree proved to be very powerful. Notice that in an integrationistic grammar these generalizations are to be expressed in the same sort of rules as those where the linguistic phenomena themselves are regulated. In contrast we declare the grouping of functions as a global phenomenon of the grammar.

Now we start introducing the rules themselves. This is done in a series of subsections each of which contains three parts (i) a theoretical introduction of the rule , (ii) an empirical example and (iii) discussions and further references.
Much more examples will be presented in the section on experimental results (Chapter 3).

1.1.1. The relations environment

The first phenomenon we will discuss is this: The occurrence of a functional relation presupposes the occurrence of other functional relations.

This takes two forms: Given a functional relation F between words w1 and w2, i.e. w1 is the subordinate and w2 the head, then

(i) the occurrence of the relation F presupposes that w2 has a particular function F', in other words a certain head is required;

(ii) the occurrence of the functional relation F presupposes that w1 is the head of another functional relation F', in other words a certain subordinate is required.

Let us discuss each of these aspects in some detail

1.1.1.1. Determination of the head

(i) theory

The first structural property of importance is given a word w1 and a word w2, for w1 to have a particular grammatical function F as regards w2, w2 should have a particular possible function F'.

For example take "the translated play", "play" can function as an object (drama for the stage)or as,amongst other things, predicator or verb. A possible function of "translated" is attributive adjunct, but obviously for "translated" to be attributive adjunct as regards "play", "play" should itself function as an object.

We express this by saying that a property of a word having the function attributive adjunct is that its head is always a word having the function object or shorter the function of the head of an attributive adjunct is an object.

Here is another example: "he translates plays", "plays" is an object of "translates" but this is so only because the head of "plays", i.e. "translates" takes objects. We express this by saying that a property of the word having the function verb is that it may take objects.

Notice our difference in talking about the two examples. In the first case (and in general for adjuncts and functionwords) we introduce the specification of the head as a property of the subordinate (i.e. attributive adjunct) and in the second case (in general for objects) we introduce the specification as a property of the head !

This is at first sight a remarkable attitude, but it will crop up again and again: information about functionwords and adjuncts is to be centered around the subordinate, information about objects around the head.

Having specified what kind of information we have in mind, we proceed by formulating the rule in which this information is presented. This turns out to be easy. We introduce two linguistic functions: function-of-head (for adjuncts and functionwords) which relates a function (the subordinate is having) to a function (the head is supposed to have) and taking-objects (for all functions) which relates a truth-value to a function to signal whether it takes objects or not.

Definition

function-of-head: $F \rightarrow F$ is a partial function defined $(\forall f)$ $(f \in F\text{-adju} \cup F\text{-functw})$ such that for w1, w2 words of the language, if w1 has the function f as regards w2, w2 should have the grammatical function f' = function-of-head(f)

f' may be a feature complex of functions (we will later explain what a feature complex is).

Definition

taking-objects: $F \rightarrow \{\text{TRUE, FALSE}\}$ is a function defined as follows:
let $f \in F$, then

$$\text{taking-objects } (f) = \begin{cases} \text{TRUE} & \text{if a word having the function } f \text{ may be the head of a relation pair with the function object.} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

(ii) example

Let us take the sentence

" a very urgent letter was sent to John"

and specify the function-of-head/taking-objects information.

We introduce the following grammatical functions:

determiner (for "a")
adverbial adjuncts (for "very")
attributive adjunct (for "urgent")
nominal object (for "letter" and "John")
finite auxiliary (for "was")
nonfinite verb (for "sent")
casesign (for "to")

Next we specify the information:

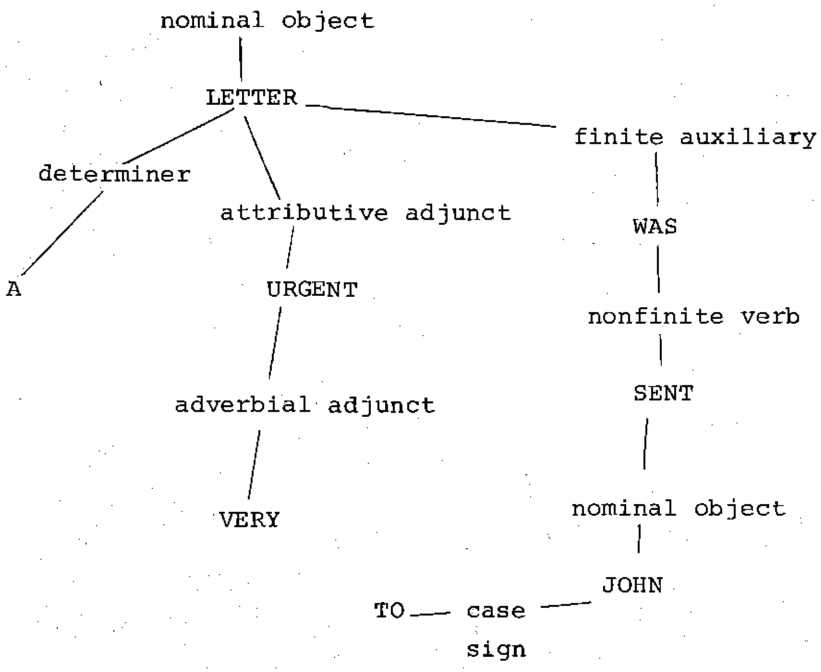
<u>function-of-head</u> (determiner) = nom.object
<u>function-of-head</u> (adverbial adjunct) = attributive adjunct
<u>function-of-head</u> (finite auxiliary) = nominal object
<u>function-of-head</u> (casesign) = nominal object

and

<u>taking-objects</u> (nonfinite verb) = true

for all other functions (in this sentence at least) taking-objects is false.

The following structure holds then for the sentence as a whole:



(iii) discussion and further references

The fact that other functional relations play an important role in the determination of the grammatical relation of one word has since long been recognized (think e.g. about the structuralistic notion of syntactic valence). Notice however that normally these functional restrictions are expressed in categorial terms, and in particular by means of the notion of phrase structure, constituent structure, or related concepts . In such a categorial context, the knowledge captured by the function-of-head and taking-objects rules, is formalized by placing the element in a whole pattern (such as in phrase structure grammars) or by a more explicit indication (such as in categorial grammars).

Our approach differs in two ways from the currently accepted one. First of all we express this information in terms of functions. The reason is that the same category (e.g. adjective) may function differently (att.adj, complement,etc.) in different environments. Second we do not include any information about order in the given rules. This is in accordance with our principle of a modular grammar. Notice that this may already lead to a more economical grammar: if the same function occurs in different orders, then this rule needs to be specified only once, in an integrated grammar we would need to specify the relational environment for every order anew. Another element of economy is that we do not need nonterminals. This reduces the number of theoretical terms being used.

Finally notice that in an integrationistic grammar it is impossible to formalize the difference in behaviour between adjuncts/function-words and objects. Although this difference was felt in traditional grammars, think e.g. about the status of the terms transitive/intransitive, which refers exclusively to objects allowed or not allowed for a certain verb.

1.1.1.2. Determination of the subordinates

Next we come to something like the reverse of the previous rule. Not only the function of the head plays a role but the function of the subordinate may equally well be of importance. This phenomenon corresponds to the notion of endocentric vs. exocentric constructions known from structural linguistics.

Take e.g. "the man in the café". Let us say that "in" has the grammatical function relationword as regards "man". But obviously we can say that if and only if there is a word with the grammatical function object as regards "in". So "in" needs the subordinate to have the function relationword. Similar cases are e.g. "he knocks the door down", where "knocks" needs "down" to become a transitive verb.

Having discussed the phenomenon we now turn to a discussion of a representation for the relation between the phenomenon and the factor function.

A possible solution for the representation problem goes as follows: We organize a grammatical rule that changes the function of the head of a grammatical relation as soon as the subordinate of the functional relation is present. E.g. we let 'from' have the function 'preliminary relationword' and change this into relationword if an object is there. Only then 'from' can start functioning as a relationword. Although this looks as being a nice solution (and it is the one used by categorial grammars e.g.) and although it works in this case, the need for another approach soon becomes obvious.

The point is that the not being active of a certain word holds up the whole analysis and this leads to dead situations during parsing. Consider e.g. the example of "he knocks the door down". Here "down" has to jump over "the door" to make contact with "knocks" and only then "the door" can be linked. But this jumping over is something we will not allow in the parsing process, and we have good reasons for that. So the analysis blocks: "the door" waits for "down" and "down" waits for "the door".

The other approach (which will be followed here) consists in associating with each function a state. If a function needs a certain subordinate we associate with it the state non-final. As soon as the required subordinate comes in, we change the state associated with the function to final. Obviously to be effective there should be a final state associated with each function at the end of the analysis.

We will define formal systems which are able to perform this sort of actions in the following section (1.1.2.2.) where we come to a discussion of order. The systems are called completion networks and a generalization over them completion automata.

1.1.2. Order

The next phenomenon is the use of order. Just as for the relations environment discussed in the previous subsection, we see again two types of rules:
(i) The first having to do with the order of the subordinate of the relation vs. its head,
(ii) the second having to do with the internal order of the subordinate of the same head.

1.1.2.1. Order of subordinate and head

The first phenomenon we investigate in relation to order is the following: Given a word w1 and a word w2, for w1 to have a particular grammatical function f as regards w2, w1 should be in a certain position as regards w2.

There are three possibilities:

(i) either w1 comes BEFORE w2
(ii) or w1 comes AFTER w2
(iii) or it is UNDETermined whether w1 comes before or after w2.

Again we introduce a grammatical rule in the form of a function this time called position which relates a grammatical function to one of the indicators BEFORE,AFTER, UNDET .

Definition

Let position: $F \to \{\text{BEFORE},\text{AFTER}, \text{UNDET}\}$ defined for $(\forall f)(f \in F\text{-adj} \cup F\text{-functw})$ be a function such that if w1 has the function f as regards w2, then if

$$\underline{\text{position}}\ (f) = \begin{cases} \text{BEFORE} & \text{w1 should come before w2} \\ \text{AFTER} & \text{w1 should come after w2} \\ \text{UNDET} & \text{w1 may come either before or after w2.} \end{cases}$$

With the difference in behaviour in mind between adjuncts/functionwords and objects, we investigate whether objects can be dealt with by this function *position*. But again it turns out that the position of objects is more easily determined by its headword whereas the position of adjuncts/functionwords is determined by the subordinate itself. Even more it is logically impossible to use the same function *position* because the position of the objects changes depending on the function of their head.

We call the linguistic function that relates a grammatical function to a position of its objects the *object-position* rule. Obviously it is only defined for those $f \in F$ such that *taking-objects*(f) = TRUE. We use again the indicators BEFORE, AFTER, UNDET meaning the objects come before their head, after their head or it is undetermined whether they come before or after it.

Definition

Let *object-position* : $F \rightarrow \{\text{BEFORE, AFTER, UNDET}\}$ be a partial function such that if w1 has the function object as regards w2 and w2 has the function f then if

$$\underline{\text{object-position}}(f) = \begin{cases} \text{BEFORE} & \text{w1 comes before w2} \\ \text{AFTER} & \text{w1 comes after w2} \\ \text{UNDET} & \text{w1 may come either before or after w2.} \end{cases}$$

(ii) Example

Let us take the same example sentence "a very urgent letter was sent to John" and specify the information for the rules position and object-position:

<u>position</u> (determiner) = before
<u>position</u> (adverbial adjunct)= before
<u>position</u> (finite auxiliary) = after
<u>position</u> (casesign) = before
<u>taking-objects</u> (nonfinite verb) = after

(iii) Discussion and further references

The fact that order is an important phenomenon has since long been recognized. In a categorial or constituent structure treatment, one would use phrase structure grammars, categorial grammars or equivalent systems to treat this order. In such systems this is done by giving a pattern in which the order relation is implicitly stated. E.g. if we say S → NP VP then this rule contains implicitly the information that the VP constituent comes after the NP .

One of the important consequences of making abstraction of the phenomenon of order as we did, is that this order can be controlled completely as an independent variable. We will see later an exiting experiment where we reverse the indicators before and after (i.e. consider as before 'coming after' and as after 'coming before') and where after that a sentence can be processed by the parsing system in right to left order !

The object-position rule is equivalent to the well known typology VSO,SOV,etc. although this may not seem to be so. First of all we have generalized over all predicates taking arguments (and not just the verb - subject - object relation). Second we consider the verb syntactically as the adjunct of one particular object, traditionally called the subject. The other objects are then all depending on the verb (cf. the example). Only then it is possible to apply the given formalism with only three theoretical terms: before, after, undet.

1.1.2.2. Internal order of subordinates

Now we come to the second usage of the phenomenon of order: the situation where the occurrence of one particular subordinate restricts the possibility that other subordinates may occur.

Take e.g. "translated the text". We know that translate cannot be att. adjunct here because its head (text) is although an object, linked with another word (the). This "the" has changed the structural properties of the object "text" to such an extent that it is no longer a valid head of an att.adjunct.

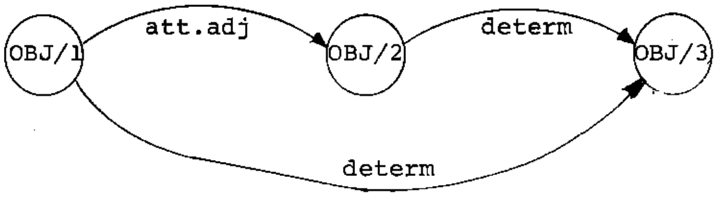
There are essentially two situations where the restriction of the internal order of the subordinates may occur:
(i) among adjuncts and functionwords (cf. the example)
(ii) among objects (consider "he gives me a book" and not "he gives a book me")
The first type will be discussed in this section, the second type will be treated later because the other factor namely case plays a very important rôle in it.

Having discussed the phenomenon we envisage for this module, we will now present a formal system in which these facts can be stated. This turns out to be a nontrivial task and we will introduce a new system called a completion automaton. The system is constructed in the tradition of automata theory but differs from already existing models in several respects. An essential part of the system (as for all automata) are the transition networks. Such transition networks will be called syntactic networks in the present context. But we will see later on that for the internal order of objects one can use the same formal system. Then we will call the networks semantic networks.

The intuitive ideas behind the use of the networks are that with a particular piece of input (e.g. a particular grammatical function in a structure) we associate a state.

When nothing is linked with the input piece the initial state is associated and whenever we make a link a new state (or more than one new states) are associated. In order to be a subordinate of a given word it is not sufficient then that this word has a particular grammatical function (as specified by the linguistic function <u>function-of-head</u>) or that it takes objects (as specified by the linguistic function <u>taking-objects</u>) and that the right order (as specified by <u>position</u> or <u>object-position</u>) is present, in addition a particular state should be associated with the head before the linking takes place.

Example: Given the function determ, att.adjunct and nom.obj then with nom.obj we associate the initial state OBJ/1. If the att.adj comes in we go from OBJ/1 to the state OBJ/2, if the determiner comes in we go from OBJ/1 to OBJ/3 or from OBJ/2 to OBJ/3.
Schematically:



Now consider as input "the translated text". Text starts with state OBJ/1;

with "translated" as att.adju we go to OBJ/2
with "the" as determ we go to OBJ/3 .

Now consider as input "translated the text". Text starts again with state OBJ/1

with "the" as determ we go to OBJ/3

and with "translated as att.adj we can go nowhere !

Notice that (in contrast to finite state automata and recursive transition networks) the network is written from the point of view of the head of the relation. Notice also that (again in contrast to existing automata) the position is considered to be not a part of the automaton, i.e. we only formalize relative order restrictions, not absolute order. The absolute order is of course captured by the previously discussed rules.

order

We now introduce the formal systems.

(1) COMPLETION NETWORKS

Definition

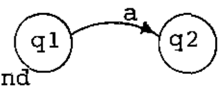
A completion network (CN) is defined by a quintuple
$CN = \langle Vn, Q, F, q0, \gamma \rangle$ with

(i) V a finite nonempty set of elements, the alphabet
(ii) Q a finite set of states
(iii) $F \subseteq Q$ the set of final states
(iv) $q0 \in Q$ the initial state
(v) $\gamma : Q \times V \rightarrow \wp(Q)$ the transition function

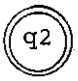
We define a graphical representation of a completion network as follows:

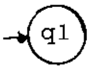
if $q1 \in \gamma(q2, a)$ with $q1, q2 \in Q$ and $a \in V$

then

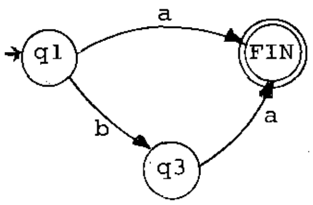


and

if $q2 \in F$ then we write 

if q1 is the initial state then 

Example



is the graphical representation of a CN $=\langle \{a,b\}, \{q1,q3,FIN\}, q1, \gamma \rangle$ with $\gamma(a,q1) = \{FIN\}$, $\gamma(q1,b) = \{q3\}$ and $\gamma(q3,a) = \{FIN\}$

There are a number of tasks that you can perform with a network. Two tasks will be of particular interest here: (i) the recognition of elements defined by the net and (ii) the ordering of elements of the alphabet into the format prescribed by the network. These two tasks both fall back on the "neutral" representation of the transition function as defined in the previous definition.

