8. Language hierarchies and complexity

8.1 Formalism of PS-grammar

8.1.1 Original definition

Published in 1936 by the American logician E. Post as *rewrite* or *Post production systems*, it originated in recursion theory and is closely related to automata theory.

8.1.2 First application to natural language

Post's rewrite systems were first applied to natural language by N. Chomsky 1957 under the name of *phrase structure grammar*.

8.1.3 Algebraic definition of PS-Grammar

A PS-grammar is a quadruple $<$ V, V_T, S, P $>$ such that

1. V is a finite set of signs,
2. V_T is a proper subset of V, called *terminal symbols*,
3. S is a sign in V minus V_T, called *start symbol* , and
4. P is a set of rewrite rules of the form $\alpha \rightarrow \beta$, where α is an element of V^+ and β an element of V^*.

8.1.4 Restrictions of PS-rule schemata

0. Unrestricted PS-rules:

   The left hand side and the right hand side of a type 0 rule each consist of arbitrary sequences of terminal and nonterminal symbols.

1. Context-sensitive PS-rules:

   The left hand side and the right hand side of a type 1 rule each consist of arbitrary sequences of terminal and nonterminal symbols whereby the right hand side must be at least as long as the left hand side.

   Example: A B C $\rightarrow$ A D E C

2. Context-free PS-rules:

   The left hand side of a type 2 rule consists of exactly one variable. The right hand side of the rule consists of a sequence from V^+.

   Examples: A $\rightarrow$ BC, A $\rightarrow$ bBCc, etc.

3. Regular PS-rules:

   The left hand side of a type 3 rule consists of exactly one variable. The right hand side consists of exactly one terminal symbol and at most one variable.

   Examples: A $\rightarrow$ b, A $\rightarrow$ bC.

8.2 Language classes and computational complexity

8.2.1 Different restrictions on a generative rule schema result in

- different *types of grammar* which have
- different *degrees of generative capacity* and generate
- different *language classes* which in turn exhibit
- different *degrees of computational complexity*.

8.2.2 Basic degrees of complexity

1. *Linear complexity*
   n, 2n, 3n, etc.
2. *Polynomial complexity*
   n^2, n^3, n^4, etc.
3. *Exponential complexity*
   2^n, 3^n, 4^n, etc.
4. *Undecidable*
   $n \cdot \infty$

8.2.3 Polynomial vs. exponential complexity (M.R.Garey & D.S. Johnson 1979)

	problem size n		
time complexity	10	50	100
n^3	.001 seconds	.125 seconds	1.0 seconds
2^n	.001 seconds	35.7 years	10^{15} centuries

8.2.4 Application to natural language

The Limas corpus comprises a total of 71 148 sentences. Of these, there are exactly 50 which consist of 100 word forms or more, whereby the longest sentence in the whole corpus consists of 165 words.

8.2.5 PS-grammar hierarchy of formal languages (Chomsky hierarchy)

rule restrictions	types of PS-grammar	language classes	degree of complexity
type 3	regular PSG	regular languages	linear
type 2	context-free PSG	context-free languages	polynominal
type 1	context-sensitive PSG	context-sensitive lang.	exponential
type 0	unrestricted PSG	rec. enum. languages	undecidable

8.3 Generative capacity and formal language classes

8.3.1 Essential linguistic question regarding PS-grammar

Is there is a type of PS-grammar which generates exactly those structures which are characteristic of natural language?

8.3.2 Structural properties of regular PS-grammars

The generative capacity of regular grammar permits the recursive repetition of single words, but without any recursive correspondences.

