Foundations of Computational Linguistics

man-machine communication in natural language

ROLAND HAUSSER
Computational Linguistics
Universität Erlangen Nürnberg
Germany

CLUE

Part II Theory of Grammar

Part II
Theory of Grammar

7. Generative grammar

7.1 Language as a subset of the free monoid

7.1.1 Definition of language

A language is a set of word sequences.

7.1.2 Illustration of the free monoids over LX = {a,b}

ε
a, b
aa, ab, ba, bb
aaa, aab, aba, abb, baa, bab, bba, bbb
aaaa, aaab, aaba, aabb, abaa, abab, abba, abbb, . . .
. . .

7.1.3 Informal description of the artificial language $a^k b^k$ (with $k \geq 1$)

Its wellformed expressions consist of an arbitrary number of the word a followed by an equal number of the word b.

7.1.4 Wellformed expressions of $a^k b^k$

a b, a a b b, a a a b b b, a a a a b b b b, etc.,

7.1.5 Illformed expressions of $a^k b^k$

a, b, b a, b b a a, a b a b, etc.,

7.1.6 PS-grammar for $a^k b^k$

$S \rightarrow a\ S\ b$
$S \rightarrow a\ b$

A formal grammar may be viewed as a filter which selects the wellformed expressions of its language from the free monoid over the language's lexicon.

7.1.7 Elementary formalisms of generative grammar

1. Categorial or C-grammar
2. Phrase-structure or PS-grammar
3. Left-associative or LA-grammar

7.1.8 Algebraic definition

The algebraic definition of a generative grammar explicitly enumerates the basic components of the system, defining them and the structural relations between them using only notions of set theory.

7.1.9 Derived formalisms of PS-grammar

Syntactic Structures, Generative Semantics, Standard Theory (ST), Extended Standard Theory (EST), Revised Extended Standard Theory (REST), Government and Binding (GB), Barriers, Generalized Phrase Structure Grammar (GPSG), Lexical Functional Grammar (LFG), Head-driven Phrase Structure Grammar (HPSG)

7.1.10 Derived formalisms of C-grammar

Montague grammar (MG), Functional Unification Grammar (FUG), Categorial Unification Grammar (CUG), Combinatory Categorial Grammar (CCG), Unification-based Categorial Grammar (UCG)

7.1.11 Examples of semi-formal grammars

Dependency grammar (Tesnière 1959), systemic grammar (Halliday 1985), stratification grammar (Lamb ??)

7.2 Methodological reasons for generative grammar

7.2.1 Grammatically well-formed expression

the little dogs have slept earlier

7.2.2 Grammatically ill-formed expression

* earlier slept have dogs little the

7.2.3 Methodological consequences of generative grammar

- *Empirical*: formation of explicit hypotheses

  A formal rule system constitutes an explicit hypothesis about which input expressions are well-formed and which are not. This is an essential precondition for incremental improvements of the empirical description.

- *Mathematical*: determining formal properties

  A formal rule system is required for determining mathematical properties such as decidability, complexity, and generative capacity. These in turn determine whether the formalism is suitable for empirical description and computational realization.

- *Computational*: declarative specification for parsers

  A formal rule system may be used as a declarative specification of the parser, characterizing its necessary properties in contrast to accidental properties stemming from the choice of the programming environment, etc. A parser in turn provides the automatic language analysis needed for the verification of the individual grammars.

7.3 Adequacy of generative grammars

7.3.1 Desiderata of generative grammar for natural language

The generative analysis of natural language should be simultaneously

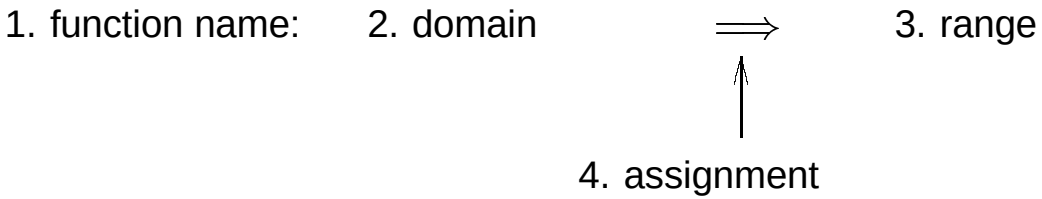
- defined *mathematically* as a formal theory of low complexity,
- designed *functionally* as a component of natural communication, and
- realized *methodologically* as an efficiently implemented computer program in which the properties of formal language theory and of natural language analysis are represented in a modular and transparent manner.

7.4 Formalism of C-grammar

7.4.1 The historically first generative grammar

Categorial grammar or C-grammar was invented by the Polish logicians LEŚNIEWSKI 1929 and AJDUKIEWICZ 1935 in order to avoid the Russell paradox in formal language analysis. C-grammar was first applied to natural language by BAR-HILLEL 1953.

7.4.2 Structure of a logical function



7.4.3 Algebraic definition of C-grammar

A C-grammar is a quintuple $<$ W, C, LX, R, CE $>$.

1. W is a finite set of word form surfaces.
2. C is a set of categories such that
   (a) *basis*
   u and v ϵ C,
   (b) *induction*
   if X and Y ϵ C, then also (X/Y) and $(X\backslash Y)$ ϵ C,
   (c) *closure*
   Nothing is in C except as specified in (a) and (b).
3. LX is a finite set such that LX $\subset$ (W $\times$ C).
4. R is a set comprising the following two rule schemata:
   $\alpha_{(Y/X)} \circ \beta_{(Y)} \Rightarrow \alpha\beta_{(X)}$
   $\beta_{(Y)} \circ \alpha_{(Y\backslash X)} \Rightarrow \beta\alpha_{(X)}$
5. CE is a set comprising the categories of *complete expressions*, with CE $\subseteq$ C.

7.4.4 Recursive definition of the infinite set C

Because the start elements u and v are in C so are (u/v), (v/u), (u\v), and (v\u) according to the induction clause. This means in turn that also ((u/v)/v), ((u/v)\u), (u/(u/v)), (v/(u/v)), etc., belong to C.

7.4.5 Definition of LX as finite set of ordered pairs

Each ordered pair is built from (i) an element of W and (ii) an element of C. Which surfaces (i.e. elements of W) take which elements of C as their categories is specified in LX by explicitly listing the ordered pairs.