-i- the recognition task

The problem is given a string $p \in V^{\star}$, decide whether it contains the right element on the right place according to a given network.

We solve this problem by the introduction of information tuples called configurations and a relation over them, the reduction relation.

<u>Definition</u>

Let α be a <u>configuration</u> with $\alpha = \langle p, q \rangle$ and $p \in V^{\star}$ and $q \in Q$.
The <u>initial</u> configuration for a string $p \in V^{\star}$, called in(p) is $\alpha = \langle p, q0 \rangle$ with q0 the initial state
The <u>final</u> configuration for a string p called fin(p) is $\alpha = \langle \lambda , q_i \rangle$ with $q_i \in F$.

<u>Definition</u>

We define the <u>reduction relation</u> denoted as $\vdash\!\!-\!\!-$ as follows
Let α, β be two configurations $\alpha = \langle p, q \rangle$ with $p = a_1 a_2 \ldots a_{n-1} a_n$, $n \gg 1$ then

$\vdash\!\!-\!\!-_{R}$ (right going reduction)

iff $\beta = \langle p', q' \rangle$ and $p' = a_2 \ldots a_{n-1} a_n$ $, q' \in \gamma(q, a_1)$

order

$\vdash_L$ (left going reduction)

iff $\beta = (p' , q')$
and $p' = a_1 a_2 \ldots a_{n-1}$, $q' \in \gamma (q,a_n)$

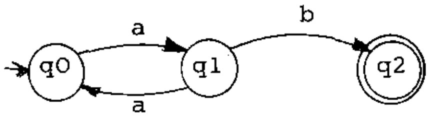
$\vdash = \vdash_L \cup \vdash_R$ and $\vdash^*$ denotes the reflexive and transitive closure of $\vdash$.

Definition

We define the left going reduction language of a CN C as LRL (C) $= \{ p \mid in(p) \vdash_L^* fin(p) \}$, the right going reduction language of a CN C as RRL(C) $= \{ p \mid in(p) \vdash_R^* fin(p) \}$ and the reduction language of a CN C as RL(C) = $\{ p \mid in(p) \vdash^* fin(p)$

Example

Given the CN



then RRL is $a^{2n+1}b$ $n \geqslant 0$

Example of operation

Let p = aaab , then(aaab,q0 $) \vdash_R \langle$ aab,q1 $\rangle \vdash_R \langle$ ab,q0$\rangle$ $\vdash_R \langle$ b,q1 $\rangle \vdash_R \langle \lambda , q_2 \rangle$.

LRL is ba^{2n+1} $n \geqslant 0$ and RL is $\{$ a,b in $|b| = 1$ and $|a| = 2n+1$, $n \geqslant 0$ $\}$.

-ii- The reordering of elements

The second problem is given an unordered series of input symbols compute as output one (or more) ordered sequences of this same input symbols. We will organize such a 'transduction' process as follows. First we distinguish an input vector in

order

which we find all symbols that are to be transmitted and the number of times that they will occur. Next we have a so called output string, i.e. the result of the process. Because of the nondeterministic property of completion automata it may be that more than one possible result is obtained. Hence we organize the process in terms of transduction. configurations containing an input vector and an output string. Then we define the transduction relation (formally represented as $\vdash$) which transforms one configuration into another one.

Here are the definitions:

First an auxiliary definition

Definition

Let V be an alphabet then an input vector I over V for a CN C is a set of pairs $I = \{\langle a,n \rangle a \in V,\ n \in N\}$
We say that $a \in V$ is in an input vector I iff $n > 0$ for $\langle a,n \rangle \in I$
An empty input vector is denoted as $\emptyset$.

Definition

Let α be a transduction configuration in CN when $\alpha = \langle a_1, a_2, a_3 \rangle$ with a_1 an input vector, $a_2 \in Q$ and $a_3 \in V^*$ the output string.

Definition

The transduction relation denoted as $\vdash$ is defined as follows:
Let α , β be two configurations $\alpha = \langle I, q, p \rangle$ and $\beta = \langle I, q', p' \rangle$ with $I = \langle a_1, n_1 \rangle , \ldots , \langle a_k, n_k \rangle$ $k \geqslant 1$ and $p = b_1 \ldots b_j$ $j \geqslant 0$;

order

$\xrightarrow[L]{}$ (left going transduction)

holds iff
1. a_i is in I
2. $I' = \langle a_1,n_1\rangle, \ldots \langle a_i,n_i-1\rangle, \ldots ,\langle a_k,n_k\rangle$
3. $q' \in \gamma(q,a_i)$
4. $p' = a_i b_1 \ldots b_j$

$\xrightarrow[R]{}$ (right going transduction)

holds iff
1. a_i is in I
2. $I' = \langle a_1,n_1\rangle, \ldots \langle a_i,n_i-1\rangle, \ldots ,\langle a_k,n_k\rangle$
3. $q' \in \gamma(q,a_i)$
4. $p' = b_1 \ldots b_j\ a_i$

$\xrightarrow{} = \xrightarrow[L]{} \cup \xrightarrow[R]{}$ and $\xrightarrow{*}$
denotes the reflexive and transitive closure of $\xrightarrow{}$.

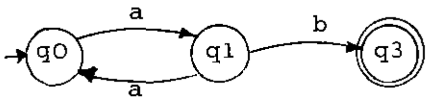
Definition

We define the left going transduction language of a CN C as $LTL(C) = \{ \langle I, q0,\lambda\rangle \xrightarrow{} \langle \emptyset,qf,p\rangle\ ,\ qf \in F \}$
The rightgoing transduction language of a CN C as $RTL\ (C) = \{ \langle I,q0, \lambda\rangle \xrightarrow{} \langle \emptyset,qf,p\rangle\ ,\ qf \in F \}$ and the transduction language of a CN C as $TL\ (C) =$
$\{ \langle I, q0,\lambda\rangle \xrightarrow{} \langle \emptyset, qf,p\rangle\ ,\ qf \in F \}$

It may be that the vector contains elements outside V or that a final state is reached with the input vector not being empty. In such a case the remaining input vector is called the rest. We will see that in practical applications this usage of the transduction relation is of interest.

Example

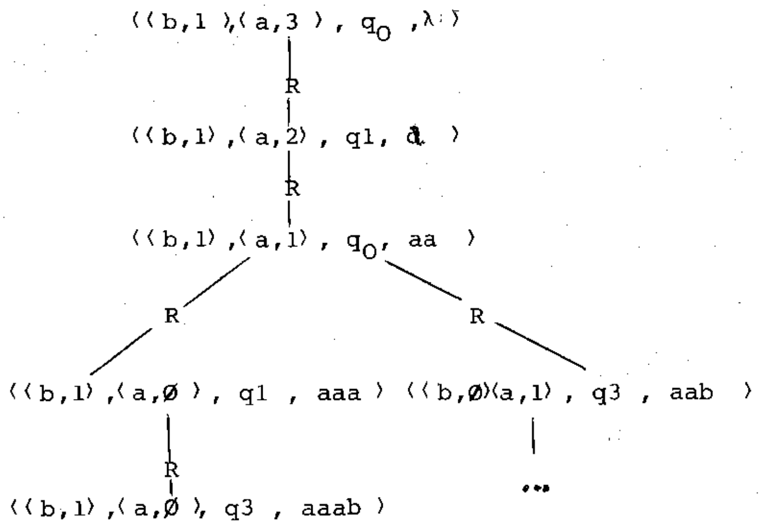
Given C



then $RTL(C) = \{a^{2n+1}b \quad n \geq 0\}$, $LTL\ (C) = \{ba^{2n+1} \quad n \geq 0\}$
and $TL(C) = \{p \mid p$ contains $\# b = 1$ and $\# a = 2n+1 \quad n \geq 0.\}$

order

Example of operation



(= valid end configuration)

So far we have presented the formal basis of completion networks. Let us before turning to the completion automata themselves discuss briefly the weak generative capacity of the present system. We do this only for the reduction languages because of the following theorem:

THEOREM 1

Let T be a completion network then LRL(T) = LTL (T) , RRL(T) = RTL (T) and RL(T) = TL (T) .

Proof

This follows immediately from the definitions.

In order to study the weak generative capacity we need the following auxiliary definitions:

Definition

Let CN denote the class of all completion networks then $\mathcal{L}_{LRL}$, $\mathcal{L}_{RRL}$, $\mathcal{L}_{RL}$ denotes the class of left reduction languages, right reduction languages and reduction languages respectively.

order

THEOREM 2

$$\mathcal{L}_{RRL} = \mathcal{L}_{REG}$$

Proof

The proof follows immediately from the definitions.

Lemma 1 $\mathcal{L}_{LRL} \subset \mathcal{L}_{REG}$

Proof

Let CN = $\langle V, Q, \gamma, q0, F \rangle$ then we define the equivalent FA $\mathfrak{A} = \langle V', Q', \gamma \;\; q0', F' \rangle$ where

(i) $V = V'$
(ii) $Q = Q' \cup \{q0'\}$
(iii) $q0'$
(iv) $F' = \{q0\}$
(v) Let $\gamma(q,a) = \{q_1, \ldots, q_n\}$, $q, q_1, \ldots, q_n \in Q$, $a \in V$
then
$\gamma'(q_1,a) = \{q\}, \ldots \quad \gamma'(q_n,a) = \{q\}$
and if $q_i, 1 \leqslant i \leqslant n$, $\in F$, $\gamma'(q0',a) = \{q\}$

The rest of the proof follows by induction on the number of steps in the application of the reduction relation.

Lemma 2 $\mathcal{L}_{REG} \subseteq \mathcal{L}_{LRL}$

Proof

Similar to lemma 2

THEOREM 3 $\mathcal{L}_{LRL} = \mathcal{L}_{REG}$

Proof

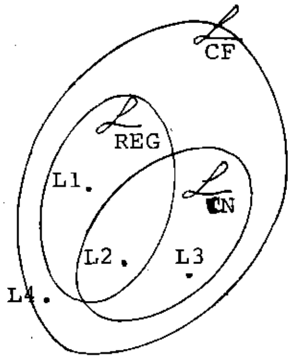
This result follows immediately from lemma 1 and lemma 2.

Now we study what happens if no strict order has been implied.

THEOREM 4 $\mathcal{L}_{CN} \subsetneq {}_{CF}$, $\mathcal{L}_{CN}$ and $\mathcal{L}_{REG}$ are incomparable but not disjoint.

Proof (due to D. Vermeir)

(1) L1 = $\{a\}\{b\}^{*} \in \mathcal{L}_{REG} \setminus \mathcal{L}_{CN}$ (see fig. 1)

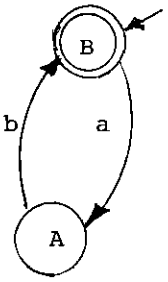


This follows immediately from the property (PROP 1) that if L $\in {}_{CN}$ then w $\in$ L implies mir(w) $\in$ L, where mir is the mirror imaage. Obviously the property holds.

(2) L2 = $\{a,b\}^{*} \in \mathcal{L}_{REG} \cap \mathcal{L}_{CN}$

Obvious.

(3) L3 is the language accepted by the following completion network:



Here the following holds:

(i) $\forall\ w \in L3 : \#_a(w) = \#_b(w)$

(ii) $\forall\ n \in \mathbb{N},\ v_n = a^n b^n \in L3$

(iii) Now recall the pumping lemma for regular languages: $(\forall L)\ \mathcal{L}_{REG}\ (\exists n)_N$:
$(x = y_1 z\ y_2 \in L,\ |z| \leqslant n+1$ implies that $\exists z' : z = z1\ z'\ z2$
and $y_1 z1 z'^{m} z2y2 \in L$, $\forall m \in N$)

Applying (iii) to words v_n(n large enough) yields words v'_n with $\#_a(v') > \#_b(v')$ and thus b.v' $\notin$ L3, consequently L3 $\in \mathcal{L}_{REG}$.

On the other hand L3 is generated by the following cfg G = $\langle \{A,B\}, \{a,b\}$, $\{A \to b\ B,\ Bb\ ;\ B \to a\ A, A\ a,\ \lambda\}$, B $\rangle$ thus L3 $\in \mathcal{L}_{CF}$

(4) L4 = $\{a^n b^n : n \geqslant 1\} \notin \mathcal{L}_{CN}$ because of property 1

(5) The fact that $\mathcal{L}_{CN} \subseteq \mathcal{L}_{CF}$ follows immediately from the obviously equivalent grammar representation (as a matter of fact $\mathcal{L}_{CN} \subsetneq \mathcal{L}_{LIN}$).

This ends the proof.

order

Comments:

In this section we defined a representational device called a completion network and two usages of the device: the reccognition and reorganization of a sequence of symbols.

2. Completion automata as generalized completion networks.

The earliest attempts to generalize over transition networks up to the level of type 2 systems is a recursive transition network (the formal basis of augmented transition networks). The idea is here to introduce as condition for a transition a whole network. By means of a pushdown store, you then store the current state before starting with the new embedded network and when a string has been recognized by this network you popup again and proceed with this earlier state.

We will now follow a quite different course of action. Instead of 'calling' the network of the embedding via another higher level network, we associate the transition networks (as defined before) with elements of the string itself ! An element is then allowed to be a condition of a transition iff it is in a final state.

Let us formalize all this in a set of new definitions. We call the system a completion automaton.

order

Definition

A completion automaton (CA) is defined by a quadruple:
$\mathcal{A} = \langle V, Q, RS, F, INIT, \gamma \rangle$ with

(i) V a finite nonempty set, the alphabet

(ii) Q a finite nonempty set of states $Q \cap V = \emptyset$

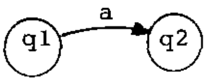
(iii) RS a set of reading states, $RS \subseteq Q$

(iv) F a set of final states, $F \subseteq RS$

(v) INIT: $V \to Q$ a partial function called the initial state assignment function

(vi) $\gamma : Q \times V \to \mathcal{P}(Q)$ the transition function

γ can be represented graphically as follows,
if $q1 \in \gamma(q2,a)$ with $q1,q2 \in Q$ and $a \in V$ then



Definition

Let α be a configuration in $\mathcal{A}$ when $\alpha = \beta_1 \ldots \beta_n$
with $\beta_i = \langle a_i, q_i \rangle$ for $1 \leqslant i \leqslant n$, $a_i \in V$ and $q_i \in Q$

Let $a_1, \ldots, a_n \in V$ for $n \geqslant 1$, then the initial configuration for a string $p = a_1 \ldots a_n$ denoted as in(p) =
$\langle a_1, q_1 \rangle \ldots \langle a_n, q_n \rangle$ such that INIT $(a_i) = q_i$, $1 \leqslant i \leqslant n$
If INIT $(a_i)$is undefined , q_i = FIN

A final configuration for a string $p = a_1 \ldots a_n$, denoted as fin$(p) = \langle a_i, q_j \rangle$ $1 \leqslant i \leqslant n$, $q_j \in F$.

Definition

We define the reduction relation for a CA $\mathcal{A}$ denoted as $\vdash$ as follows:
Let

$$\alpha = \langle a_1, q_1 \rangle \ldots \langle a_{j-1}, q_{j-1} \rangle \langle a_j, q_j \rangle \langle a_{j+1}, q_{j+1} \rangle \ldots \langle a_n, q_n \rangle$$

$(1 \leqslant j \leqslant n)$

order

then

(i) $\underset{L}{\vdash}$ holds if

1. $q_{j-1} \in RS \cup \{FIN\}$
2. $q_j' \in \gamma(a_{j-1}, q_j)$
3. $\beta = \langle a_1, q_1\rangle \ \ldots \ \langle a_{j-2}, q_{j-2}\rangle\langle a_j, q_j'\rangle$

$\langle a_{j+1}, q_{j+1}\rangle \ \ldots \ \langle a_n, q_n\rangle$

(ii) $\underset{R}{\vdash}$ holds if

1. $q_{j+1} \in RS \cup \{FIN\}$
2. $q_j' \in \gamma(a_{j+1}, q_j)$
3. $\beta = \langle a_1, q_1\rangle \ldots \langle a_{j-1}, q_{j-1}\rangle\langle a_j, q_j'\rangle\langle a_{j+2}, q_{j+2}\rangle \ \ldots$

$\langle a_n, q_n\rangle$

We call $\underset{L}{\vdash}$ the left going reduction relation and $\underset{R}{\vdash}$ the right going reduction relation.

$\vdash \; = \; \underset{L}{\vdash} \cup \underset{R}{\vdash}$ and $\overset{*}{\vdash}$ is the reflexive and transitive closure of $\vdash$.