8.3.3 Regular PS-grammar for ab^k (k $\geq$ 1)

$V =_{def}$ {S, B, a, b}
$V_T =_{def}$ {a, b}
$P =_{def}$ {S $\rightarrow$ a B,
B $\rightarrow$ b B,
B $\rightarrow$ b }

8.3.4 Regular PS-grammar for {a, b}$^+$

$V =_{def} \{S, a, b\}$
$V_T =_{def} \{a, b\}$
$P =_{def} \{S \rightarrow a\ S,$
$S \rightarrow b\ S,$
$S \rightarrow a,$
$S \rightarrow b\}$

8.3.5 Regular PS-grammar for $a^m b^k$ (k,m $\geq$ 1)

$V =_{def} \{S, S_1, S_2, a, b\}$
$V_T =_{def} \{a, b\}$
$P =_{def} \{S \rightarrow a\ S_1,$
$S_1 \rightarrow a\ S_1,$
$S_1 \rightarrow b\ S_2,$
$S_2 \rightarrow b\}$

8.3.6 Structural properties of context-free PS-grammars

The generative capacity of context-free grammar permits the recursive generation of pairwise inverse correspondences, e.g. a b c ... c b a.

8.3.7 Context-free PS-grammar for $a^k b^{3k}$

$V =_{def}$ {S, a, b}
$V_T =_{def}$ {a, b}
$P =_{def}$ { S $\rightarrow$ a S b b b,
S $\rightarrow$ a b b b}

8.3.8 Context-free PS-grammar for WW^R

$V =_{def}$ {S, a, b, c, d}, $V_T =_{def}$ {a, b, c, d}, $P =_{def}$ { S $\rightarrow$ a S a,
S $\rightarrow$ b S b,
S $\rightarrow$ c S c,
S $\rightarrow$ d S d,
S $\rightarrow$ a a,
S $\rightarrow$ b b,
S $\rightarrow$ c c,
S $\rightarrow$ d d}

8.3.9 Why WW exceeds the generative capacity of context-free PS-grammar

aa
abab
abcabc
abcdabcd
. . .

do not have a reverse structure. Thus, despite the close resemblance between WW^R and WW, it is simply impossible to write a PS-grammar like 8.3.8 for WW.

8.3.10 Why $a^k b^k c^k$ exceeds the generative capacity of context-free PS-grammar

a b c
a a b b c c
a a a b b b c c c
. . .

cannot be generated by a context-free PS-grammar because it requires a correspondence between three different parts – which exceeds the *pairwise* reverse structure of the context-free languages such as the familiar $a^k b^k$ and WW^R.

8.3.11 Structural properties of context-sensitive PS-grammars

Almost any language one can think of is context-sensitive; the only known proofs that certain languages are not CSL's are ultimately based on diagonalization.

J.E. Hopcroft and J.D. Ullman 1979, p. 224

8.3.12 PS-grammar for context-sensitive $a^k b^k c^k$

$V =_{def} \{S, B, C, D_1, D_2, a, b, c\}$
$V_T =_{def} \{a, b, c\}$
$P =_{def} \{$

$S \rightarrow a\ S\ B\ C,$	*rule* 1
$S \rightarrow a\ b\ C,$	*rule* 2
$C\ B \rightarrow D_1\ B,$	*rule* 3a
$D_1\ B \rightarrow D_1\ D_2,$	*rule* 3b
$D_1\ D_2 \rightarrow B\ D_2,$	*rule* 3c
$B\ D_2 \rightarrow B\ C,$	*rule* 3d
$b\ B \rightarrow b\ b,$	*rule* 4
$b\ C \rightarrow b\ c,$	*rule* 5
$c\ C \rightarrow c\ c\}$	*rule* 6

The rules 3a–3d jointly have the same effect as the (monotonic)

rule 3 C B → B C.

8.3.13 Derivation of a a a b b b c c c

	intermediate chains	rules
1.	S	
2.	a S B C	(1)
3.	a a S B C B C	(1)
4.	a a a b C B C B C	(2)
5.	a a a b B C C B C	(3)
6.	a a a b B C B C C	(3)
7.	a a a b B B C C C	(3)
8.	a a a b b B C C C	(4)
9.	a a a b b b C C C	(4)
10.	a a a b b b c C C	(5)
11.	a a a b b b c c C	(6)
12.	a a a b b b c c c	(6)

8.3.14 Structural properties of recursive languages

The context-sensitive languages are a proper subset of the recursive languages. The class of recursive languages is not reflected in the PS-grammar hierarchy because the PS-rule schema provides no suitable restriction (cf. 8.1.4) such that the associated PS-grammar class would generate exactly the recursive languages.