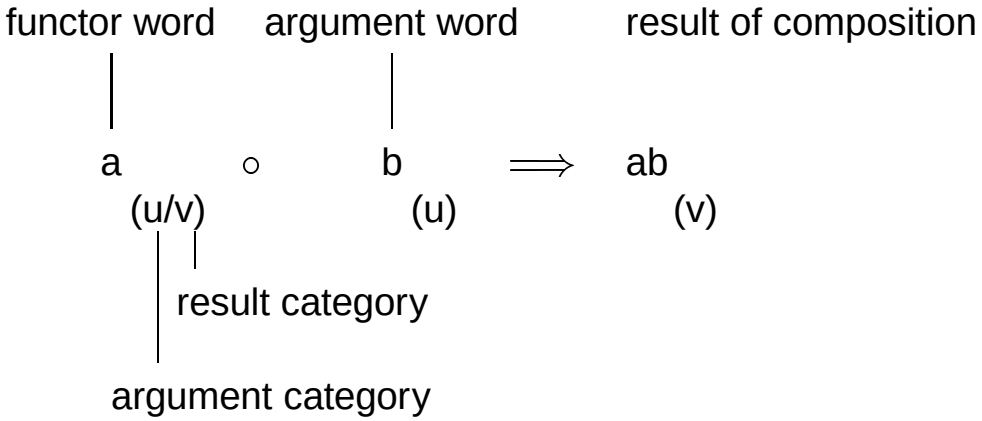
7.4.6 Definition of the set of rule schemata R

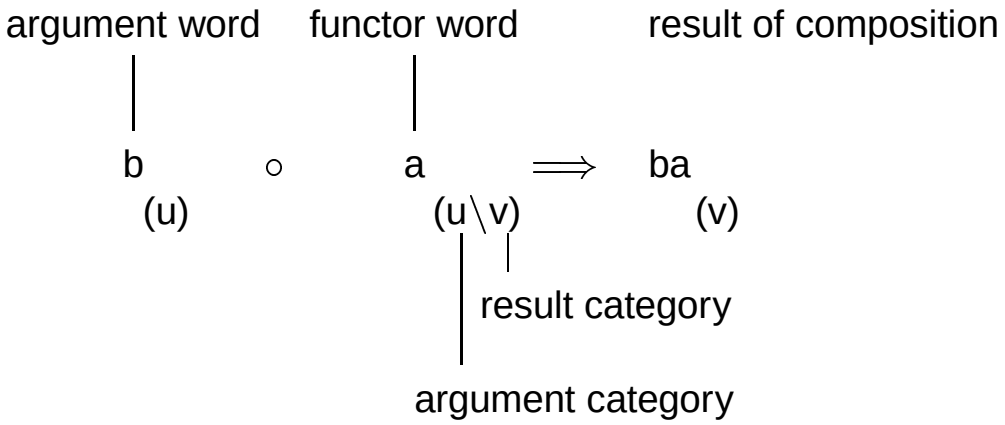
The rule schemata use the variables α and β to represent the surfaces of the functor and the argument, respectively, and the variables X and Y to represent their category patterns.

7.4.7 Definition of the set of complete expressions CE

Depending on the specific C-grammar and the specific language, this set may be finite and specified in terms of an explicit listing, or it may be infinite and characterized by patterns containing variables.

7.4.8 Implicit pattern matching in combinations of bidirectional C-grammar



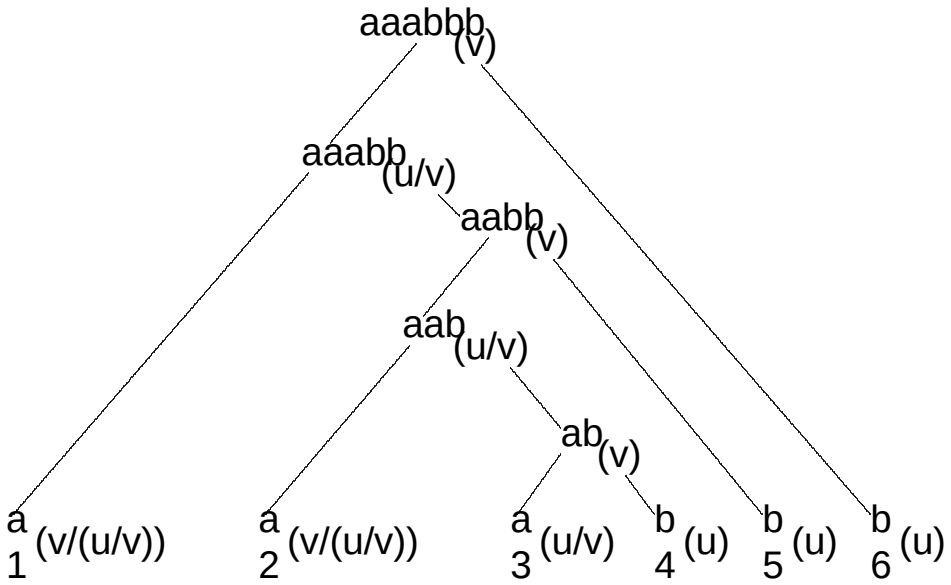


7.4.9 C-grammar for $a^k b^k$

$LX =_{def} \{a_{(u/v)}, b_{(u)}, a_{(v/(u/v))}\}$
$CE =_{def} \{(v)\}$

The word a has two lexical definitions with the categories (u/v) and $(v/(u/v))$, respectively, for reasons apparent in the following derivation tree.

7.4.10 Example of $a^k b^k$ derivation, for k = 3



7.5 C-grammar for natural language

7.5.1 C-grammar for a tiny fragment of English

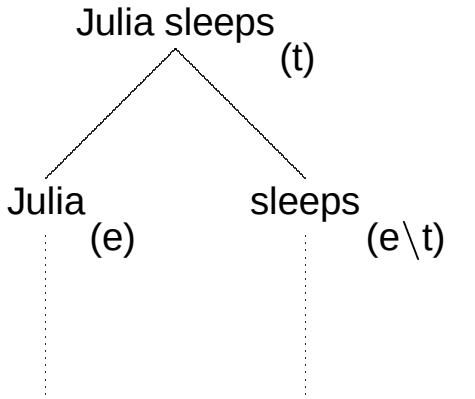
LX $=_{def}$ {$W_{(e)} \cup W_{(e\backslash t)}$}, where