Definition

The reduction language of a CA denoted as L(CA) = $\{p \mid \text{in}(p) \overset{*}{\vdash} \text{fin}(p)\}$, the left-going reduction language LRL(CA) = $\{p \mid \text{in}(p) \overset{*}{\underset{L}{\vdash}} \text{fin}(p)\}$, and the right-going reduction language RRL (CA) = $\{p \mid \text{in}(p) \overset{*}{\underset{R}{\vdash}} \text{fin}(p)$

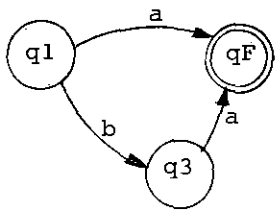
Let CA denote the class of completion automata then
$\mathcal{L}_{CA} = \{RL(\mathcal{Q}), \mathcal{Q} \in CA\} \cup \{LRL(\mathcal{Q}), \mathcal{Q} \in CA\} \cup \{RRL(\mathcal{Q}), \mathcal{Q} \in CA\}$

order

Example

Let $\mathcal{A} = \langle V,\ Q,\ qF\ ,\ qF\ ,\ INIT, \delta \rangle$ with $V = \{a,b\}$
$Q = \{q1, q3\}$, $INIT\ (b) = q1$
and

δ :



Then the left -going language of $\mathcal{A}$ is $a^n b^n \quad n \geqslant 1$

example of operation:

σ = aaabbb

$in(\sigma) = \langle a,FIN\rangle\langle a,FIN\rangle\langle a,FIN\rangle\langle b,q1\rangle\langle b,q1\rangle\langle b,q1\rangle$

$\vdash \langle a,FIN\rangle\langle a,FIN\rangle\langle b,qF\rangle\ \langle b,q1\rangle\langle b,q1\rangle$

$\vdash \langle a,FIN\rangle(a,FIN\rangle\langle b,q5\rangle\langle b,q1\rangle$

$\vdash \langle a,FIN\rangle\langle b,qF\rangle\ \ \langle b,q1\rangle$

$\vdash \langle a,FIN\rangle\langle b,q3\rangle$

$\vdash \langle b,qF\rangle$

$= fin(\ \sigma\ ).$

And the right-going language is $b^n a^n \quad n \geqslant 1$

Example of operation:

σ = bbbaaa

order

$$\mathrm{init}(\sigma) = \langle b,q1\rangle\langle b,q1\rangle\langle b,q1\rangle\langle a,FIN\rangle\langle a,FIN\rangle\langle a,FIN\rangle$$

$$\vdash \langle b,q1\rangle\langle b,q1\rangle\langle b,qf\rangle\langle a,FIN\rangle\langle a,FIN\rangle$$

$$\vdash \langle b,q1\rangle\langle b,q3\rangle\langle a,FIN\rangle\langle a,FIN\rangle$$

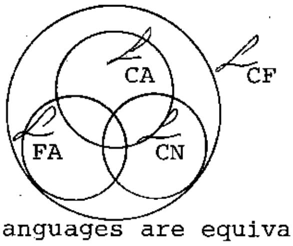
$$\vdash \langle b,q1\rangle\langle b,qF\rangle\langle a,FIN\rangle$$

$$\vdash \langle b,q3\rangle\langle a,FIN\rangle$$

$$\vdash \langle b,qF\rangle = \mathrm{fin}\ (\sigma)\ .$$

The detailed study of the formal properties of completion automata would lead us too far from the main subject of this works. We will present here a summary of the results, proofs and detailed discussion can be found in the references at the end of this section.

As regards the weak generative capacity we obtained a very interesting result . The weak generative capacity of completion automata is similar but not identical to completion networks, more in particular we have the following situation with unrestricted order:



The right-going reduction languages are equivalent with the context-free languages (compare this with completion networks) and the same result holds for the left-going reduction languages. The strong restrictedness is a very strong theoretical result especially from a linguistic point of view.

As regards the closure under AFL-operations we discovered that completion automata behave very awkward (no closure under union, etc;).

Finally we mention that the transduction relation can be defined just as for completion networks.

order

Discussion and further references

We have published quite a number of papers which show the evolution in our thinking about completion automata of which the latest stage has been presented here. See Steels(1975a),Steels(1975b),Steels and Vermeir (1976a) Steels(1976a),Steels (1976b),Steels and Vermeir (1977).

The formal difference with recursive transition networks lies in the point that networks are associated with elements of the input directly, rather than called via higher level networks. By this method we could (i) remove the concept of nonterminals from the automata, (ii) remove the necessity of having an independent memory, namely a pushdownstore. Although it may be difficult to see this at the moment, the advantages both for efficiency and power of the use of the presentation are enormous. Especially because (i) irrelevant parts of the network are not to be brought into the memory of the parser, (ii) due to the 'call by input' strategy unfruitful paths are cut out not by processing until they are dead but by the fact that they are simplv not called. Moreover we will see later that it provides for the first time the possibility of formalizing so called 'semantic parsing'.

The theory of completion automata of which only a little part has been shown (there are e.g. related completion grammars) is the first sort of results that can be obtained by applying the modular viewpoint to automata theory itself.

1.1.3. Concord

In natural languages it is common to associate certain features with the words of the language. These features which may show up by morphological affixes are used for various purposes in the language. One is the indication of functional relations (by the presence/absence of relationships between the features) or of case (by the use of so called case indicators).

The following points will interest us (i) how can we represent such syntactic feature complexes, (ii) how can we perform operations with such features, in particular the comparing of two complexes, and (iii) where are they used and for what purpose. The last question will only partially be considered, namely for functional relations where the subordinate is an adjunct or functionword. In such a situation the phenomenon of concord (or agreement) may occur: the features associated with the subordinate match with the features associated with the head. The other part (which we will be considering later) is that where the functional relation object holds (notice again the dichotomy between objects and adjuncts/functionwords). In such a situation the phenomenon of government occurs: the case relation prescribes the presence of certain syntactic features. This aspect is treated later when we have introduced the factor case.

Our first job now is the definition of a representation construct for features. This turned out to be very difficult but we feel to have found a powerful solution. For its introduction we invite the reader now to a short excursion in another area of mathematical linguistics : representation theory.

(i) Theory

Introduction to feature complexes

First we will analyse the requirements of a nontrivial representation of features (part1) then we will define the notion of a feature complex (part2) and some operations over feature complexes (part 3). Finally we will discuss the possibility of using an inference tree for cross reference (part 4).

part 1 : requirements

Consider the German (definite) article system which expresses information about (i) number (singular vs. plural) (ii) case (nomin, accus, dative, genitive), (iii) gender (male, female, neuter).
Instead of having 3 x 4 x 2 = 24 wordforms corresponding to each combination of features there are only 6 forms: der, dem, des, den, die, das.
Obviously one form has to have more than one function. In particular the following diagram represents the distribution of features over words.

singular :	male	female	neuter
NOM	der	die	das
GEN	des	der	des
DAT	dem	der	dem
ACC	den	die	das
plural :			
NOM	die	die	die
GEN	der	der	der
DAT	den	den	den
ACC	die	die	die

Such diagrams, well known from school grammars illustrate the point that complex features for one unit do not consist of only one sequence of features but of a set of sequences of features. However the diagrams are inadequate for certain purposes, because they are constructed so as to reflect the association between a sequence of features and a word but NOT to reflect what feature sequence is associated with what word. To know this we have to search through the whole diagram or we need an additional diagram as follows:

features

	CASE	GENDER	NUMBER
der	nom	male	sing
der	gen	female	sing
der	dat	female	sing
der	gen	male	plural
der	gen	female	plural
der	gen	neuter	plural

etc;

From this we conclude that it must be possible to associate with one unit (e.g. der) a set of sequences of features (requirement 1). We can represent this set by listing all the members(as is done in the above diagram) but obivously it would be stronger to have a more compact representation for one set, in which such generalizations as "all plural genitives have der" can be expressed (requirement 2). Note that such a compact representation would allow us to carry ambiguities around until they are resolved, something which we feel to be very important, especially for an analysis procedure.

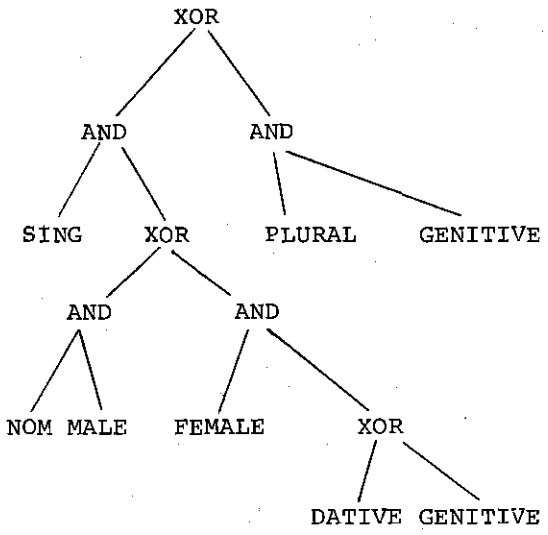
In an operational system it must be possible to do something with complex features. The most common operation is that two complexes of features are matched, e.g. the feature complex of a determiner is matched with that of a nomen. Or the feature complex of an object is mathed with a feature complex representing the selection restrictions in the case slot. There is one important aspect about this matching, namely relevance: only those feature are considered which are relevant for a particular matching process.
By relevance we mean that only a subsequence (maybe even only one element of a feature sequence) is checked and the rest is not important in the final decision. E.g. given the requirement that feature complex 1 contains feature A and feature complex 2 contains A and B, then feature complex 1 matches with feature complex 2 because A is in A and B, but not the reverse, B is not in the feature complex1.
We will need a special kind of relevance logic for this (requirement 3).

Another useful operation is the combination of two feature complexes to form a new one. This happens e.g. if a new semantic unit is formed which has the properties of its components. In other words the operation of combining two feature complexes must be available (requirement 4).

Needless to say that to design a representation construct that meets requirement (1-4) is a nontrivial task. In this work we will propose a possible solution.

Intuitively the representation construct constitutes a tree where the nonterminal nodes contain directions (AND, OR, XOR (= exclusive OR), NOT) and the terminals the features themselves.

Example (for der)



The idea is that to find whether the unit to which the tree corresponds contains the features being looked for, one walks through it and performs matches with the terminal nodes. On the other hand, when you want to know what sequence of features is associated with the unit, you compute the extension of the tree, i.e. the set of sets of features that corresponds to it.

Intuitively AND means both sides are members of a sequence, OR means both sides are members but one may be not, XOR means only one side constitutes the members of a sequence, NOT means that the depending sequence is not in the feature sequence.

Before we now turn to a more exact account of the formalism it must be noted that we will use again our standard convention for representing trees in a linear expression by means of the list-notation introduced earlier.
So, for the example tree of DER we get

```
(XOR(AND SING (XOR (AND NOM MALE)
                   (AND FEMALE (XOR DATIVE GEN))))
    (AND PLURAL GENITIVE)   )
```

part 2: Definitions

(a) Syntactic definitions

We define the formal outlook of a feature complex (for short FC) by a context-free grammar which generates the linear representation of an FC.

<u>Definition</u>

Let FCG $= \langle$ Vn, Vt, P, $\langle$ FC $\rangle\rangle$ be a context-free p.s. grammar with

Vn = $\{\langle$ FC $\rangle\}$
Vt = $\{$(,),AND,OR,NOT,XOR$\} \cup$ FS where FS denotes the set of features

P contains the following productions:

1. $\langle$ FC $\rangle$ → A A $\in$ FS
2. $\langle$ FC $\rangle$ → (AND $\langle$ FC $\rangle\langle$ FC $\rangle$)
3. $\langle$ FC $\rangle$ → (OR $\langle$ FC $\rangle\langle$ FC $\rangle$)
4. $\langle$ FC $\rangle$ → (XOR $\langle$ FC $\rangle\langle$ FC $\rangle$)
5. $\langle$ FC $\rangle$ → (NOT $\langle$ FC $\rangle$)

(Note that the brackets are terminal symbols !)

The set of feature complexes as a whole is then L(FCG).

Example

Let FS = $\{$SING, NOM, MALE, PLURAL, DAT, GEN, FEM$\}$
then
(AND (XOR (NOT NOM) MALE) GEN$)\in$ L(FCG) .

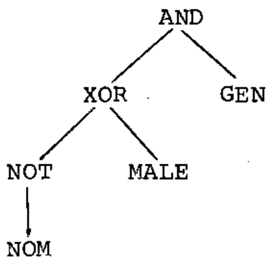
Proof:

$\langle FC\rangle \Rightarrow$ (AND $\langle FC\rangle$ $\langle FC\rangle$) $\Rightarrow$ (AND (XOR $\langle FC\rangle\langle FC\rangle$)$\langle FC\rangle$)

$\Rightarrow$ (AND (XOR (NOT $\langle FC\rangle$)$\langle FC\rangle$)$\langle FC\rangle$) $\overset{+}{\Rightarrow}$

(AND (XOR (NOT NOM) MALE) GEN)

which is equal to the following tree:



The main 'trick' now is to define an extensional AND truthlogical semantics for the expressions. The extensional interpretation yields the set of sequences of features which are expressed in a compact feature complex (cf. requirement 1 & 2) . On the other hand the truthlogical interpretation for the same expressions yields a truthvalue, using the 'relevance' idea (cf. requirement 3).

(b) Semantics

-i- extensional

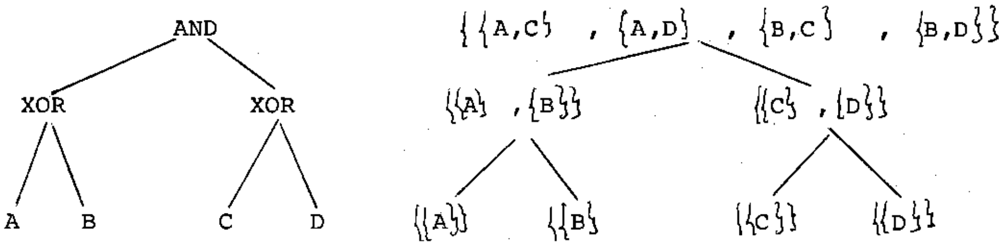
Feature complexes are the representation of sets of sets of features. Each FC represents therefore the complete characterization of a possible feature combination. Let us define this set interpretation of an FC, denoted as ext (FC) as follows:

Definition

1. $\underline{\text{ext}} \llbracket A \rrbracket = \{\{A\}\}$ with $A \in FS$
2. $\underline{\text{ext}} \llbracket (\text{AND } X\ Y) \rrbracket$ with $X,Y \in L(FCG)$
   $= \{(X' \cup Y') \text{ with } X' \in \underline{\text{ext}} \llbracket X \rrbracket ,\ Y' \in \underline{\text{ext}} \llbracket Y \rrbracket\}$
3. $\underline{\text{ext}} \llbracket (\text{OR } X\ Y) \rrbracket$ with $X,Y \in L(FCG)$
   $= \text{ext} \llbracket (\text{AND } X\ Y) \rrbracket$
4. $\underline{\text{ext}} \llbracket (\text{XOR } X\ Y) \rrbracket$ with $X,Y \in L(FCG)$
   $= \underline{\text{ext}} \llbracket X \rrbracket \cup \text{ext} \llbracket Y \rrbracket$
5. $\underline{\text{ext}} \llbracket (\text{NOT } X) \rrbracket$ for $X \in L(FCG)$
   $= \emptyset$

Example

Let FC = (AND (XOR A B) (XOR C D)) .
The tree on the right contains for each node the semantic interpretation of the corresponding node in the tree on the left



$\text{ext} \llbracket FC \rrbracket = \{\{A,C\} , \{A,D\} , \{B,C\} , \{B,D\}\}$

Note:
Perhaps a more exact account of $\underline{\text{ext}} \llbracket (\text{OR } X\ Y) \rrbracket$ would be
$\underline{\text{ext}} \llbracket (\text{XOR } (\text{AND } X\ Y)\ (\text{XOR } (\text{AND } X\ (\text{NOT } Y))\ (\text{AND } (\text{NOT } X)\ Y))) \rrbracket$
In the application however the simplification as introduced in the main definition never lead to any problems.