A language is recursive if and only if it is decidable, i.e., if there exists an algorithm which can determine in finitely many steps for arbitrary input whether or not the input belongs to the language. An example of a recursive language which is not context-sensitive is the Ackermann function.

8.3.15 Structural properties of unrestricted PS-grammars

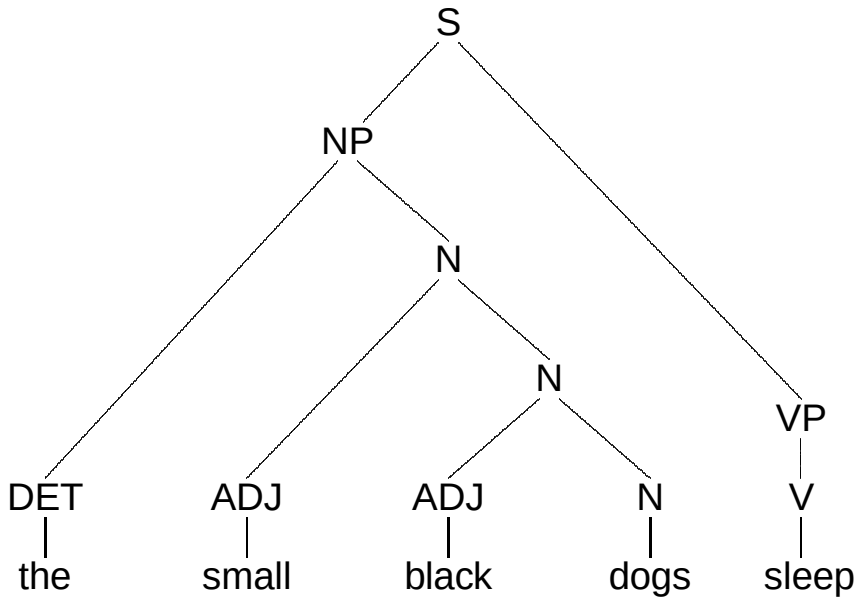
Because the right hand side of a rule may be shorter than the left hand side, a type 0 rules provides for the possibility of *deleting* parts of sequences already generated. For this reason, the class of recursively enumerable languages is undecidable.

8.4 PS-Grammar for natural language

8.4.1 PS-grammar for example 7.5.4

$V =_{def}$ {S, NP, VP, V, N, DET, ADJ, black, dogs, little, sleep, the}
$V_T =_{def}$ {black, dogs, little, sleep, the}
$P =_{def}$ { S $\rightarrow$ NP VP,
VP $\rightarrow$ V,
NP $\rightarrow$ DET N,
N $\rightarrow$ ADJ N,
N $\rightarrow$ dogs,
ADJ $\rightarrow$ little,
ADJ $\rightarrow$ black,
DET $\rightarrow$ the,
V $\rightarrow$ sleep}

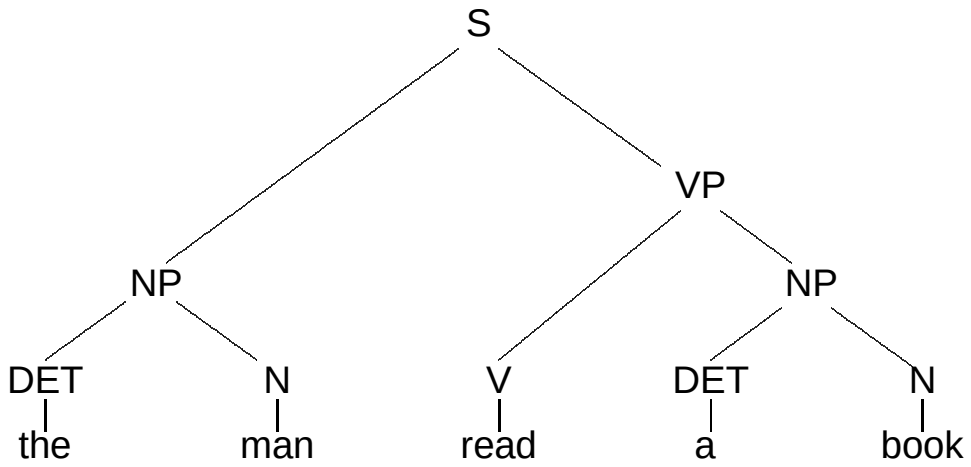
8.4.2 PS-grammar analysis of example 7.5.4