$W_{(e)}$ = {Julia, Peter, Mary, Fritz, Suzy . . . }

$W_{(e\backslash t)}$ = {sleeps, laughs, sings . . . }

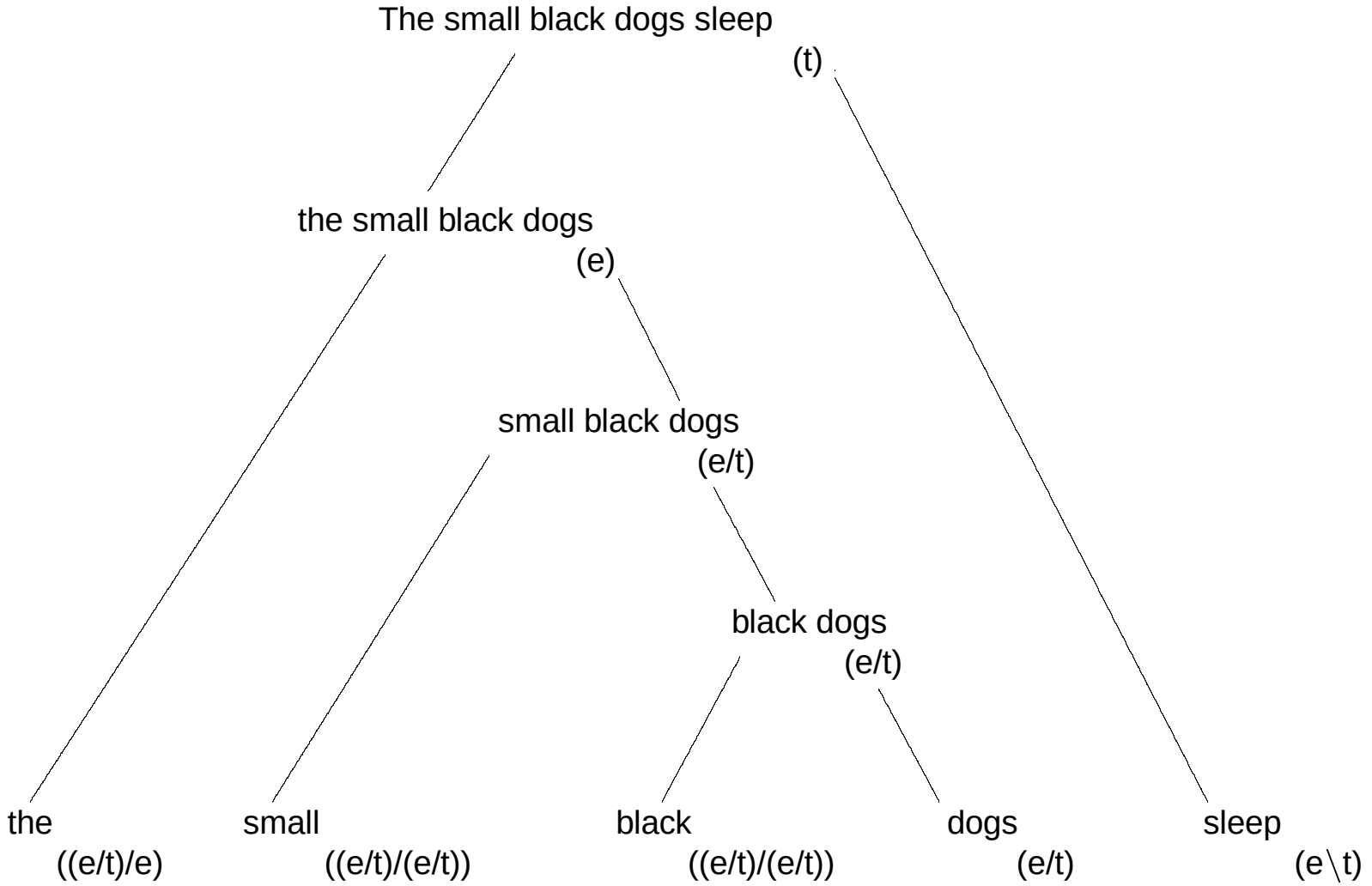
CE $=_{def}$ {(t)}

7.5.2 Simultaneous syntactic and semantic analysis



Denotations (in the model $\mathcal{M}$): entity {set of entities}

7.5.3 C-analysis of a natural language sentence



7.5.4 C-grammar for example 7.5.3

LX $=_{def}$ {$W_{(e)} \cup W_{(e\backslash t)} \cup W_{(e/t)} \cup W_{((e/t)/(e/t))} \cup W_{((e/t)/t)}$ }, where

$W_{(e)}$ = {Julia, Peter, Mary, Fritz, Suzy . . . }

$W_{(e\backslash t)}$ = {sleeps, laughs, sings . . . }

$W_{(e/t)}$ = {dog, dogs, cat, cats, table, tables . . . }

$W_{((e/t)/(e/t))}$ = {small, black . . . }

$W_{((e/t)/t)}$ = {a, the, every . . . }

CE $=_{def}$ {(t)}

7.5.5 Empirical disadvantages of C-grammar for natural language

- Deriving expressions relative to a C-grammar has the character of problem solving.
- The handling of alternative word orders and agreement phenomena requires an extremely high degree of lexical ambiguities.

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

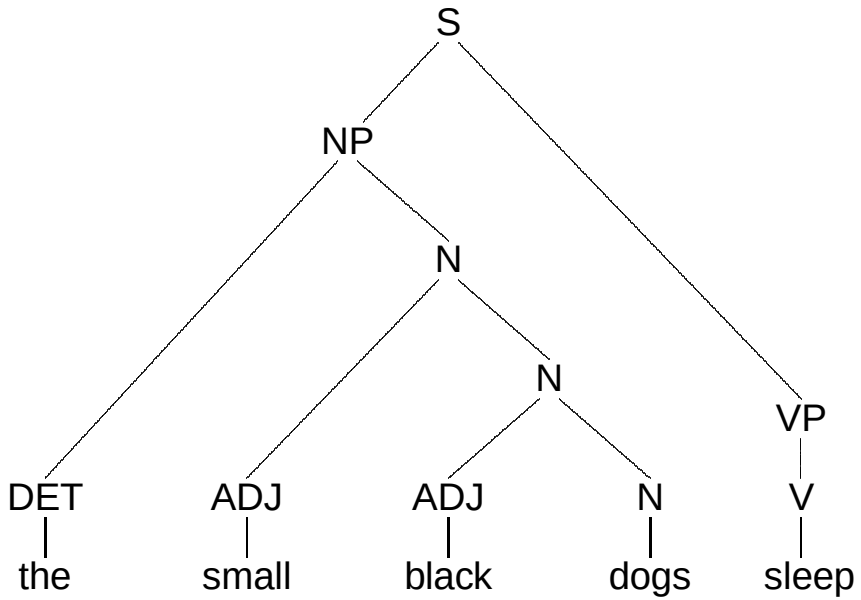
Because the right hand side of a rule may be shorter than the left hand side, a type 0 rules provides for the possibility of *deleting* parts of sequences already generated. For this reason, the class of recursively enumerable languages is undecidable.