- ii - truthlogical

Definition

The domain of an FC is a set of sets

Definition

Let $X \in L(FCG)$ and D a domain, then we define the truth-value of an FC as regards a domain D, denoted as $\underline{eval}$ (X, D)
as follows:
First we define $\underline{eval}'(X, d_D)$ for an arbitrary $d_D \in D$

1. $\underline{eval}'(X, d_D)$ for $X \in FS$

$$= \begin{cases} \text{TRUE} & \text{if } X \in d_D \\ \text{FALSE} & \text{otherwise} \end{cases}$$

2. $\underline{eval}'(Z, d_D)$ for $Z = (\text{AND } X\ Y)$ and $X, Y \in L(FCG)$

$$= \begin{cases} \text{TRUE} & \text{if } \underline{eval}'(X, d_D) \ \underline{\text{and}}\ \underline{eval}'(Y, d_D) \text{ is true} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

3. $\underline{eval}'(Z, d_D)$ for $Z = (\text{OR } X\ Y)$ and $X, Y \in L(FCG)$

$$= \begin{cases} \text{TRUE} & \text{if } \underline{eval}'(X, d_D)\ \underline{\text{or}}\ \underline{eval}'(Y, d_D) \text{ is true or both} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

4. $\underline{eval}'(Z, d_D)$ for $Z = (\text{XOR } X\ Y)$ and $X, Y \in L(FCG)$

$$= \begin{cases} \text{TRUE} & \text{if } \underline{eval}'(X, d_D) \text{ or } \underline{eval}'(Y, d_D) \text{ is true but not both} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

5. eval' (Z, d_D) for Z = (NOT X) and X $\in$ L(FCG)

$$= \begin{cases} \text{FALSE} & \text{if eval}'(X,\ d_D) \ \text{is true} \\ \text{TRUE} & \text{otherwise} \end{cases}$$

Now we generalize over eval' as follows:

eval(X, D) for X $\in$ L(FCG)

$$= \begin{cases} \text{TRUE} & \text{if eval}'\ (X,\ d_D) \ \text{is true for at least one } d_D \in D \\ \text{FALSE} & \text{otherwise} \end{cases}$$

Example

Let D = $\{\{A\}\}$ and FC = (OR A (NOT A)) then eval (FC,D) = TRUE

Proof:
Let $d_D \in$ D be $\{A\}$, eval'(A, $\{A\}$) = TRUE and eval'((NOT A), $\{A\}$) = FALSE, so eval' ((OR A (NOT A)), $\{A\}$) = TRUE.
So eval (FC , $\{\{A\}\}$) is true

Example

Let D = $\{\{A,C\},\{A,D\}, \{B,C\}, \{B,D\}\}$ and FC = (AND (XOR A B) (XOR C D)) then eval (FC , D) is true.

Proof:
Let $d_D \in$ D be $\{A, C\}$
then (i) eval'(A, $\{A,C\}$) = true and (ii) eval' (B, $\{A,C\}$) = false. So (iii) eval'((XOR A B), $\{A,C\}$) = true (from (i) and (ii)).
Moreover (iv) eval'(C, $\{A, C\}$) = true and (v) eval'(D , $\{A, C\}$) = false.

So (v) eval' ((XOR C D) , {A,C}) = true (from iv and v)
and therefore eval'((AND (XOR A B) (XOR C D)) , {A,C}) = true
(from iii and v).
This ends the proof.

To illustrate the relation between the truthlogical and set theoretical interpretation of FC's a small table illustrating some sample relationships in detail is presented.
In the table: eval(Y, ext(X)) with Y on the lines and X on the columns.

	A	(AND A B)	(OR A B)	(NOT A)	(XOR A B)
A	T	T	T	F	T
(AND A B)	F	T	T	F	F
(OR A B)	T	T	T	F	T
(NOT A)	F	F	F	T	T
(XOR A B)	T	F	F	F	T

part 3: operations

-i- Matching

Feature complexes are used in linguistic systems in the context of tests investigating whether two feature combinations match. For this purpose FC's as formalized in previous sections are particularly useful, because now we can define exactly what nontrivial matching is about.

Definition

Let FC1, FC2 be two feature complexes then we say that FC1 matches with FC2 if and only if eval(FC1, ext(FC2)) is true.

Note that according to the definitions the functions pick out those features of FC2 which are relevant as regards FC1 and not vice-versa. E.g. if the adjective agrees only in gender, say, with the noun, then whatever other information may be contained in the FC associated with the noun, only that feature will determine the truth value.
Note also that we can compare complexes of features with each other and in both directions.

In some cases it may be important to remember for what subsets of the domain the two feature complexes match. E.g. if the determiner matches with the noun, then a verb later on should match with the same subsets as was the case for the determiner. We call the sets for which a match resulted in true the satisfied domain.

Definition

Given two feature complexes FC1 and FC2 then the satisfied domain is

$\{ d \mid d \in \underline{ext}(FC2) \text{ and } \underline{eval}'(FC1, d) \}$ is true

- ii- Combination

We finally discuss the notion of combination

Definition

If FC1 and FC2 are feature complexes and G1 = ext (FC1) and G2 = ext (FC2) then the extensional combination of FC1 and FC2 denoted as $comb_e(FC1,FC2) = \{ Y \cup Z \mid Y \in G1, Z \in G2. \}$

Inference trees

So far cross classification was formalized as a local process: As soon as certain features appear we make inference by considering only that part of the tree further on that contains the features already present. This works out very well for such applications as concord where cross classification is typically local. But in other situations (e.g. semantic feature matching) it may be of interest to have a global cross classification, in other words if, say +HUMAN, is present in a feature complex that we can match this with +ANIMATE, without the need to say in each feature complex (AND HUMAN ANIMATE).

We therefore introduce an additional tool in the representation language of feature complexes, namely global inference rules which are applied embedded in the calculus itself.

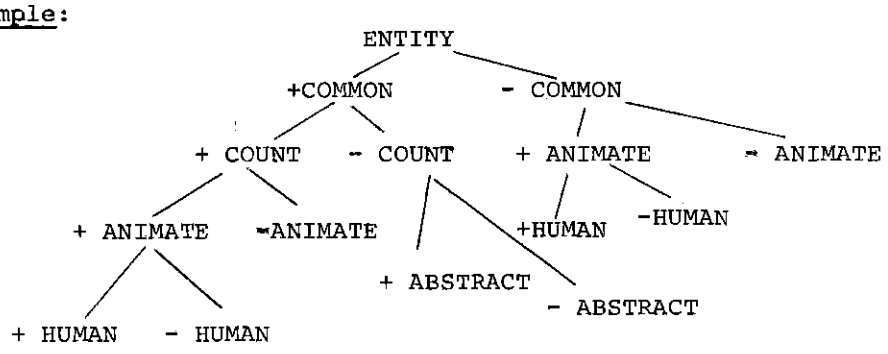
First we define a representation for the inference rules , the so called inference tree, then we define how it can be applied during the matching of feature complexes.

(i) Inference trees

Definition

An inference tree is a tree in the usual sense with features on the nodes.

Example:



Definition

The list representation of an inference tree is the standard list representation of a tree as defined earlier.

(ii) Evaluation

The only thing we have to redefine as regards the given definition of eval in the feature complex calculus is the truthlogical interpretation.

Recall that

$$\text{eval}'\,(X, d_D) \quad \text{for } X \in FS$$

$$= \begin{cases} \text{TRUE} & \text{if } X \in d_D \\ \text{FALSE} & \text{otherwise} \end{cases}$$

Now we extend this as follows

Definition

The father of a node X, denotes as father(x) is the node immediately dominating a node X.

The fathers of X denoted as fathers(X)

$$= \{Y \mid Y = \text{father}(x) \text{ or } Y = \text{father}(x'), x' \in \text{fathers}(Y)\}$$

Definition

$$\text{eval}'\,(X, d_D) \text{ for } X \in FS$$

$$= \begin{cases} \text{TRUE} & \text{if } X \in d_D \quad \text{or } (\exists x)_{d_D} \quad (x \in \text{fathers}(X)) \\ \text{FALSE} & \text{otherwise} \end{cases}$$

The rest of the definitions remains the same.

The use of syntactic features

We now have a way to represent and compare feature complexes with each other. Let us now discuss their role in language. It turns out that the discussion can best be split up in three parts according to the major classes of functions: object, adjuncts and functionwords.

(i) Objects

With each object a particular feature complex is associated right from the start. This feature complex contains at least all the possible feature constellations as regards gender, number and case.
Th ambiguity present in the feature complex of the object is during analysis restricted or extended.

(i) restricted by all subordinates for which the concord rule applies (each subordinate defines a subset of the feature sets of the object) and by the surface case signal tests (see later) which further restrict the case indicators in the feature complexes;

(ii) extended by means of a rule (to be defined soon) by which features of a word are attached to the feature complex of the object. E.g. a case sign sends some signal to the feature complex of its head. The indefininte article may send the feature 'undefinite' to the feature complex of the object, etc.

(ii) Functionwords

The task of restricting or extending the feature complex of objects seems to be the main task of words having the function functionword. Indeed it can be said that it is their only purpose of being there.

(iii) Adjuncts

A more complicated situation occurs with the adjuncts. They seem to have the behaviour of both objects and functionwords as regards features. On the one hand adjuncts restrict the feature

complex of their heads, e.g. the verb 'sleeps' in 'the sheep sleeps' restricts the ambiguity of 'sheep' (sing or plural) to only singular.
But on the other hand verbs e.g. have a feature complex on their own which contains such things as future, perfective or other modification items.
The latter feature complex is also subject to restrictions and extensions, either by other adjuncts or by functionwords.

It follows that

(i) with objects we associate in the lexicon one feature complex subject

(ii) with functionwords we associate in the lexicon one feature complex that is itself not subject to change during analysis but which itself evokes the change;

(iii) with adjuncts we associate in the lexicon two feature complexes:

-a- one used to restrict the feature complex of others (we call this the external feature complex),

-b- one that is associated with the adjunct itself (we call this the internal feature complex) and objects have only an internal syntactic feature complex according to this terminology).

We need some additional rules to cover the use of syntactic features as described above. First a rule saying whether there is concord or not.

<u>Definition</u>

Let <u>concord</u>: $F \rightarrow \{\text{TRUE, FALSE}\}$ be a function such that

$$\underline{\text{concord}}\ (f) = \begin{cases} \text{TRUE if the feature of the} \\ \quad \text{word having the function should} \\ \quad \text{match with the features associated} \\ \quad \text{with the head} \\ \text{FALSE otherwise} \end{cases}$$

the function is defined $(\forall f)\ (f \in \text{F-adju} \cup \text{F-functw})$

Second a rule telling whether synt. features are sent through

Definition

Let send-through : F → {TRUE, FALSE} be a function such that

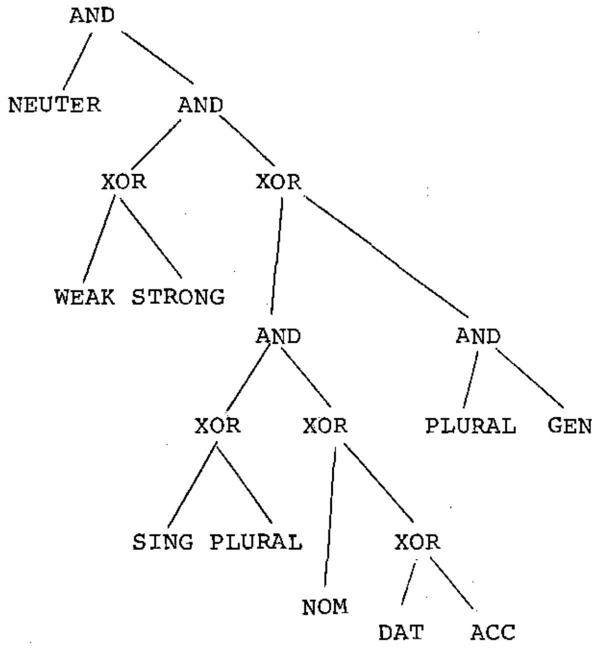
$$\text{send-through}(f) = \begin{cases} \text{TRUE} & \text{implies that features of the subordinate are to be attached to the internal feature complex of the head} \\ \text{FALSE} & \text{implies no action} \end{cases}$$

(ii) Example

To see the functioning of feature complexes in the language system, consider the following example from German (all feature complexes are due to K. Lambrechts):

" (Er setzte sich) neben ein fremdes Fraülein"

We start with "Fraülein" having the feature complex:

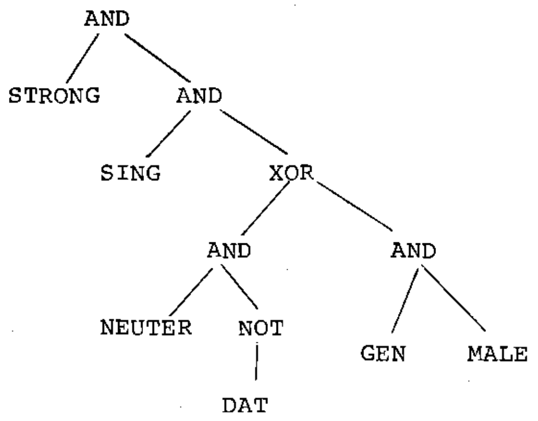


with extension:

```
((NEUTER STRONG SING NOM)  (NEUTER WEAK SING NOM)
 (NEUTER STRONG SING ACC)  (NEUTER WEAK SING ACC)
 (NEUTER STRONG SING DAT)  (NEUTER WEAK SING DAT)
 (NEUTER STRONG PLURAL ACC)  (NEUTER WEAK PLURAL ACC)
 (NEUTER STRONG PLURAL DAT)  (NEUTER WEAK PLURAL DAT)
 (NEUTER STRONG PLURAL GEN)(NEUTER WEAK PLURAL GEN))
```

So 14 possibilities.

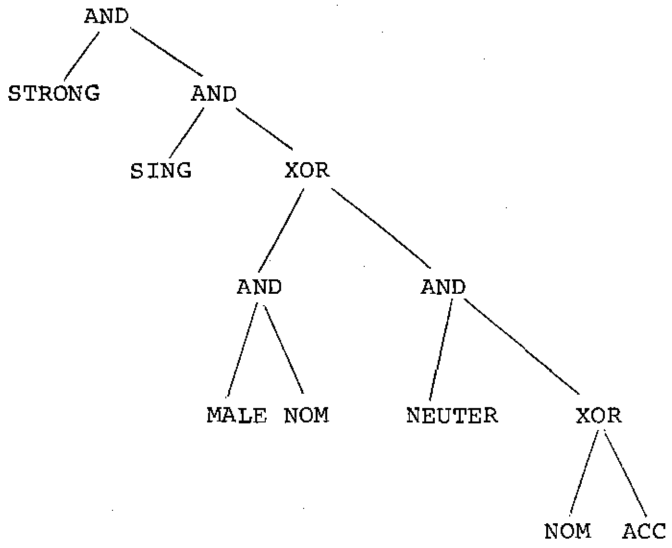
Then "fremdes" comes in with feature complex:



This feature complex matches with that of the following subsets of the domain:

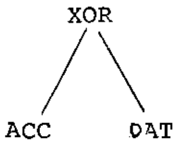
((NEUTER STRONG SING NOM)
(NEUTER STRONG SING ACC))

So we are left with 2 possibilities then the word "ein" comes in with features:



"ein" matches with the same subsets of the domain, so it does not help us any further.

Finally we have "neben" with features:



and we are left with only one satisfied subset:

((NEUTER STRONG SING ACC))

This reduction from 14 to 1 possible feature sequence is typical for the functioning of the feature matches and it is extraordinary that for such a complex feature system as used in German the efficiency for removal of ambiguity is almost 100 %. Notice that we worked from right to left here. It is possible to go from left to right also, although then the processing becomes more complex.

(iii) Discussion and further references

The concord phenomenon has since long been recognized as being an essential feature of language functioning. In some languages (e.g. German and Latin) it plays a much more important role than in others (e.g. English). That may be the reason why in most linguistic theories from Anglo-American origin concord is treated rather badly (consider e.g. T.G.).

The representation construct we have introduced here is we believe the first nontrivial approach towards the problem of feature representation within a formal framework. We are currrently using this calculus not only for syntactic feature matches but at several other places in the theory and more in particular at every point where a complex specification is given.

The feature complex calculus was applied to concord within the German nominal group. Results appear in Lambrechts and Steels (1977). Some more examples will be given later.

The idea of cross classification is already present in existing grammars,especially for the cross classification of semantic features or selection restrictions (the tree is a translation of the tree in Chomsky,1965,83). According to the general spirit of integrative grammars such a cross classification was incorporated in the grammar itself by means of rewriting rules ! Obviously it is more powerful to let the cross classification be active over the whole language, after all that is what cross classification is about.

SUMMARY OF SECTION 1.1.

In this first subsection we presented the first pieces of a modular grammar. In particular some rules having to do with function.

We have first of all defined the notion of function and a representation construct for functional relations in the sentence (1.1.0) . Then we introduced some modules related to the functional environment of a particular function (1.1.1.) . In particular how the function of the subordinate may be determined by the occurrence of a functional relation of the head (1.1.1.1) and how the function of the subordinate may be determined by the occurrence of other functional relations for the same head (1.1.1.2).

The second phenomenon was that of order (1.1.2.). First we investigated how the order of a subordinate is determined as regards its head (1.1.2.1.), and second how the subordinates themselves may have an internal order (1.1.2.2.).

The third phenomenon we have investigated is that of syntactic feature concord (1.1.3.).

In the following sections we go on with the presentation of more rules. But now a second factor comes in, namely case. In a first subsection we introduce this new factor.

1.2. CASE

1.2.0. Introduction to case

Although the theory of cases will here be introduced in connection to the words of the language themselves, it should be noted that there is a 'semantic counterpart' to the terms and concepts. This counterpart will be presented later on.

Definition

Let us consider a finite nonempty set of words W over a language, then the *case relations over W*, denoted as CR, is a relation in the set theoretic sense , $CR \subseteq W \times W$.