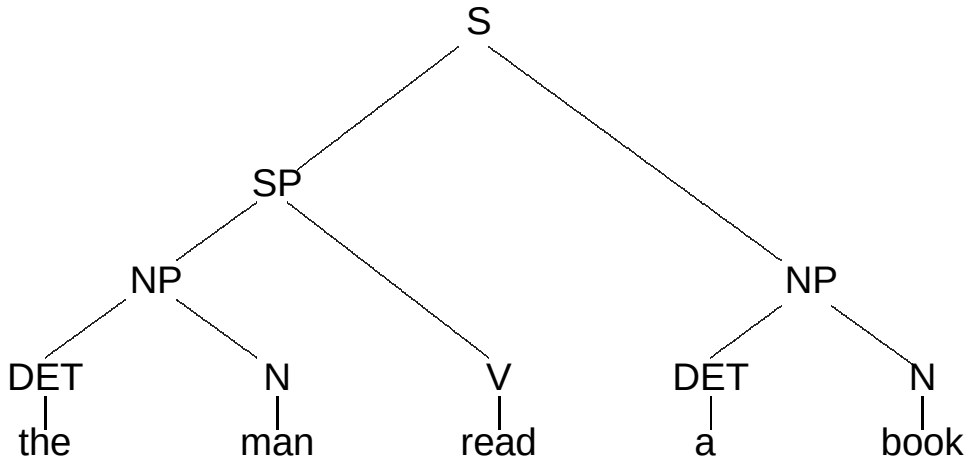
8.4.3 Definition of constituent structure

1. Words or constituents which belong together semantically must be dominated directly and exhaustively by a node.
2. The lines of a constituent structure may not cross (*nontangling condition*).

8.4.4 Correct constituent structure analysis



8.4.5 Incorrect constituent structure analysis

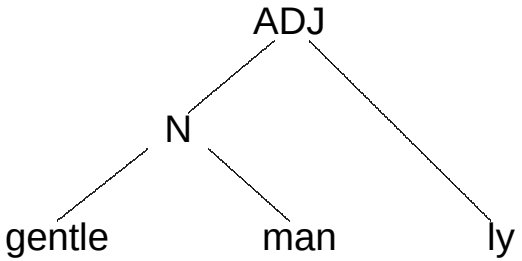


8.4.6 Origin of constituent structure

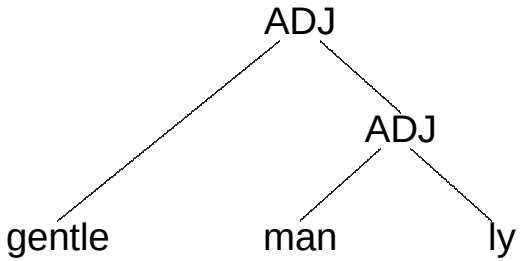
Historically, the notion of constituent structure evolved from the *immediate constituent analysis* of the American structuralist L. BLOOMFIELD (1887–1949) and the distribution tests of his student Z. Harris.

8.4.7 Immediate constituents in PS-grammar:

correct:



incorrect:



8.4.8 Substitution test

correct substitution:

Suzanne has [eaten] an apple

⇓

Suzanne has [cooked] an apple

incorrect substitution:

Suzanne has [eaten] an apple

⇓

* Suzanne has [desk] an apple

8.4.9 Movement test

correct movement:

Suzanne [has] eaten an apple $\Longrightarrow$ [has] Suzanne eaten an apple (?)

incorrect movement:

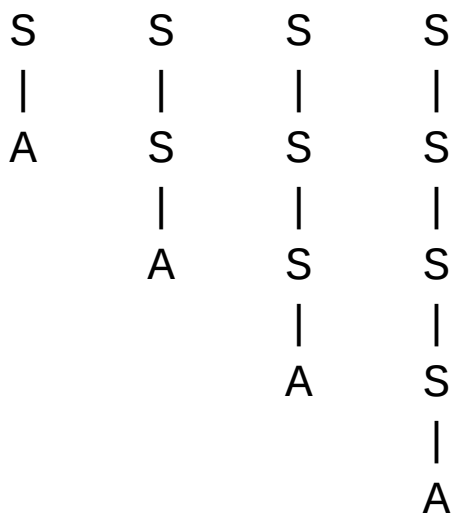
Suzanne has eaten [an] apple $\Longrightarrow$ * [an] Suzanne has eaten apple

8.4.10 Purpose of constituent structure

The distribution tests seemed important methodologically in order to support intuitions about the *correct segmentation* of sentences. The distinction between linguistically correct and incorrect phrase structures trees seemed necessary because for any finite string the number of possible phrase structures is infinite.

8.4.11 Infinite number of trees over a single word

Context-free rules: S $\rightarrow$ S, S $\rightarrow$ A
Indexed bracketing: $(A)_S$, $((A)_S)_S$, $(((A)_S)_S)_S$, $((((A)_S)_S)_S)_S$, etc.
Corresponding trees:

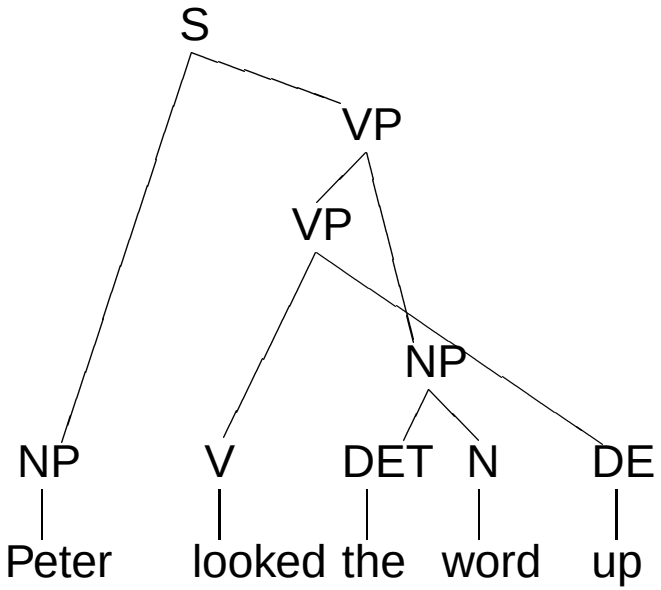


8.5 Constituent structure paradox

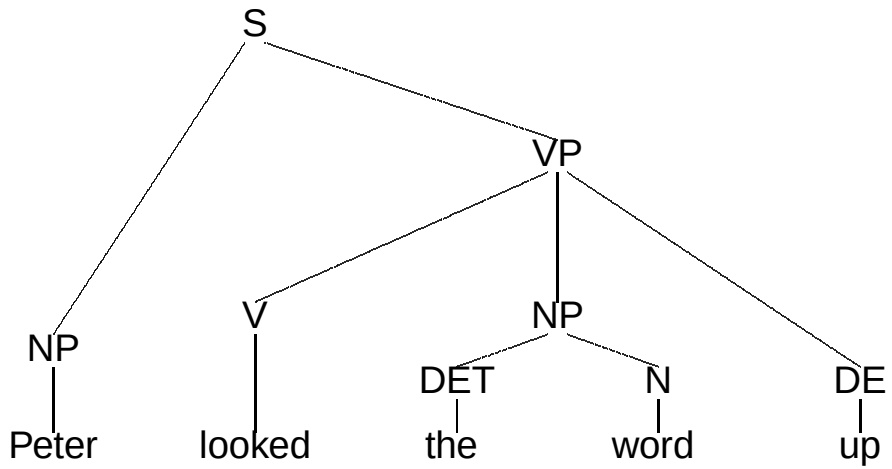
8.5.1 Constituent structure from the viewpoint of the SLIM theory of language

- Constituent structure and the distribution tests claimed to support it run counter to the time-linear structure of natural language.
- The resulting phrase structure trees have no communicative purpose.
- The principles of constituent structure cannot always be fulfilled.