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}

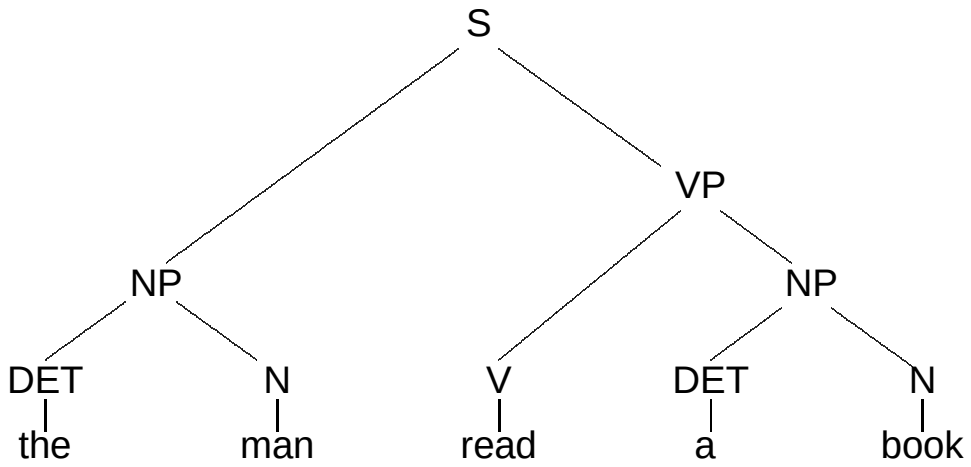
8.4.2 PS-grammar analysis of example 7.5.4



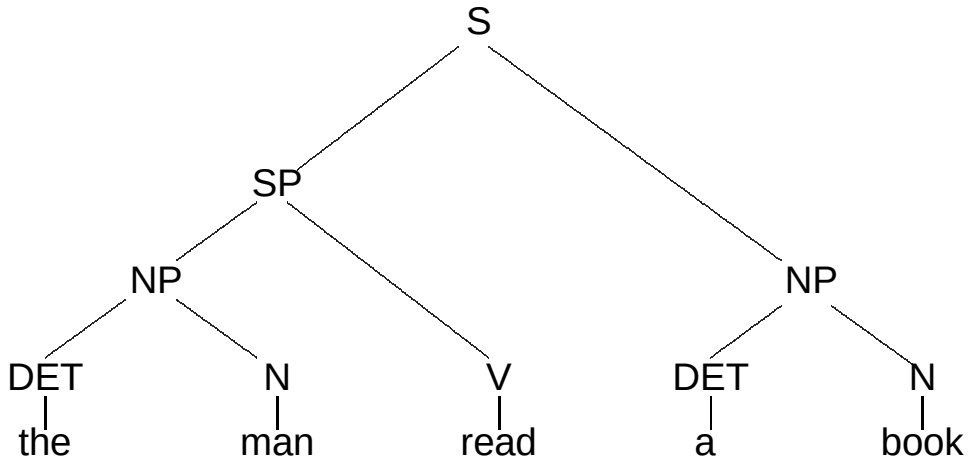
8.4.3 Definition of constituent structure

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

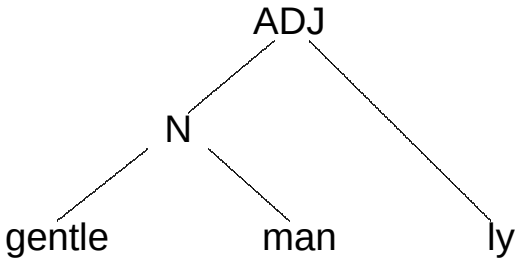


8.4.6 Origin of constituent structure

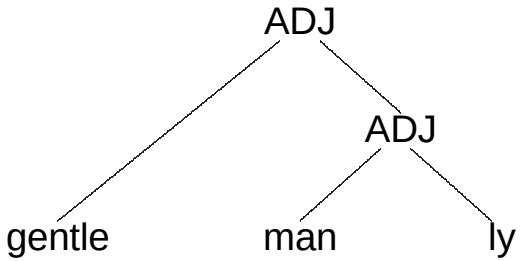
Historically, the notion of constituent structure evolved from the *immediate constituent analysis* of the American structuralist L. BLOOMFIELD (1887–1949) and the distribution tests of his student Z. Harris.

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:

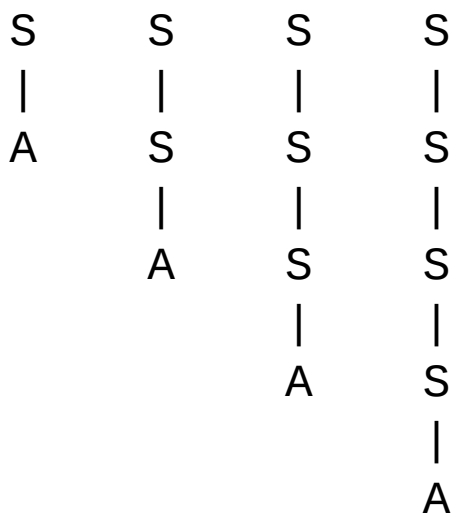
Suzanne has eaten [an] apple $\Longrightarrow$ * [an] Suzanne has eaten apple

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:

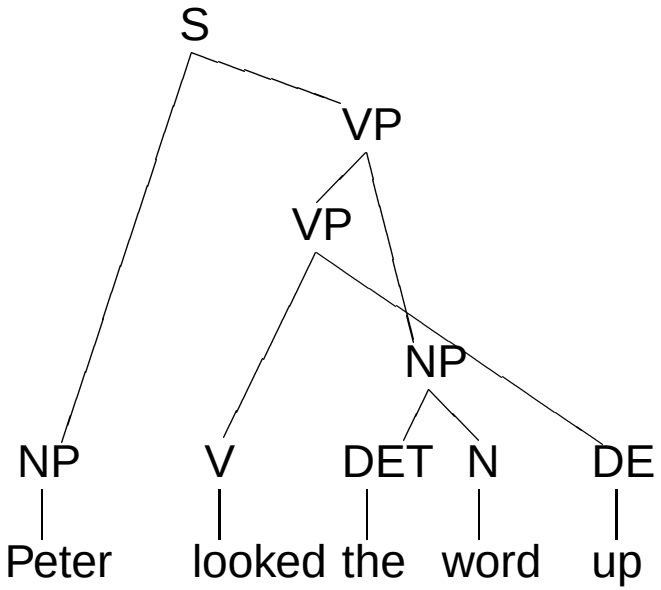


8.5 Constituent structure paradox

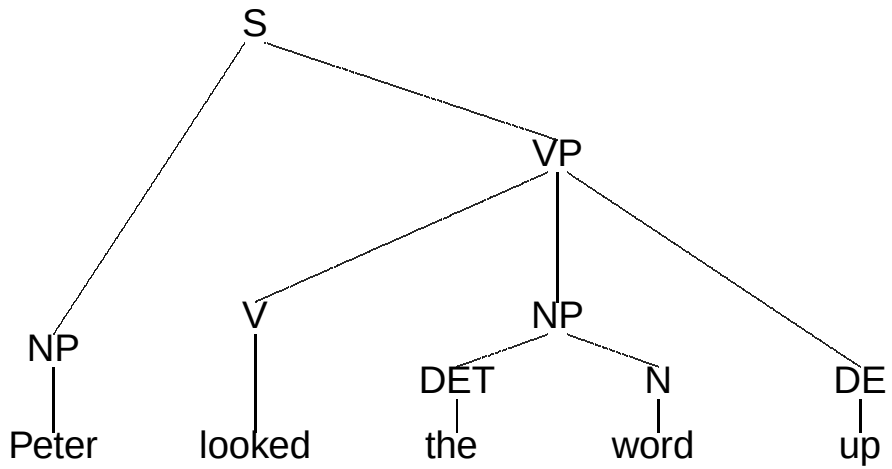
8.5.1 Constituent structure from the viewpoint of the SLIM theory of language