If $\langle w1,w2 \rangle \in CR$ then we say that a *case relation holds between* w1 *and* w2.

We furthermore distinguish subsets in CR, where each subset defines a particular case relation. If a particular case relation say $C \subseteq CR$ holds between w1 and w2, then we say that w1 *has the case* C *as regards* w2 or that w1 *is a C of* w2. w1 is called the *(slot)filler* and w2 the *frame carrier* of the relation pair $\langle w1,w2 \rangle$.

If $\langle w1,w2 \rangle \in CR$ then we say that the *empty case*, denoted as NIL holds between w1 and w2.

Example

In "(the) boy sings" a case relation holds between "boy" and "sings". This particular case relation is often called the AGENT case. We say that "boy" is the slot filler and "sings" the frame carrier and that "boy has the case AGENT as regards "sings" or simply that "boy" is the agent of "sings".

(Comment: compare these definitions with those of the notion of grammatical function.)

We now bring the notion of case in relation to a sequence of words:

Definition

Let $w_1 \ldots w_n$ be a sequence of words then the case structure of $w_1 \ldots w_n$ is defined as follows:
if n = 1 then the case structure is empty
if n is greater than 1 the case structure is the set of all pairs $\langle w_k, w_{k+1} \rangle$ such that a case relation holds from w_k to w_{k+1}.

(Note the lack of any further restrictions compared to the functional structure defined earlier)

Example

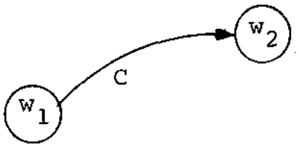
Given the sentence "John consulted the edited translation", then

the case relation AGENT holds between "John" and "consulted"
the case relation SOURCE holds between "translation" and "consulted"
the case relation SOURCE holds between "edited" and "translation".

Now we define a graph representation for case structures following the standard mathematical conventions.

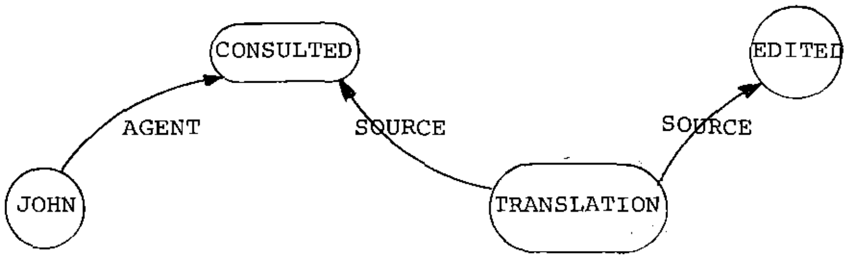
Convention

If a case relation C holds between w1 and w2 we draw a node for w1 and w2, if such nodes did not yet exist, and label it with w1 and w2 respectively. Then we draw a directed line between the nodes and label the line with C:



Example

For "John consulted the edited translation":



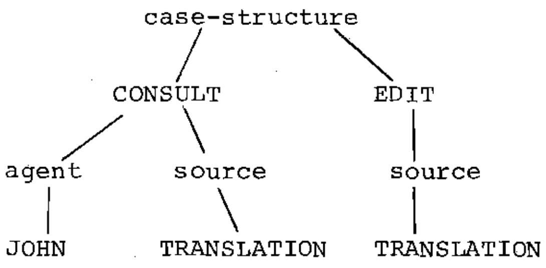
The reader may recall that we introduced a simplification in terms of trees of the graph structure representing functional structures. This simplification is now impossible because the conditions that guaranteed the possibility of performing the simplification are no longer fulfilled. In particular there is not necessarily a unique topnode as is illustrated in the example. However it is possible to apply the following operation on the graph which yields a tree structure, albeit that it does not reflect the graph structure anymore.

Convention

A subtree is constructed by grouping all pairs (a_j, b_i) $1 \leqslant i \leqslant n$ with a_j the top and b_i all the branches such that (a_j (case$_1$ b_1) ... (case$_n$ b_n)).
All subtrees are then grouped under one top with label case structure.

Example

For "John consulted the edited translation":



Now we introduce a number of additional concepts related to these case relations by making gradually further abstraction of the surface account given above.

(1) Predicates

If we investigate in more detail the case relations that hold in the language, certain regularities can be discovered in that a number of words all have the same particular cases. To capture this regularity we introduce abstract predicates which are directly related to the words themselves.

The idea is that the case relations of a language are not expressed in the grammar in terms of the actual words but rather in terms of the predicates associated with these words. We will see later that these predicates play a very important role in the semantic processing.

In order to enable us to speak about the predicates of a word w, we define a function assigning a predicate to a word (and one word may have different predicates).

Definition

Let W bet the set of words and P the set of predicates, then predicate: $W \rightarrow \mathcal{P}(P)$ is a function.
We say then that the case relation $\langle w1,p \rangle$ ¯ $w1 \in W$ and $p \in P$ holds if $\langle w1,w2 \rangle \in CR$ and predicate(w2) = p.

(2) Argument slots

In order to specify in the grammar what particular case relations holds, we introduce an auxiliary notion, that of an argument slot. We denote an argument slot by the sign $\sqcup_i$ where i is an index.

Just as the notion of predicate, the concept of an argument slot can only be understood in a semantic context (cf. supra), nevertheless we introduce it here making abstraction of these deeper motivations.

An argument slot is simply an 'open place' that can be filled (under certain conditions). For the moment we say that words fill this open place.

Definition

Let AS be the set of argument slots, then we say that a potential case relation $\langle \sqcup_i, p \rangle$ holds iff

$$(\exists w1)\ (\exists w2)\ (\langle w1, w2\rangle \in \mathrm{CR} \text{ and predicate } (w2) = p)$$

where $p \in P$ and $w1, w2 \in W$.

In addition we introduce a label function that assigns a case label to each member of a particular case relation:

Definition

Let label: AS x P → L with L the set of labels be defined as follows label $(\sqcup_i, p) = C$ iff $\langle \sqcup_i, p \rangle$ holds with $\sqcup_i \in$ AS and $p \in P$.

We have now made abstraction of both members of a case relation. Now comes the next step: to make abstraction of the case structures.

(3) Case frames

The regularity mentioned before was such that the same set of cases occurred for a number of words in the language. It follows that we need a way to state explicitly what cases occur with what predicates. We call such a statement a case frame.

Definition

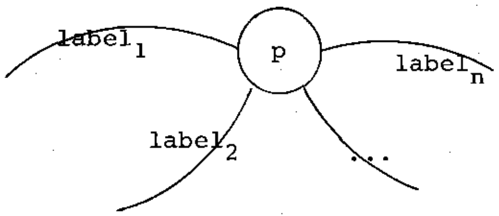
A case frame $\langle p, \sqcup_1, \ldots, \sqcup_n \rangle$ is an n-tuple $1 \leqslant i \leqslant n$ where $p \in P$ and $\langle \sqcup_i, p\rangle$ is a potential case relation .

Convention

Let $\langle p, \sqcup_1, \ldots, \sqcup_n \rangle$ be a case frame then we normally write

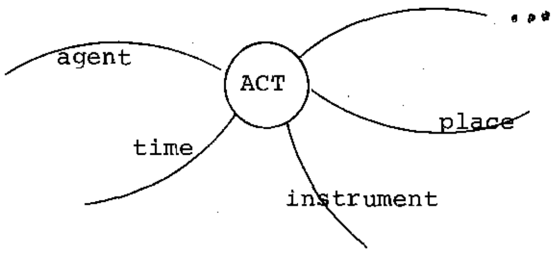
$(p \quad label_1 \ \ldots \ label_n)$ for $label(\langle \sqcup_i, p \rangle) = label_i$

Graphically:



Example

Let (ACT agent time instrument place ...) be a case frame then we represent this graphically as



On the relation between grammatical functions and cases

If we compare the functional relation that holds between two words and then parallel to it the case relation we discover that there are two situations:
(i) The grammatical function between filler and frame carrier is one between subordinate and head. This is the best known situation and it is often the only one take into account.
Examples are "he gives a book to John", "a book" and "to John" both fill a slot in the case frame of "gives" and are functionally both subordinates of "gives" .

(ii) The grammatical relation between filler and frame carrier is one between head and subordinate. So, the exact reverse ! An example is "the translated text" where "text" fills a slot in the frame of "translate", although "translate" is a subordinate of"text".

This second sort has been considered in the past as less fundamental than the first one, some theories (particularly in a transformational context) express all case relations as relations of the first sort, where transformations are applied to bring the second sort in the format of the first . We do not see any reason for that. Both sorts are equally valid, although the strategies to parse the first sort are quite different from those of the second one.

On the relation between case structures and surface phenomena

In the next paragraphs we study the surface phenomena which the language producer is using to signal the presence of certain functional relations and certain case relations. These surface phenomena are:

(i) Each potential case relation implies the occurrence of certain semantic properties for the candidate filling the slot;

(ii) each potential case relation implies the occurrence of certain syntactic signals (case signs, morphological affixes word order) for the candidate filling the slot.

Before we can discuss in detail how these two phenomena are determined we have to introduce the two factors which play a role in that. These two factors are

(i) the communicative function of the predicate,

(ii) the viewpoint by which the case frame is related to the rest of the semantic information.

(a) The communicative function of the predicate

In a communication situation a predicate can be used for various purposes: it can be used to introduce an entity or a class of entities to the listener, to modify or amplify other predicates, to give more information about an already introduced entity, etc. As should be clear from the previous sections ,in our linguistic theory communicative functions are studied under the heading of grammatical functions.

We have indicated that with each of these grammatical functions there corresponds a number of surface phenomena. This section is a continuation of this discussion but the notion of case is now a supplementary factor.

Recall that there are three main classes: objects, adjuncts and functionwords. As functionwords are words not introducing per definitionem any new semantic predicates they can be left out of the present discussion.

(b) The viewpoint of the predicate

The second factor is the viewpoint of the predicate (in some earlier publications we have called this the informative function of the predicate). The viewpoint of a predicate is the way in which the predicate is related to the rest of the information. This differs slightly from one function to another.

(i) When the function of the predicate is the introduction of entities (i.e. the predicate has the function object) then the viewpoint is the case relation that holds between the entity that is being introduced and the predicate.
E.g. take the case frame of TRANSLATE with cases self, agent, source, result, then the viewpoint is
self in "the translating of a text"
agent in "the translator of a text"
result in "the translation of a text".

Notice that each time the same predicate is used, namely translate and each time the same function: object, but the viewpoint has changed.
Note that there is not necessarily a language word for each possible viewpoint in a case frame (e.g. there is no single English word introducing the source of translate).

Here is another example: take the case frame of TRAVEL with cases self, agent, destination, then the viewpoint is

self in "the travelling of John" and "to travel is great fun"
agent in"the traveller arrived earlier".

(ii) When the function of the predicate is to provide more information about an already introduced object or predicate, then the viewpoint is the case slot that is filled by the object or predicate in the case frame of the predicate.

Take first the case where a predicate provides more information about an already introduced entity, i.e. the predicate has the function of a qualifying adjunct, e.g.

(a) the translated text
(b) he translates the text
(c) the text translated by him...

The viewpoint of translate in (a) is result (or source ! there is ambiguity here) because the entity introduced by "text" fills the result case of translate. The viewpoint of translate in (b) is agent because the entity introduced by "he" fills the agent slot. The viewpoint is again the result (or source) case in (c) because the object introduced by "text" fills the result slot of the frame associated with translate.

Now take the other situation, a predicate provides mre information about another predicate, i.e. the predicate has the function of a modifying adjunct, then the viewpoint is the case slot filled by the predicate of the head.

Consider the abstract case frame for SLOW with cases self and patient and for WRITE with cases self, agent and result then in

"slowly written text"

the predicate of WRITE (i.e. the activity of writing itself) fills the patient slot in the case frame of slowly. In other words the viewpoint of slowly is patient.

From the discussion it should be clear that although the notion of viewpoint differs slightly from one functiontype of the words to the other, a viewpoint of a word is always one of the cases of the case frame associated with the predicate of the word. The viewpoint indicates the relation by which the rest of the information is linked to the predicate having the viewpoint and this relation is always a case relation, i.e. a predicate-argument slot relation.

In conclusion, we introduce a rule to relate a viewpoint to a word.

Definition

Let W be the set of words and L the set of case labels, then viewpoint; W → L is a function relating a viewpoint to a word.

For the same predicate and the same function surface case signals (in particular affixes) are often used to indicate a difference in viewpoint.
Consider:

"translator" , predic: translate, viewpont: agent
"translation", predic: translate, viewpoint:result

and

"the translated text", predic: translate, viewpoint:source or result,
"the translating interpreter", predic: translate, viewpoint: agent.

The active-passive distinction is another example where the viewpoint is changing but the predicate remains the same.

Important remark

Just as there is for all predicates a literal and a nonliteral usage,a viewpoint can be used both literally and nonliterally. When you say"the author" then you introduce an entity by saying that it is the agent of a write act, but not necessarily literally at the moment of speaking. In general the viewpoint of an object is more often nonliteral than literal. What interests us are the syntactic repercussions of the viewpoint, literal or not.

Notice that this situation often happens in the grammar. Consider e.g. the gender distinction male/female/neuter, as used in Dutch, German, French,etc. Although there may be a relation between the natural sex, more often this relation is no longer to be taken literally.

On the relation between case frames and semantic processing

As a final part in this introduction to case, we make the link to the semantic interpretation process.

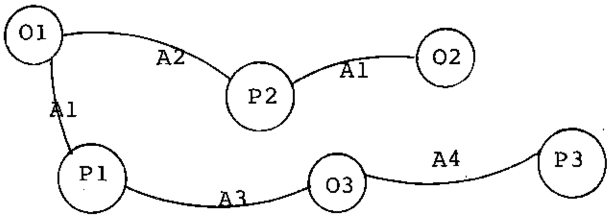
One of the main goals of a natural language communication is the exchange of information. To make this process operational one needs therefore a way to store information. This store is called a data base, a universe of discourse, a memory structure (such as a semantic network e.g.). The notion of case plays an important role in its construction.

Let us describe very roughly how such a memory structure may be organized. Note that we will only deal with information from episodic memory, i.e. the properties of the objects in a particular universe of discourse or the factual knowledge rather than the communication of purely semantic knowledge which is still another problem.

A universe of discourse consists of a set of objects and particular properties (possibly relations) of the objects. Let us assign to each object a unique node and label it for ease of reference. Besides object nodes we must have a way of representing the properties. For this purpose we introduce other nodes and call them property nodes. We label these nodes with a signal indicating what property is contained in the node. The object nodes are brought in contact with the concept nodes by connecting them by lines. As a particular object node has a particular relation to a property, we will label these lines also. The labels are called the case indicators. Finally we bring properties in contact with other properties by connecting their respective nodes by lines and labelling them also.

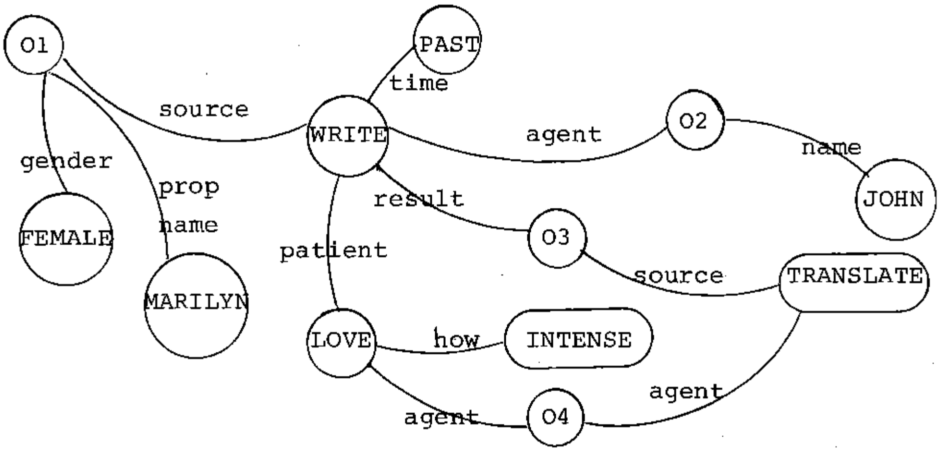
Example 1

Let P1, P2 and P3 be labels for properties, O1, O2, O3 labels for object nodes and A1, A2, A3, A4 the case indicators,then we can construct the following memory structure:



Example 2

Using English like words for the labels of properties one can construct the following example:



Note
(a) Although we use natural language words as labels for the properties, they should in no way be considered as such. Rather one should consider them as expressions in some coneptual language, e.g. as used in conceptual dependency graphs (Schank,1975).

(b) Do not take these memory structures as a representation of the content of a sentence . (In most linguistic systems there is no difference between the memory representation and the representation used to specify what meaning will be conveyed in a particular sentence, a viewpoint which we strongly object).

The extraction of information is guided by various processes, in particular cognitive or other psychological machinery (starting with a stimulus to communicate) pragmatic knowledge such as to whom the message is being addressed, what the speaker is supposed to know about the subject matter, etc. As a consequence the extraction process can only be made operational by embedding it in another task environment such as a question/answering system, where there is a need to communicate particular information.