8.5.2 Violating the second condition of 8.4.3



8.5.3 Violating the first condition of 8.4.3



8.5.4 Assumptions of transformational grammar

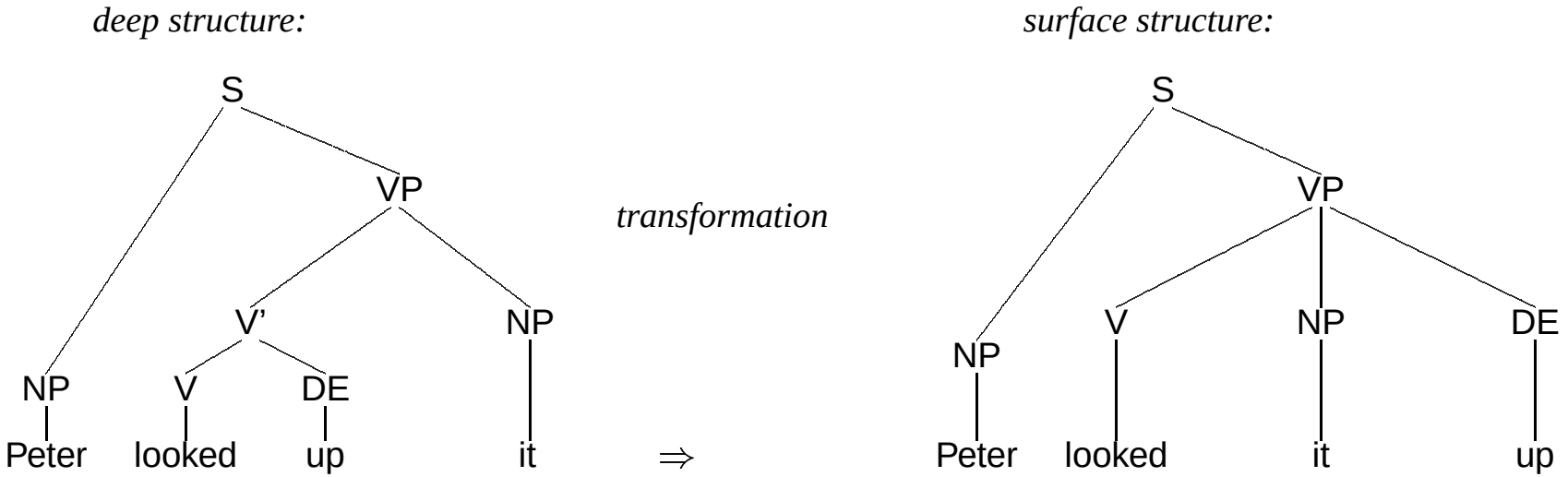
In order to maintain constituent structure as innate, transformational grammar distinguishes between a hypothetical deep structures claimed to be universal and the concrete language dependent surface structure.

- Thereby the two levels are assumed to be semantically equivalent,
- deep structures need not be grammatical, but must obey constituent structure, and
- surface structures must be grammatical, but need not obey constituent structure.

8.5.5 Example of a formal transformation

$$[[\text{V DE}]_{\text{V}'}\ [\text{DET N}]_{\text{NP}}]_{\text{VP}} \Rightarrow [\text{V}\ [\text{DET N}]_{\text{NP}}\ \text{DE}]_{\text{VP}}$$

8.5.6 Applying transformation 8.5.3



8.5.7 Mathematical consequences of adding transformations to PS-grammar

While the context-free deep structure is of low polynomial complexity (n^3), adding transformations raises complexity to recursively enumerable. In other words, transformational grammar is undecidable.

8.5.8 Example of a Bach-Peters-sentence

The man who deserves it will get the prize he wants.

8.5.9 Deep structure of a Bach-Peters-sentence