- 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

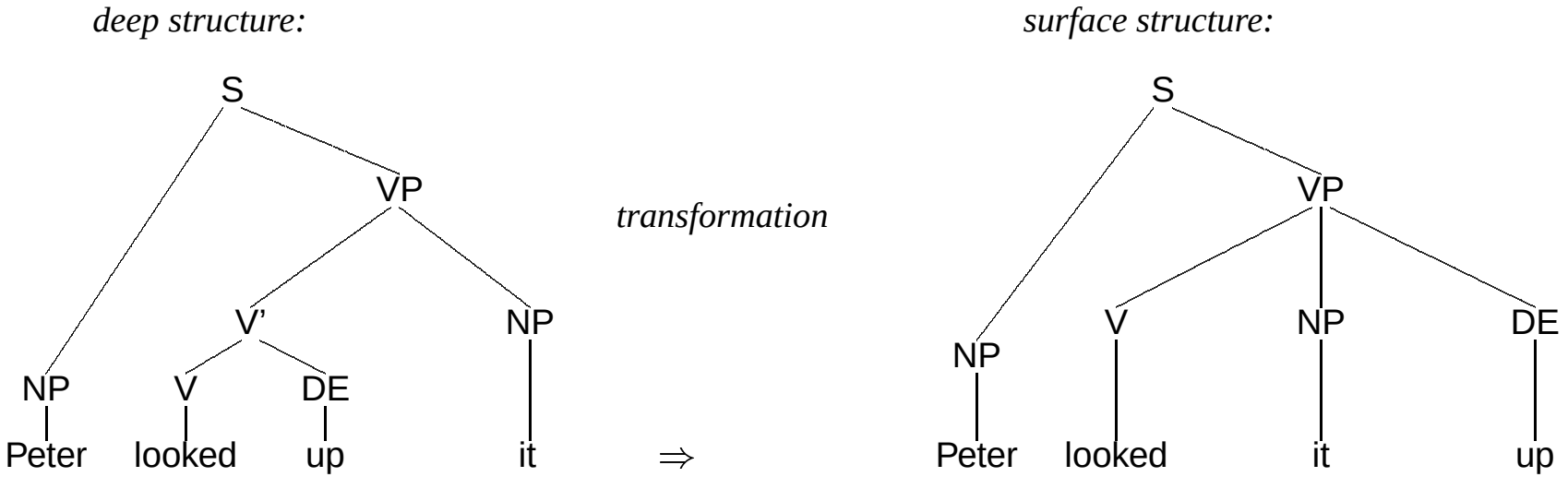
In order to maintain constituent structure as innate, transformational grammar distinguishes between a hypothetical deep structures claimed to be universal and the concrete language dependent surface structure.

- 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



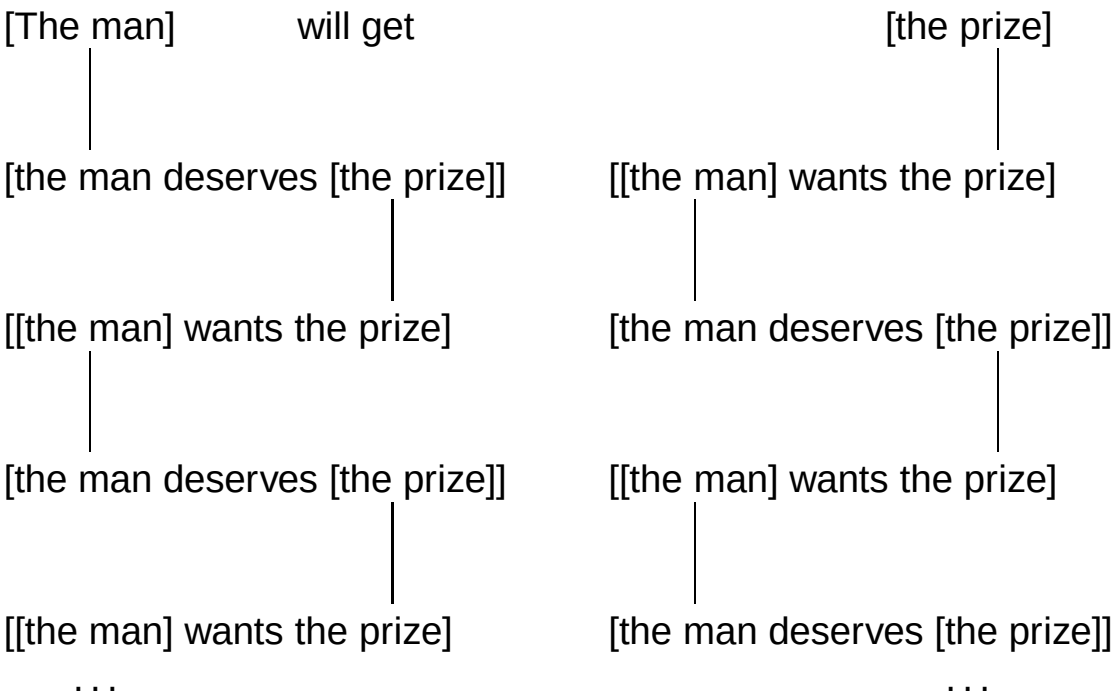
8.5.7 Mathematical consequences of adding transformations to PS-grammar

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



9. Basic notions of parsing

9.1 Declarative and procedural aspects of parsing

9.1.1 Declarative & procedural aspects in linguistics

- The *declarative* aspect of computational language analysis is represented by a generative grammar, written for the specific language to be analyzed within a general, mathematically well-defined formalism.
- The *procedural* aspect of computational language analysis comprises those parts of the computer program which interpret and apply the general formalism in the automatic analysis of language input.

9.1.2 Example

rule 1: A $\rightarrow$ B C
rule 2: B $\rightarrow$ c d
rule 3: C $\rightarrow$ e f

9.2 Fitting grammar onto language

9.2.1 Context-free structure in German

Der Mann, schläft.
(*the man*) (*sleeps*).
der die Frau, liebt,
(*who the woman*) (*loves*)
die das Kind, sieht,
(*who the child*) (*sees*)
das die Katze füttert,
(*who the cat*) (*feeds*)

9.2.2 Alternative implications of natural language not being context-free

1. PS-grammar is the only elementary formalism of generative grammar, for which reason one must accept that the natural languages are of high complexity and thus computationally intractable.
2. PS-grammar is not the only elementary formalism of generative grammar. Instead, there are other elementary formalisms which define other language hierarchies whose language classes are orthogonal to those of PS-grammar.