Roughly such an extraction process might go as follows: "Let us say something about the object node O2, first we decide how to introduce O2, let us do that by means of its proper name, then we decide about the basic topic to be discussed in connection with O2: WRITE. With WRITE several other case slots are connected, we decide to realize the result case. Also we realize the concept PAST. Now we have to choose a way of introducing O3. For this purpose we pick out one of the properties attached to O3 namely TRANSLATE. With 'translate' another case slot is being associated in which the object O4 is located. To introduce O4 we use the concept LOVE. With LOVE we realize the patient case which yields O1. To realize O1 we use its proper name which is Marilyn. The sentence resulting from the whole process might be the following one:'John wrote a text which was translated by someone who loves Marilyn'.

Resulting from other extraction processes many other sentences are possible for the same piece of information, e.g.:

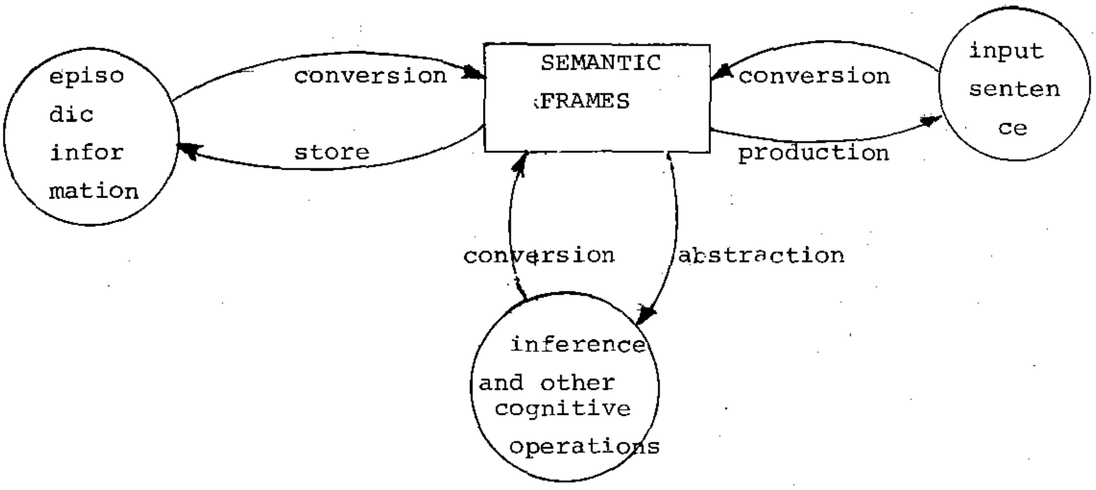
'John wrote about Marilyn'
'The translator of a text written by John loves Marilyn'
'The author of a text about Marilyn is called John'
'Marilyn is being loved by someone', etc;

The association of a case frame with a concept consists of matching processes between a sequence of properties in the memory and a series of properties associated with a predicate. Also the different case relations that occur in the memory are matched against the case relations found in the case frames and the various objects depending on these case relations are associated to their corresponding argument places or case slots in the case frame.

The latter process can be compared to the process of lambda conversion (as it is used in Church's lambda calculus, Church,1941) and in the programming language LISP. Also here one starts from 'abstracted' forms or frames containing a function name and various slots for arguments (the bound variables). The bound variables are then brought into contact with the actual arguments by pairing the values of the actual arguments to the bound variables on the association list.

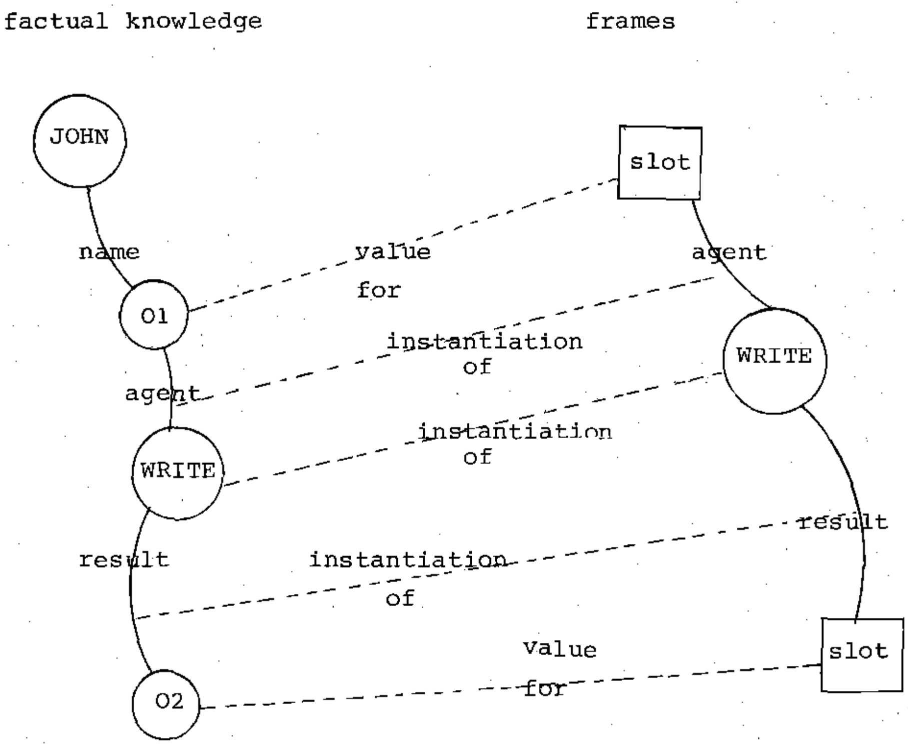
Moreover the analysis process might also be regarded as such a conversion process, so, we obtain a two-way conver tibility of the deep case frames, one way from the memory and another way from the language input. Other tasks such as inference making need the same sort of process, i.e. the information has also to be bound to the abstract case frames in order for these systems to become active. Seen in this way the case frames are really the 'filter' through which all activities pass:

case



Another way to express what happens when the case frames are related to factual knowledge is to consider the memory structures as instantiations of the concepts in the abstract case frames and the main task is then to find frames such that particular information can be regarded as an instantiation.

Schematically:



Although there is a lot more to say about semantic networks and case frames we trust that the reader has at least some idea now about the way in which we see the further usage of case frames and the interrelations with semantic interpretation.

Discussion and further references

Although the notion of case is very old (See Lyons 1968, 289 ff for its use in Latin and Greek grammar theories) its reintroduction into modern grammar theories is normally credited to Fillmore (1968).

Our own view on case has been more influenced by its use in artificial intelligence (cf. Wilks (1977), Simmons (1973),Bruce(1975)) or cognitive psychology (see e.g. Norman and Rummelhart (1975)). The memory model introduced in the text is strongly related to the LNR memory structure (ibid.).

Equivalents to the notion of a case frame as used here is that of case paradigm (Celce Muria, 1972) of formula and paraplate (Wilks, ibid.) and of units in the KRL representation language (Bobrow and Winograd, 1977).

As far as we know the notion of viewpoint as used here is new. (Do not confuse this with the notion of perspective (Fillmore,1977), or topic/focus)

The idea that there are a fixed number of (universal) cases has been proposed by various authors (Fillmore, 1968, see Samlowski(1975) however for an evolution of the cases, Schank(1975)). We do not follow it here.
It will become obvious in the applications later on that we take a very free position as regards the substantial claims about case.

Now we start with a discussion of the rules which use as factors function and case.

1.2.1. Semantic features

We mentioned already that with each slot in a case frame certain semantic properties are associated that the entity which is going to fill the slot is supposed to have. There are two problems in this context:
(i) how do we represent and compare semantic features;
(ii) how do we know what features become active in a certain matching process.

The first question is quickly resolved. We will use the same representation construct as for syntactic features: a feature complex. The matching process is equal as the one for syntactic features and we refer to the formal definition already given. Moreover an inference tree for semantic features can be introduced and used during matching.

The second question is more difficult. It will be treated in two parts: (i) first we define a formalism to associate semantic features with a certain case slot, and (ii) then we discuss how we can find the semantic feature complexes relevant to a certain match process.

We relate the features to a case slot by a rule called value restriction assignment.

Definition

Let SF be the set of semantic feature complexes then value-restriction: AS x P → SF is a function.

We now update our definition of a case frame, such that semantic features can be specified in the same formalism as we earlier defined
This is done by presenting a generative grammar defining abstract case frames. Abstract case frames are case frames to which the value restriction has been added.

Convention

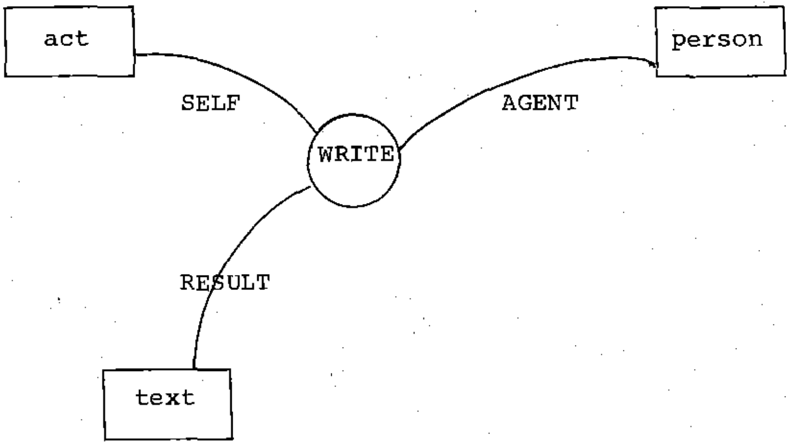
Let G =⟨{⟨abstract-case-frame⟩ ,⟨predicate⟩,⟨sem.feat. complex⟩,
⟨case-list⟩,⟨case-label⟩} , Vt, ⟨abstract-case-frame⟩ , P ⟩
be a context-free grammar with
P :

⟨abstract-case-frame⟩ → (⟨predicate⟩ ⟨case-list⟩)
⟨case-list⟩ → (⟨caselabel⟩ ⟨sem.feat.complex⟩)
⟨case-label⟩ → .... the case labels
⟨sem.feat.complex⟩ → ... the sem.feature complexes
⟨predicate⟩ → .... the predicates.

<u>Example</u>

(WRITE (SELF act) (AGENT person) (RESULT text)) is
an abstract case frame.

We will graphically represent abstract case frames as case frames to which the sem.features have been added:



Remarks:

No claim is here made about there being a universal and definite list of semantic features, nor do we make a claim about a definite and universal list of cases. This all depends on the interpretation of the formal theory. For the same reason no claim is made about the depth or conceptualness of the predicates and whether there should be a limited number of them.

The semantic features test.

Now we investigate in detail how this information about semantic properties can be used in the language system. The following points are relevant in this respect:

(i) How do we know the semantic features that are to be satisfied, and

(ii) how do we know the semantic features associated with the slot filler.

(a) Situation 1: the slot filler is the head and the frame carrier the subordinate.

Example 1 : "the edited translation", where translation fills a slot in the case frame of "edited", and "edited" is the adjunct of "translation".
Example 2: "The slowly written text", where written fills a slot in the case frame of "slowly" and "slowly" is an adjunct of written.

Question 1: How do we know the features to be satisfied ?

Answer: By means of the viewpoint of the case frame carrier.

Recall that for adjuncts the viewpoint denotes the case slot that is to be filled by the entity about which the predicate provides more information, it follows that this entity must have the features associated with this viewpoint.

E.g. Given the frame: (EDIT (SELF act) (SOURCE text) ...) and the words 'the edited translation", then with the viewpoint source, for edited, the semantic features to be satisfied are 'text'.

Question 2: How do we know the features to be associated with the slotfiller ?

Answer:by means of the viewpoint of the slotfiller or by means of the SELF case.

Recall the distinction we made between modifiers and qualifiers. A modifier 'modifies' the predicate, used for whatever purpose, itself. Whereas a qualifier provides more information about the entity denoted by a predicate.

Notice that adjuncts which are the subordinate of other adjuncts always modify the latter.

So, if the slot filler is itself an adjunct, things are easy, the semantic features of the slot filler are those which are associated with the self-case in the case frame of the predicate.

Consider: "slowly written ...", "slowly" modifies the activity of writing itself. We could call the viewpoint of slowly patient, then the features of the self-case of write must match with the features of the patient case of slowly.

If the slotfiller itself is an object, we have to take the modifier/qualifier distinction into account:

(a) Qualifying adjuncts
In this situation the adjunct provides more information about the entity introduced by the object. Bus as we specified already, the entity denoted by the predicate of the object fills the case called the viewpoint ! Hence the semantic features of the slot filler are the semantic features associated with the viewpoint of this slot filler.

E.g. "edited translation" with (TRANSLATE (self act) (result text)...) and viewpoint of translate result, then the semantic features of the slotfiller are text.

(b) Modifying adjuncts.
But if the adjunct modifies the predicate used to introduce the entity, then obviously the self case again leads us to the semantic feature complex of the slot filler just as for adjuncts.

Consider e.g. "slow writer", where "slow" can be modifying as well as qualifying (his writing goes slowly - he is a writer and he is slow). If modifying the activity of writing is the argument filling the patient slot of slowly if qualifying the person itself is the argument filling the patient case of slowly.

(2) Situation 2: The slot filler is the subordinate and the frame carrier is the head.

Example: "He translated a text", where text is functionally an object of translated and at the same time it fills a slot in the frame of translate.

Question 1: How do we know the features to be satisfied ?

Now the answer is not so straightforward, the language understander has to find out himself what case the slot filler is filling. He does this largely on the basis of surface phenomena to be discussed in next section. For the time being let us assume that we know what case the object is filling, then it is obvious that the semantic features to be satisfied are those that are assigned to this case.

Question 2: How do we know the features of the slot filler ?

No complication arises here. We compute the semantic features of the object via the viewpoint of the object and the features that are assigned to this viewpoint in the case frame of the predicate associated with the object.

Final remark:

Note that the semantic features of a word are NOT stored directly with the words of a language in the lexicon (as is usually the case) but computed in an active way from the case frames. The advantage of this method should be obvious.

Discussion and further references

In the first generation of recent linguistic theories and AI systems which made use of semantic fea tures the role of these features was located after syntactic processing, i.e. right before the process of semantic interpretation (and some even thought that this was the semantic interpretation process itself (Katz, 1973)).
In the second generation of systems (so called semantics directed parsers) semantic features are applied immediately in connection to the input itself (cf.Wilks(1977),Riesbeck(1975))

We believe to have made some improvements about how that should be done. The main improvement is the notion of viewpoint which enables us to treat several generalizations not captured by semantics directed systems, such as the usage of the same frame for different surface frames (active/ passive, nominalization, adjective forms). In the syntax based systems this generalization is obtained by transforming all these surface forms into one format that can then be matched with one single deep pattern.
We do not need to do that because we actively compute the features from the same abstract case frame without changing the structures of the representation.

A second improvement is the usage of a global inference tree over the whole system and of feature complexes instead of simple features.

Just as Wilks(1977) we would like to allow case frames as value restriction and we will build this into the system as soon as possible.

1.2.2. Order

It was mentioned in a previous paragraph that it is necessary for the language user to find out exactly what case a slot filler fills on the basis of surface case signals if the slot filler is the subordinate of the frame carrier. These signals are:

(i) a priori restriction
(ii) order
(iii) surface case affixes and prepositions.

We will introduce a new representation construct called a surface case frame or semantic network in which information about (i) , (ii) and (iii) can be expressed. It will turn out that viewpoint and function are the major decision factors in the process of computing the surface case frame of a given abstract case frame.

(1) A priori restriction

Not neeessarily every case slot that occurs in the case frame is a candidate for being filled in a given situation. In particular there will never be an object filling the case of the viewpoint of the predicate. But other cases may be missing as well.
Consider:

"The hammer broke the window" (the agent case is missing).
This restriction is function and viewpoint dependent because if we take the same function but change the viewpoint - from instrument to patient we can express the agent case:

"The window was broken by John".

We conclude that the first thing which is to be specified in a surface case frame is what cases are allowed.

(2) Order

Although the order of the cases in an abstract case frame is considered to be irrelevant, the order in a surface case frame is indeed relevant.
Consider e.g.

"He gives John the book"

and not

" He gives the book John".

Note that this is a similar situation to one already discussed, namely the phenomenon that the occurrence of one subordinate may restrict the linking of other subordinates. Here the occurrence of one case influences the structural property of the predicate to such an extent that only certain other cases are allowed or conversely that other cases should occur.

Let us now decide on a representation construct expressing order and a priori restriction. Let us use for this purpose completion automata already introduced earlier.
Although we will now use the system in a different context, the formal concept remains the same.

Recall that a completion automaton is a 5-tuple $CA = \langle V, Q, A, \delta, F \rangle$ with V the alphabet, Q a set of states, A the initial state assignment function, δ the transition function and F the set of final states.

In this application we interpret the alphabet not as grammatical functions (as done earlier) but as cases. Initially when no cases have been processed, the initial state (defined by the initial state assignment function) will be associated with the predicate. Whenever we fill a new case slot, a new state (or more than one new states) is associated with the predicate. If we want to see whether an object fills a slot in the frame, it will not be sufficient to check whether the semantic features match, in addition the appropriate state should be associated at that moment with the word. Moreover at the end, i.e. when no more objects occur, there should be a final state linked with the predicate.

order

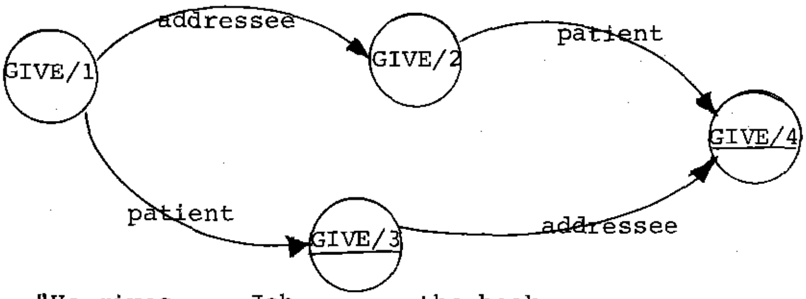
Examples

abstract frame:

(GIVE (self act) (agent person) (patient thing) (addressee person))

Some surface case frames : (we underline the final states)

with viewpoint agent and function adjunct:



```
E.G.: "He gives     John        the book
                     ||            ||
                  addressee     patient
                     ||            ||
        give/1 ──────────▶ give/2 ─────▶give/4
```

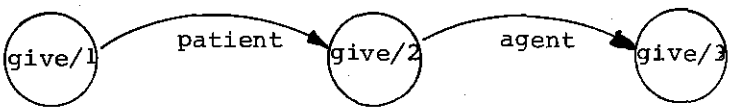
Note that for "he gives John" with John the addressee,
no final state is reached.

```
E.g. : "He gives     the book       to John"
                        ||             ||
                     patient        addressee
                        ||             ||
           give/1 ──────────▶ give/3 ──────────▶give/4
```

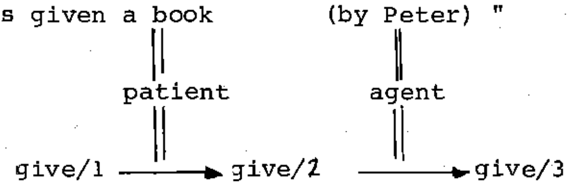
Note that "He gives the book" would equally well be accepted.

order

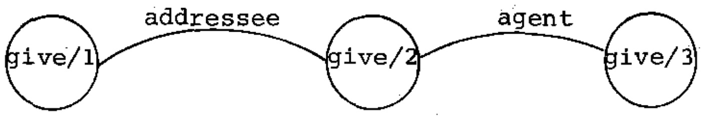
With viewpoint addressee and with function adjunct:



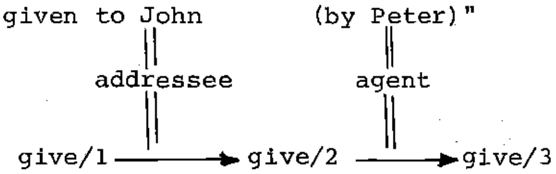
E.g.: "John was given a book (by Peter) "



With viewpoint patient and with function adjunct:



E.g.: A book was given to John (by Peter)"



1.2.3. Government

The next phenomenon in relation to case is that of surface case signals.
A surface case signal is a syntactic feature that is associated with an object which is a candidate for filling a slot in the frame.
There are 2 types of si als:

- features which are associated already via morphological processes to the object, e.g. genitive, objective, dative, etc;
- prepositions which are subordinates of the object with the function case sign. We can treat the latter as being equal to the former by means of the earlier introduced send-through rule: the preposition sends a signal, usually we will take for this the name of the preposition itself, to the feature complex of its head. In other languages the preposition can be said to 'cut out' a subset of the feature complex of the object. In any case the surface case signals are syntactic features and they are resident in the syntactic feature complex of the object; indeed they should be because the case features may play a role in the concord phenomenon.

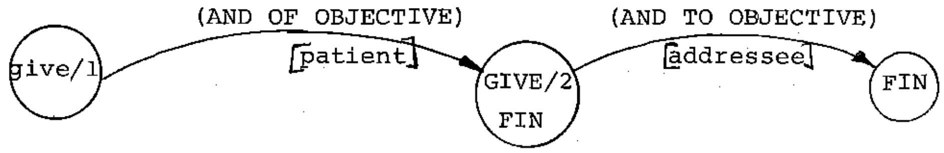
Again the question raises whether there is only one type of syntactic feature complex for each slot of the frame or whether there are more, and so, depending on what factors. The answer is that there are more and in particular that there is a feature complex for each case depending on the viewpoint AND on the function AND on the path in the case network associated with the predicate. So the condition of a transition in the surface case networks introduced earlier is not a case but a feature complex.

In the case network it is indicated what particular case signal should be present. So in the analysis process we will compute (on the basis of the viewpoint and the function) what the surface case frame is of a given predicate. Then we will try to make transitions for the objects on the basis of the surface case signals. If a transition can be made we know immmediately what case this object is filling (and we can start computing the semantic features).

We can use the matching process defined earlier to see whether the signals are present in the extension of the feature complex of the object. Note again the importance of the relevance logic underlying the matching process.

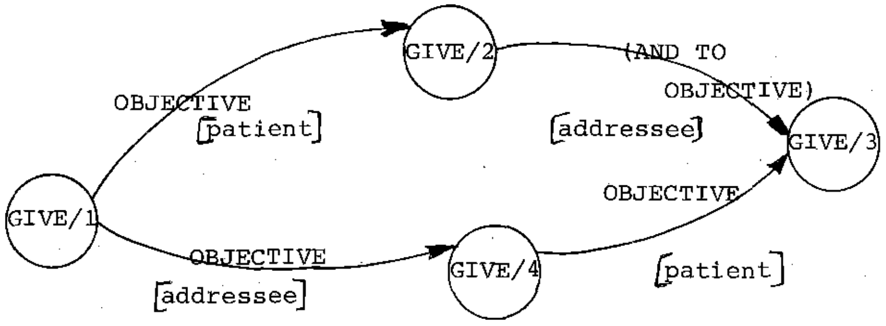
Some examples:

viewpoint: agent
function: object



example: the giver of a book (to John)

viewpoint: agent
function: adjunct



One may wonder that such a detailed information will soon lead to extraordinary databases all filled with surface case networks. But this need not be so, if we assume that there is a limited, finite number of 'conceptual' predicates and that many different words which have the same implications as regards case frames (that means the same surface case signals, the same order restrictions AND the same value restriction determining the semantic features) then we need only one abstract and surface case frame for a whole class of words.

Discussion and further references

Several investigators currently working on semantics directed parsers are trying to apply some sort of network formalism to regulate the order(Wilks, personal communication). It turns out that the completion automata introduced earlier for order restrictions of subordinates constitute a very interesting and powerful solution. Mainly for the following reasons:

(i) A completion network is called by an input element whereas in normal network systems you go from the network to the input via 'nonterminals' which call each other.

(ii) In a completion automaton the networks are 'local' in the sense that each network takes care of its relevant surroundings without bothering about other networks running parallel to it.

(iii) The condition for a transition has nothing to do with categorial information but with surface case signal tests.

(iv) In fact the networks here are transducers because they process a sequence and yield as output the cases.

government

Another major improvement is the following that instead of transforming the surface structure representations we compute actively the surface consequences of a given case frame on the basis of viewpoint and function.
In this way we are able to relate case frames to the surface format directly.

SUMMARY AND EXTENSIONS

In the preceding paragraphas we have introduced a modular grammar as a formalism to express linguistic knowledge. The most peculiar feature of this grammar is the modularity of the description: each phenomenon is investigated on its own and is assigned a special rule and it is not at all clear (i.e. determined by the grammar) how the rules interact to produce or analyse a complete natural language sentence.

This is in contrast to most available models where all the phenomena are incorporated in an integrated description. The reader will have noticed that this attitude change is leading to a fundamental re-thinking of the properties of natural language.

We have investigated two important factors: grammatical function and case. In relation to these factors we dealt with the following phenomena:

(i) The relations environment

We have seen two situations where a certain grammatical relation can only occur if other grammatical relations are present:
1.1. The first situation is that the head of a relation should itself have a certain relation for the relation to hold
The two rules introduced in this context are:
FUNCTION -OF-HEAD specifying explicitly for adjuncts and functionwords what the function of their head should be
TAKING-OBJECTS specifying whether a certain word with a certain function may have a word with the function object as its subordinate.
1.2. The second situation is that the subordinate should itself be the head of another relation. This is regulated by the syntactic networks (cf. infra).

(ii) The ordering

The next phenomenon is the role of order made possible by the time dimension of language. There are two aspects here: ordering of the head and the subordinate and internal ordering of the subordinates of the same head.

2.1. Ordering of head and subordinate
Again we needed two rules: one for adjuncts and functionwords and one for objects:

POSITION specifies where a word having the function adjunct or functionword stands as regards its head;

OBJECT-POSITION specifies where the objects of a given word come.

2.2. Internal order of the subordinates
Again we need two rules one for adjuncts and functionwords and one for objects:

SYNTACTIC NETWORKS associates with each function a transition network of a completion automaton, where each subordinate will induce a transition in the network and thus restrict the possible subordinates left.

CASE NETWORKS: associates with each function for each viewpoint of a predicate a transition network. Each case induces a transition in the network and thus restricts the cases left.

(iii) Features
We introduced a representation construct for representing complexes of features that showed to be of great use in the language system. It can be used as well for processing syntactic as semantic features.

3.1. Synta ctic features
Syntactic features are associated directly with the natural language word or result from the SEND-THROUGH operation which dynamically changes the feature complex of a head.
The following rules make use of them:

CONCORD specifies whether the features of a subordinate should match with those of the head of a relation

GOVERNMENT:to make a transition in a case network a sequence of features should match with those of an object ready to fill the case slot.

3.2. Semantic features
Semantic features result from active computation on the basis of case networks. Th .r use is based on the assumption that to fill a slot in a case frame, a value restirction must be satisfied. Two rules are necessary here:

SEM-FEAT-ADJUNCTS: specifies whether the head should fill a case slot in the frame of the subordinate and so if this is by a modifying, or qualifying relations.

SEM-FEAT-OBJECTS: specifies that the object filling the slot should satisfy the value restriction of the case.

EXTENSIONS

It is obvious that the list of rules given here is far from complete and more research is needed before all linguistic phenomena will be covered. We will now very briefly indicate in what directions the current research is going. This will give the reader an idea about the extendibility of modular grammars:

(i) The problem of sentence structure

At the moment the grammar itself does not deal explicitly with the structure of a whole sentence.
What is clearly needed here is some superimposed control structure for sentences which evolves in parallel with the rest.
In order to represent sentence structures such that they can be consulted easily during parsing and producing we are thinking about a new set of networks, this time called sentence networks. The sentence networks come into action right from the beginning of the input , and the condition for a transition is the presence of a particular function. The idea is that when going through a sentence you also go through a network and when a certain path has been successful, a certain type of sentence (affirmative, imperative, question,etc;) is recognized.

Similarly for language production, you organize the elements of the sentence in the format of such a path.

(ii) Interconnection of sentences

This brings us to a second problem namely intersentential relationships realized by relative pronouns or conjunction words. It seems that such words depart from the axiom of functional structures that one word can be the subordinate in only one other structure, because they play a role in both sentences. Thus in the sentence 'he left when she came in','when' would be the subordinate of a relation to 'come' but also of a relation to 'left'. The implications of this viewpoint should be seriously considered. In particular it would no longer be possible to consider functional structures as trees and some other aspects (especially for the parsing process) should be reworked.

(iii) Coordination

Another aspect on which we are working at the moment is coordination. It is hoped that due to (i) the fact that our representation level is that of functions and (ii) the modular character of the grammar, a powerful start position for the investigation and processing of coordination will be found. Rather than introducing extra extensions of the existing grammar rules, we are looking for some general principles that underly coordination.

There are still other factors and syntactic phenomena that will deserve attention. The point is however that a modular grammar is per definitionem extendible with whatever sort of rules that may turn out to be necessary.

1.3. The structure of the lexicon

When discussing the rules of the grammar it could be noticed that for several rules we need information that is uniquely associated with the words of the language. In this section we investigate what information exactly is to be associated with the words. This association is considered to be an explicit assignment, i.e. wo do not deal with morphological processes that would enable us to economise on the explicit information.

Because the same word form can have many different functions or meanings, it should be logically possible to assign more than one information sequence to the same word.

(i) function.

The first item in an information sequence is a subfunction. If there is more than one subfunction and the rest of the information is exactly the same, we will allow there to be a list of possible functions instead of just one.

(ii) predicate

The second item is the name of the predicate denoted by the word. This predicate should be seen as 'conceptual' as possible, because it will be the key to the abstract case frame relevant for the word.

(iii) subpredicate (or concrete predicate)

In addition to the predicate we assign a subpredicate which can restrict the general concept stated in the predicate to a narrower application. We need this subpredicate because otherwise semantic information is lost. At this moment the subpredicate is optional. We therefore often define it to be NIL.

(iv) viewpoint

The next item is the viewpoint of the predicate in the case frame associated with the predicate. From the discussion of the grammatical rules which involve the notion of case, it must be clear that there is a viewpoint for each word except for those having as subfunction some kind of functionword (but these words have no predicate either).

(v) syntactic features

In contrast to the semantic features which are computed from the case frames, the syntactic features are immediately assigned to each word for obvious reasons. As we explained earlier, for adjuncts there may be two feature complexes: the external and internal feature complex. These two feature complexes are then brought together in a list and thus associated with the word.

(vi) send-through feature

Finally we need a specification of what kind of feature complex is sent to the head if indicated so by the 'send-through'rule.

This brings us to six information items in a sequence. We summarize this in the following definition.

Definition

lexicon: W → (I) is a function relating words with sets of information sequences where an information sequence I =⟨a1,a2,a3,a4,a5,a6⟩ with

a1 a function or a list of functions
a2 a predicate
a3 a subpredicate
a4 a viewpoint
a5 a syntactic feature complex
a6 the send-through feature.

Example

for 'father':

```
((nom:object   fam.relation   male-parent   self  (AND MALE SING)  NIL))
      ||             ||             ||        ||         ||            ||
   function      predicate     subpredic.   viewp.   synt.feat    send-through
```

1.4. SEMANTIC STRUCTURING

Although our investigations have not yet reached the level of semantics as such we will deal in this section with some topics situated on the borderline between syntax and semantics. In particular we will present a representation construct that should serve as the basis for semantic interpretation. Later we will show how this construct can be computed from a natural language sentence and how it can be translated back into natural language.

Many important and interesting problems will remain outside the scope of the present discussion. What we present here is again the essential ground work: How function and case relate to the structures we will present. First we introduce our viewpoint on semantics which will of course be relevant before we start with the treatment of the representation constructs themselves.

1.4.1. Introduction to semantics

The whole area of semantics is somewhat unclear at the moment and it is is therefore not wholly unnecessary to formulate an overview of the field as we see it.

(a) The representational viewpoint.

The first "school" of thinking about semantics assumes that the final result of a semantic investigation should be the definition of semantic structures in which the meaning of a piece of language is represented in a nonambiguous and fully explicit way. The task of a semantic theory consists then in the definition of a formal language in which semantic structures can be specified. To be meaningful it should also be made explicit how the formal language relates to natural language sentences. Moreover the formal language itself should be defined completely: not only the syntax of the expressions but also the (so called formal) semantics, that is how the semantic structures themselves are to be interpreted.

Let us call this conception of semantics representational semantics. It has been the main interest of linguists (cf. generative semantics) and logicians (cf. predicate calculus modal logic, etc.).Formal semantics is the specialty of logicians and Frege's method of interpretation is an obvious example of their results.

One could say that the intuitive basis for representational semantics is the idea that a meaning structure is the end-product of language understanding, cf. Searle (1976,49): 'understanding a sentence is knowing its meaning'.

(b) The procedural viewpoint

The second more recent "school" of thinking about semantics claims that the final result of a semantic investigation should be the execution of processes. This is based on the idea that meaning is not a representational structure but a process (that uses representational structures as byproduct). The basic processes during interpretation are about the storing and retrieval of facts, the planning and execution of commands, problem solving in order to resolve inferential problems or answer input questions, etc.

Let us call this kind of semantics <u>procedural semantics</u> . It is the specialty of the computational linguists. Just as for syntax computational linguists started with applying existing linguistic models before they turned to a development of their own syntactic theories, the first attempts within procedural semantics consisted in the application of (basically logical) theories of representational semantics. It seems that at the moment important developments are going on in the procedural semantics world. For one think the theory of programming language semantics is currently reaching a state where important results are coming out, for another thing, it becomes more and more clear that fundamental problems of semantics will only find a satisfactory solution within a 'process' environment.