9.2.3 Possible relations between two grammar formalisms

- *No equivalence*
  Two grammar formalisms are not equivalent, if they generate/recognize different language classes; this means that the two formalisms are of different generative capacity.
- *Weak equivalence*
  Two grammar formalisms are weakly equivalent, if they generate/recognize the same language classes; this means that the two formalisms have the same generative capacity.
- *Strong equivalence*
  Two grammar formalisms are strongly equivalent, if they are (i) weakly equivalent, and moreover (ii) produce the same structural descriptions; this means that the two formalisms are no more than *notational variants*.

9.2.4 Weak equivalence between C-grammar and PS-grammar

> The problem arose of determining the exact relationships between these types of [PS-]grammars and the categorial grammars. I surmised in 1958 that the BCGs [Bidirectional Categorial Grammar *á la* 7.4.1] were of approximately the same strength as [context-free phrase structure grammars]. A proof of their equivalence was found in June of 1959 by Gaifman. ... The equivalence of these different types of grammars should not be too surprising. Each of them was meant to be a precise explicatum of the notion *immediate constituent grammars* which has served for many years as the favorite type of American descriptive linguistics as exhibited, for instance, in the well-known books by Harris [1951] and Hockett [1958].
>
> Y. Bar-Hillel 1960 [1964, p. 103]

9.2.5 General relations between notions of generative grammar

- *Languages* exist independently of generative grammars. A given language may be described by different formal grammars of different grammar formalisms.
- *Generative grammars* is (i) a general formal framework or (ii) a specific rule system defined for describing a specific language within the general framework.
- *Subtypes of generative grammars* result from different restrictions on the formal framework.
- *Language classes*
  The subtypes of a generative grammar may be used to divide the set of possible languages into different language classes.

  Nota bene: *languages* exist independently of the formal grammars which may generate them. The *language classes*, on the other hand, do not exist independently, but result from particular restrictions on particular grammar formalisms.
- *Parsers*
  Parsers are programs of automatic language analysis which are defined for whole subtypes of generative grammars.
- *Complexity*
  The complexity of a subtype of generative grammar is determined over the number of *primitive operations* needed by an equivalent abstract automaton or parsing program for analyzing expressions in the worst case.

9.3 Type transparency between grammar and parser

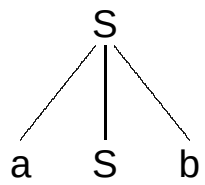
9.3.1 Natural view of a parser as the motor or driver of a grammar

> Miller and Chomsky's original (1963) suggestion is really that grammars be realized more or less directly as parsing algorithms. We might take this as a methodological principle. In this case we impose the condition that the logical organization of rules and structures incorporated in the grammar be mirrored rather exactly in the organization of the parsing mechanism. We will call this *type transparency*.
>
> R.C. Berwick & A.S. Weinberg 1984, p. 39.

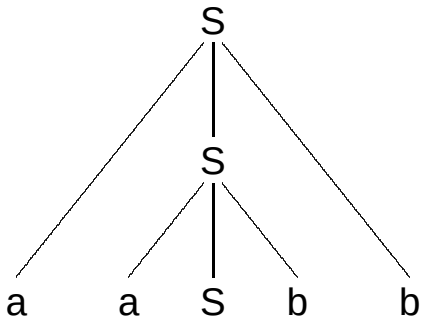
9.3.2 Definition of absolute type transparency

- For any given language, parser and generator use the *same* formal grammar,
- whereby the parser/generator applies the rules of the grammar *directly*.
- This means in particular that the parser/generator applies the rules in the *same order* as the grammatical derivation,
- that in each rule application the parser/generator takes the *same input* expressions as the grammar, and
- that in each rule application the parser/generator produces the *same output* expressions as the grammar.

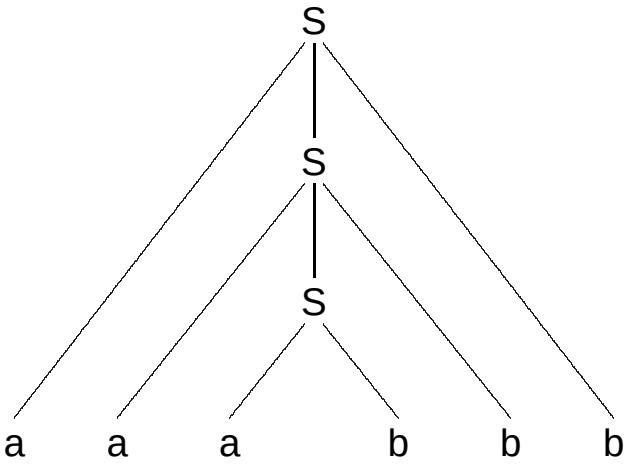
9.3.3 *Top-down* derivation of a a a b b b



S ⟶ a S b step 1

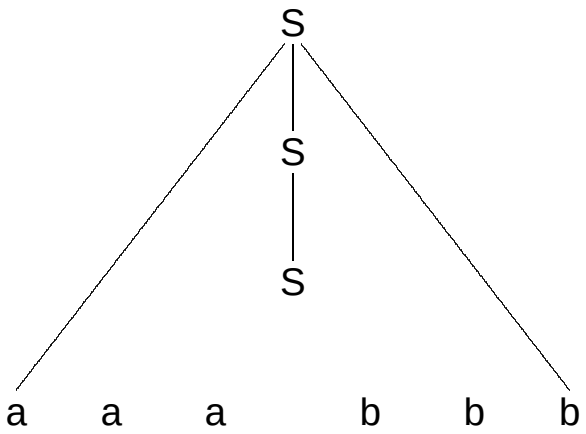


S ⟶ a S b step 2

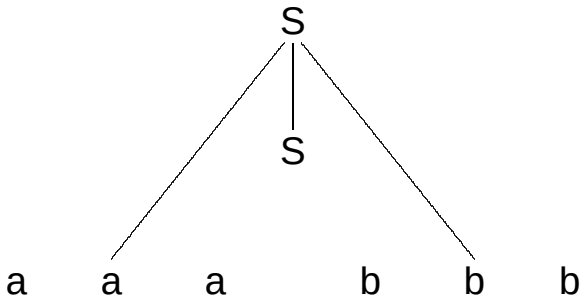


S ⟶ a b step 3

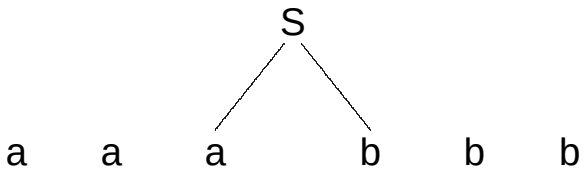
9.3.4 *Bottom-up* derivation of a a a b b b