Thus e.g. the formal semantics methods used in logic (i.e. hierarachical control structure from bottom to top) are being replaced by more flexible control structures, where results of the evaluation of a part are spread over the other parts of the structure. Thus also another conception of the representation of the language input itself emerges: instead of being the representation of the meaning the representations are now seen as the control structure of the process of semantic evaluation.

This final point will be of particular importance for the rest of our investigation. The structures we are proposing are seen as useful information for the semantic evaluation but they are by no means the only information necessary (think about episodic information resulting from previous text or world knowledge). Moreover the actual meaning of the words, which is a program stating how the evaluation goes, is called on the basis of the information structure rather than that the information structure itself contains already the meanings.

It was not the aim of this thesis to put forward results on the level of semantics. It will therefore not be possible to discuss these controversial issues in any level of detail. What we will do here is define structures which contain every information that the grammar can offer to the semantic evaluation process.

We call such structures SR-constructs and the whole set of possible structures, or the language of SR-constructs, the SR-language or SRL.

Although we will give a provisional formal semantics for SRL (provisional because it still follows Frege's method of interpretation), the issue of effective interpretation will not be dealt with here (although work in this connection is already going on at the moment in our computational linguistics laboratory, in particular work about memory representations.) Let us now give a definition of SRL.

1.4.2. The definition of SRL

The semantic representation language we will define in this section consists of (recursive) trees. It is tailored to logical representation languages such as the predicate calculus or extensions of it. The use of trees instead of linear symbolic expressions is justified by the internal complexity of the constructs which are easier processed by humans as well as computers if the internal structure is apparent from the formal outlook. For didactic purposes, we gradually introduce the components of the structures until we have the full power of the language. For the definition of the syntax of SRL we will use a context-free grammar. A complete definition of the language is given at the end of this section.

(1) Predicates and their arguments.

Let us call the objects in the semantic representation language semantic representation constructs or briefly SR-constructs.

-i-

The first notion of importance is that of a variable familiar from logic or mathematics. In this context a variable will mean two things: (a) on the level of syntax of SRL the variable will be the topnode of an SR-construct such that it can serve in another (or the same)tree to call the SR-construct again (in other words we allow recursive trees). (b) on the level of a semantic interpretation, a variable is a place address which receives the values of evaluating (i.e. interpreting) the SR-construct.

The second notion of importance is that of a predicate. A predicate is the name of a function or a relation in the logical sense. Predicates can after interpretation have as value an entity, a class of entities, a list of entities, a truthvalue, etc.

We formalize this in the following rules (the nonterminal ⟨pred-constr⟩ is an auxiliary symbol that will simplify the grammar as will become obvious soon.)

```
1. ⟨ SR-construct ⟩ → (   ⟨ var ⟩   ⟨ pred-constr ⟩   )
2. ⟨ var ⟩  →  X1,X2,X3 ... names of  variables
3.⟨ pred-construct ⟩  →   (PRED  ⟨ pred ⟩   )
4.  ⟨ pred ⟩  →   AND, FATHER, ... names of predicates.
```

-ii-

Some predicates may take arguments in the usual logical sense. If this is the case we add them to the SR-construct with an explicit label for the argument slot and a variable referring to another SR-construct in which the semantics of the variable are specified. The label for the argument slot is in linguistic theory called the case label. It denotes the particular relation of an argument to its predicate. In order to incorporate arguments we extend the grammar as follows:

Rule 3 becomes

```
2.  ⟨ pred-construct ⟩→(PRED   ⟨ pred ⟩  )
                       [(ARGS   ( ⟨ case-label⟩ ⟨ var ⟩  )+  )]
```

and

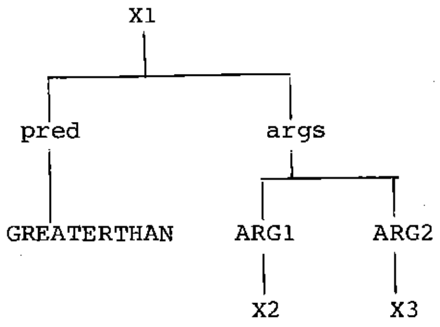
```
5. ⟨ case-label ⟩  →   agent,patient, ...     case labels
```

Example:

1. ⟨SR-construct⟩ $\overset{1,2,3,4,5,5,2,2}{\Longrightarrow}$

```
(X1 (PRED  GREATERTHAN)
    (ARGS  (ARG1 X2)
           (ARG2 X3) ) )
```

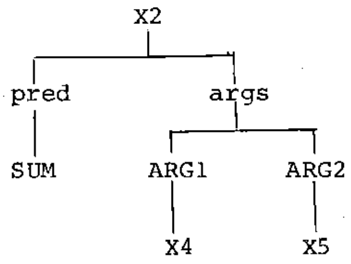
or as a tree (according to our standard conventions):



$\langle$SR-construct$\rangle \xRightarrow{1,2,3,4,5,5,2,2}$

```
(X2   (PRED   SUM)
      (ARGS  (ARG1  X4)
             (ARG2  X5) ) )
```

or as a tree

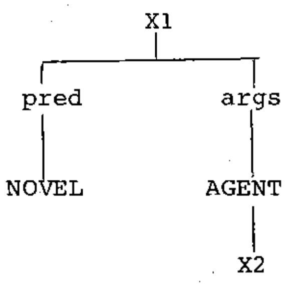


etc;

2.
$\langle$SR-construct$\rangle \xRightarrow{1,2,3,4,5,2}$

```
(X1 (PRED  NOVEL)
    (ARGS  (AGENT X2)))
```

or

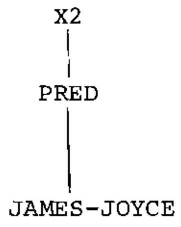


and

$\langle$Sr-construct$\rangle \xRightarrow{1,2,3,4}$

(X2 (PRED JAMES-JOYCE))

or



Semantics

The semantic rule associated with the syntax so far is called predicate application, it can be stated as follows: The value of the variable on top of the construct is obtained by first evaluating the variables of the arguments and by then applying the predicate to these resulting values.

Example

For example 2 to know the value of X1, we first evaluate X2. This yields us a pointer to the entity named James-Joyce, then we apply this result to the predicate NOVEL and obtain a pointer (or a set) to the entities defined as the novels of Joyce.

(2) Elaborating the basic structure

(a) Viewpoint

It should be well known by now that the notion of viewpoint is a fundamental aspect of our thinking about language. It is a way to treat many of the relationships between surface case frames of the same abstract case frame and an alternative to the transformational treatment. Due to its importance we will therefore incorporate viewpoints in the semantic structures themselves.

If the structure is introducing an entity, the viewpoint will indicate what case slot the entity is filling in the case frame of the predicate, i.e. in what way the entity is related to the information contained in the predicate. If the structure is introducing more information about an already introduced entity, the viewpoint will indicate the relation to the rest of the information in particular via which concepts the predicate is brought into the expression.

To incorporate this aspect in the grammar, we change rule 3. as follows:

3. ⟨pred-constr⟩ → (PRED ⟨caselabel⟩ ⟨pred⟩)
[(ARGS (⟨caselabel⟩ ⟨var⟩)$^{+}$)]

(b) Concrete predicate

It may be of interest to divide the predicate itself into two parts: the abstract predicate, which is the call name of the abstract case frame used to externalize the predicate, and the concrete predicate, which is the call name of the semantic procedures of the predicate, i.e. a pointer to the "meaning" of the predicate. Because we are not yet involved in effective interpretation, this concrete predicate is sometimes NIL.

This yields another extension of the grammar for rule 3:

3. ⟨pred-constr⟩ → (PRED ⟨viewpoint⟩ ⟨pred⟩ ⟨pred⟩)
[(ARGS (⟨case-label⟩ ⟨var⟩)$^{+}$)]

The just mentioned extensions have no implications directly for the formal semantics rule stated before , but the following extension has, although we do not see very clear in the situation at the moment.

(c) Determinators

There are many problems of semantic representation having to do with effects on the usual evaluation processes caused by determiners and related words: Are the predicates to be interpreted extensional (i.e. with reference to the universe of discourse) or intensional ? Should the number of entities be further restricted bo an arbitrary element of the set defined by the predicate, only one of them, to the whole class collectively or individually, etc. This kind of determination is a well known problem area of semantics and the reader should not expect us to find solutions here. Instead we put all determinators in a sort of garbage can and hang it under the label DETERMINATION. By doing so we can go on with our investigations without needing to resolve all the problems involved. For the same reason we will be silent about the formal semantics of determination. Let us just assume that it involves indicators which play a role in the evaluation. It is hoped that later developments will bring more clarity in the issue.

We extend the grammar then as follows:

```
   rule 3 becomes:
    3.  <pred -construct> ⟶ (PRED  <viewpoint> <pred> [<pred>] )
                            [(DETERMINATION  <feature>⁺ )]
                            [( ARGS  <caselabel> <var> )⁺ )]

    6.  <feature> ⟶ distrib, ...      features
```

(In practice we will allow feature complexes instead of simple features).

(3) Combination of predicates

It is possible to relate in two important ways one predicate to a particular SR-construct:

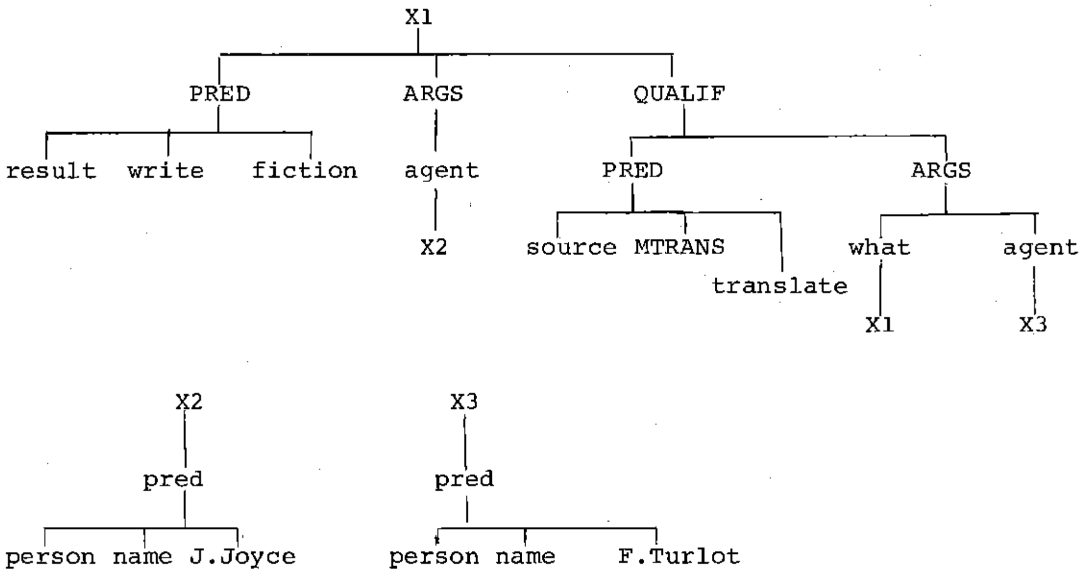
(a) Qualifying: The predicate may introduce a new property of the entity introduced by the main predicate in the construct. E.g. in the sentence 'he had a French gardner', we introduce an entity by the predicate 'gardner' and then we relate this entry with the property 'being from France'. In a predicate calculus notation one combines the two predicates via a conjunction;

e.g. given P1 and P2 then it is said that P1(x) $\wedge$ P2(x).

(b) Modifying: Second it is possible to modify the other predicate itself (without direct consideration of the entity). E.g. in "the early riser woke up late", "early" modifies the "rising" and is not given as a property of the entity introduced by the predicate riser. In a predicate calculus notation one represents this as composition of predicates: Given predicates P1 and P2, then in P2(P1 (x)), P2 'modifies' P1.

Now if we want to incorporate the distinction in the semantic structures, we will need two different rules, one incorporating qualifiers and one incorporating modifiers. But there is a small problem here. Sometimes the syntactic information alone is not enough to make the distinction properly. Hence we add a third type of structure where it is undetermined.

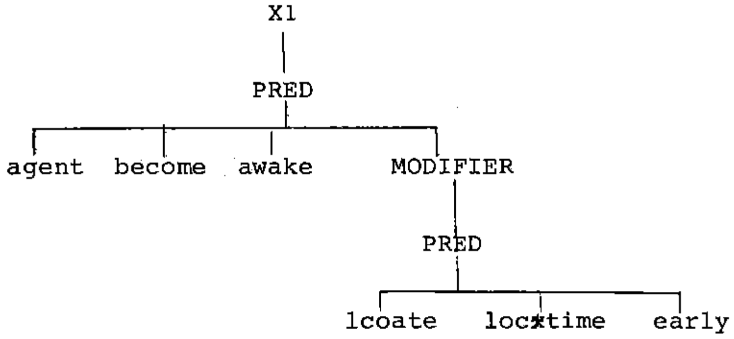
We will attach qualifiers in an SR-construct by hanging each of them on the top level variable with the label qualifier. A qualifier can be another pred-construct. E.g. for the phrase 'a novel by James Joyce translated by François Turlot' we have



(Notice how the label at the top of the SR-construct is used to introduce the entity in the frame of the qualifier).

Now for modifiers, we incorporate them in an SR-construct directly under the predicate node and the viewpoint here will have as slot filler the predicate itself.

Example: "early riser"



i.e. the'becoming awake' fills the slot 'locate' in the predicate loc* time (= locate in time).

Finally if it is undetermined whether a predicate is modifying or qualifying, we will hang the structure under the topnode of the SR-construct with the label UNDET.

We extend rule 3 of the grammar to deal with all these aspects as follows:

```
3.   ⟨pred-construct⟩ ──► (PRED  ⟨viewp⟩ ⟨pred⟩ [⟨pred⟩]
                                 [(MODIFIER  ⟨pred-construct⟩)+ ])
                     [(DETERMINATION  ⟨feature⟩   )]
                     [(ARGS     ( ⟨case-label⟩ ⟨var⟩)+)]
                     [(QUALIF  ⟨pred-construct⟩ )+]
                     [(UNDET  ⟨pred-construct⟩  )+]
```

Semantics

The semantic rule associated with the extensions just provided goes as follows:
To evaluate a predicate with a modifier node, first evaluate the arguments of the topnode and the arguments of the modifier then apply the result to the combination of the modifying and the main predicate.

To evaluate a qualifier, evaluate the predicate construct hanging under the qualifier node.

Complete syntax of SRL

```
1.  ⟨SR-construct⟩ → ( ⟨var⟩⟨pred-construct⟩  )
2.   ⟨var⟩   →  X1,X2, ... names of variables
3.   ⟨pred-construct⟩ → (PRED  ⟨viewp⟩ ⟨pred⟩ ⟨pred⟩
                             [(MODIFIER  ⟨pred-construct⟩ )+])
                         [(DETERMINATION  ⟨feature⟩+)]
                         [(ARGS   (⟨case-label⟩ ⟨var⟩  )+ )]
                         [(QUALIF ⟨pred-constr⟩ )+]
                         [(UNDET   ⟨pred-constr⟩ )+]
4.  ⟨pred⟩ →AND, FATHER, ...    names of predicates
5.  ⟨case-label⟩ →agent, patient, ...  caselabel
6.  ⟨feature⟩ → distri, ...     features
```

On the relation between SRL and natural language

In next chapter we discuss a system that will enable us to relate natural language sentences to SR-constructs. We here discuss very briefly the principles on which this relation will be based.

In the foregoing we discussed two important factors of language: function and case. When introducing the notion of factor we mentioned that a factor has a double role, on the one hand it induces a number of surface phenomena; on the other hand it has an impact on the semantic processes. This impact is such that with each grammatical function there corresponds a particular process of structure building. It follows that the linguistic knowledge necessary to construct structures from functions is essentially procedural knowledge. We will see clearcut examples of this in next chapter.

The second basis for the construction process is the application of a number of so called 'optimalization rules', i.e. rules which expand the bare structures by decomposing the predicates by spreading local information over the whole structure, etc.

E.g. For 'French wine', 'French' will be decomposed in a predicate (e.g.'out-of') and an entity node introducing 'France'. Or for 'he hammered nails into wood' we expand 'hammer' with a caseslot for the instrument and a new entity node introducing the entity 'hammer'.

Again more information about this will be provided in the next chapter when we come to a detailed discussion of the parsing process.

Discussion and further references

There is an enormous literature with examples and discussions of semantic representation languages and it would lead us too far to review it here.

The procedural viewpoint is as the moment not yet very widespread in linguistics. The term procedural semantics is due to Woods (1965). A very strong example is provided by Winograd (1972). For an example of the approach followed in the theory of programming language semantics, the formal basis for the procedural viewpoint, see Milney and Strachey (1976).

A typical semantic representation language from a procedural viewpoint was designed by the Philips research team (see Landsbergen (1976) and Scha(1976)). For further references about the process of constructing semantic structures see the notes after its detailed discussion in next chapter.