S ⟵ a S b step 3

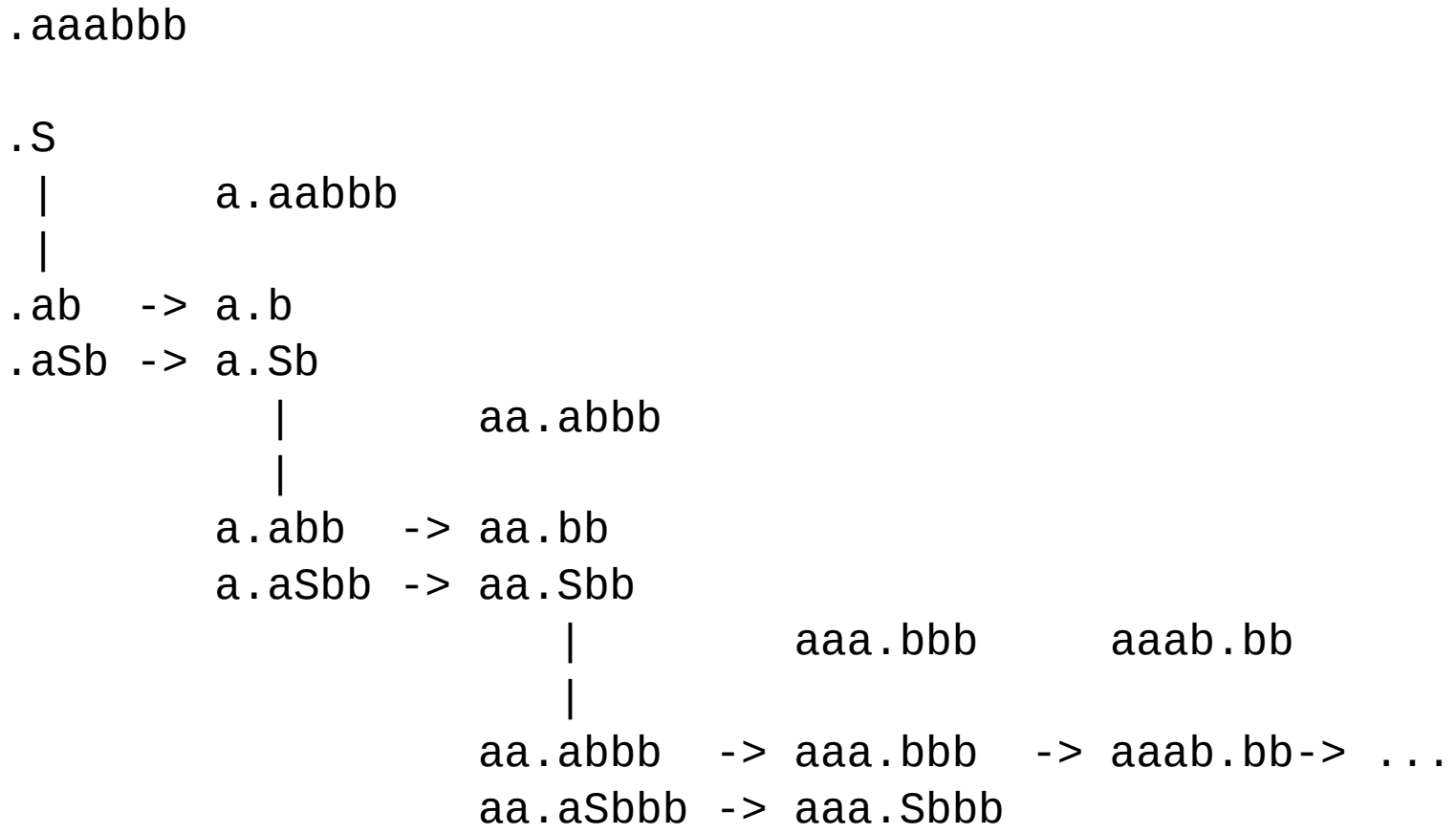


S ⟵ a S b step 2



S ⟵ a b step 1

9.3.5 The Earley algorithm analyzing $a^k b^k$

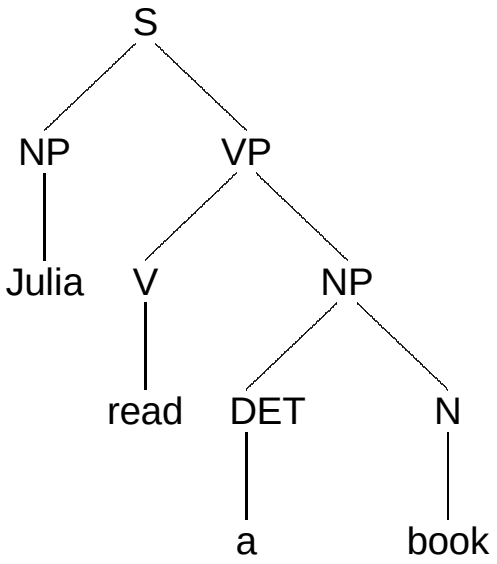


9.4 Input-output equivalence with the speaker-hearer

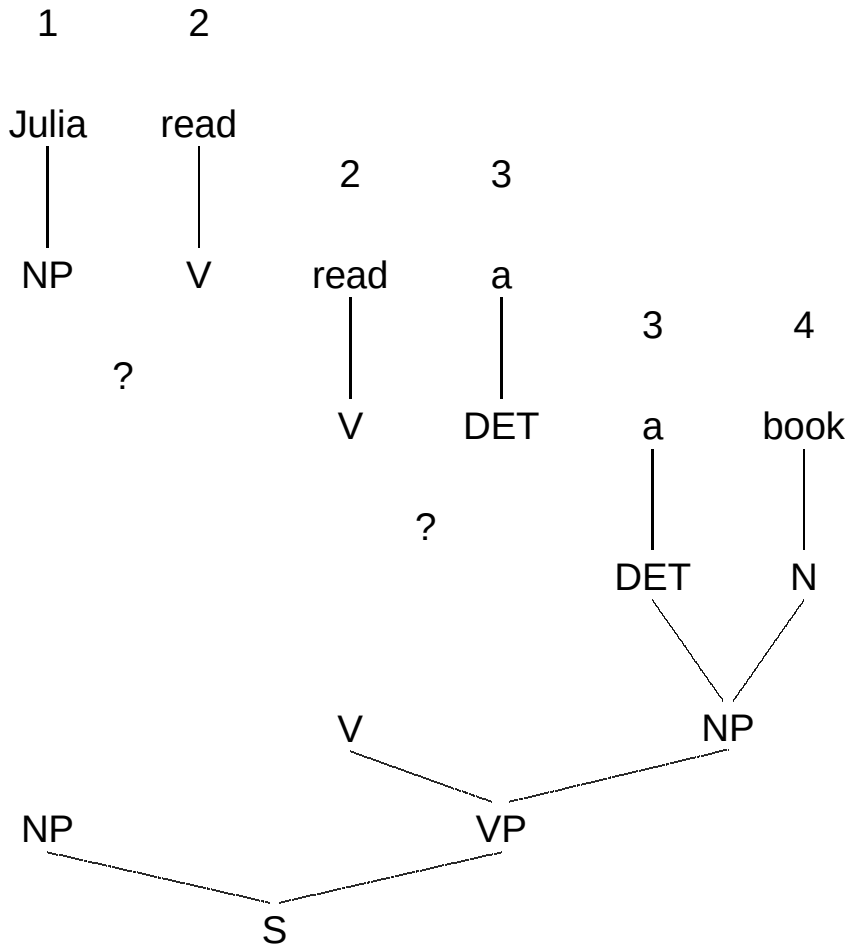
9.4.1 Context-free PS-grammar for a simple sentence of English

1. S → NP VP
2. NP → DET N
3. VP → V NP
4. NP → Julia
5. V → read
6. DET → a
7. N → book

9.4.2 PS-grammar analysis (*top-down* derivation)



9.4.3 Attempt of a time-linear analysis in PS-grammar



9.5 Desiderata of grammar for achieving convergence

9.5.1 Symptons of lacking convergence in nativism

- Development of ever new derived systems instead of consolidation.
- Additional mechanisms regarded as descriptively necessary have consistently degraded mathematical and computational properties.
- Empirical work has lead continuously to problems of the type descriptive aporia and embarrassment of riches.
- Practical systems of natural language processing pay either only lip service to the theoretical constructs of nativism or ignore them altogether.

9.5.2 Reasons for lacking convergence of nativism

- Nativism is empirically underspecified because it does not include a functional theory of communication.
- The PS-grammar formalism adopted by nativism is incompatible with the input-output conditions of the speaker-hearer.

9.5.3 Properties of PS-grammar

- *Mathematical:*
  Practical parsing algorithms exist only for context-free PS-grammar. It is of a sufficiently low complexity (n^3), but not of sufficient generative capacity for natural language. Extensions of the generative capacity for the purpose of describing natural language turned out to be of such high complexity (undecidable or exponential) that no practical parse algorithm can exist for them.
- *Computational:*
  PS-grammar is not type transparent. This prevents using the automatic traces of parsers for purposes of debugging and upscaling grammars. Furthermore, the indirect relation between the grammar and the parsing algorithm requires the use of costly routines and large intermediate structures.
- *Empirical:*
  The substitution-based derivation order of PS-grammar is incompatible with the time-linear structure of natural language.

9.5.4 Desiderata of a generative grammar formalism

1. The grammar formalism should be mathematically well-defined and thus
2. permit an explicit, declarative description of artificial and natural languages.
3. The formalism should be recursive (and thus decidable) as well as
4. type transparent with respect to its parsers and generators.
5. The formalism should define a hierarchy of different language classes in terms of structurally obvious restrictions on its rule system (analogous – but orthogonal – to the PS-grammar hierarchy),
6. whereby the hierarchy contains a language class of low, preferably linear, complexity the generative capacity of which is sufficient for a complete description of natural language.
7. The formalism should be input-output equivalent with the speaker-hearer (and thus use a time-linear derivation order).
8. The formalism should be suited equally well for production (in the sense of mapping meanings into surfaces) and interpretation (in the sense of mapping surfaces into meanings).

10. Left-associative grammar (LAG)

10.1 Rule types and derivation order

10.1.1 The notion *left-associative*

When we combine operators to form expressions, the order in which the operators are to be applied may not be obvious. For example, a + b + c can be interpreted as ((a + b) + c) or as (a + (b + c)). We say that + is *left-associative* if operands are grouped left to right as in ((a + b) + c). We say it is *right-associative* if it groups operands in the opposite direction, as in (a + (b + c)).

A.V. Aho & J.D. Ullman 1977, p. 47

10.1.2 Incremental left- and right-associative derivation

left-associative:

```
   a
 (a + b)
((a + b) + c)
(((a + b) + c) + d)
    ...
```

$\Longrightarrow$

right-associative:

```
                a
          (b + a)
     (c + (b + a))
(d + (c + (b + a)))
          ...
```

$\Longleftarrow$

10.1.3 Left-associative derivation order

Derivation is based on the principle of possible *continuations*

Used to model the time-linear structure of language

10.1.4 Irregular bracketing structures corresponding to the trees of C- and PS-grammar

```
(((a + b) + (c +d)) + e)
((a + b) + ((c +d)) + e)
(a + ((b + c)) + (d + e))
((a + (b + c)) + (d + e))
(((a + b) + c) + (d +e))
        ...
```

The number of these irregular bracketings grows exponentially with the length of the string and is infinite, if bracketings like (a), ((a)), (((a))), etc., are permitted.

10.1.5 Irregular bracketing structure

Derivation is based on the principle of possible *substitutions*

Used to model constituent structure

10.1.6 The principle of possible continuations

Beginning with the first word of the sentence, the grammar describes the possible continuations for each sentence start by specifying the rules which may perform the next grammatical composition (i.e., add the next word).

10.1.7 Schema of left-associative rule in LA-grammar

r_i: cat_1 $cat_2 \Rightarrow$ cat_3 rp_i

10.1.8 Schema of a canceling rule in C-grammar

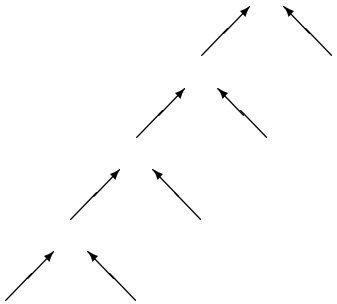
$\alpha_{(Y|X)} \circ \beta_{(Y)} \Rightarrow \alpha\beta_{(X)}$

10.1.9 Schema of a rewrite rule in PS-grammar

$A \rightarrow B\ C$

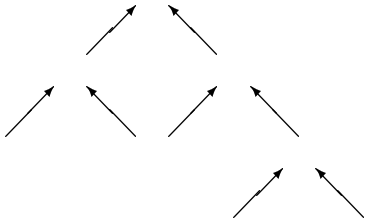
10.1.10 Three conceptual derivation orders

LA-grammar



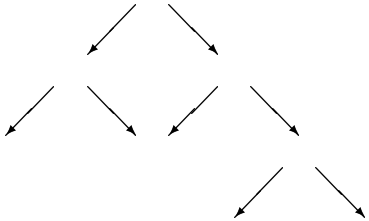
bot.-up left-associative

C-grammar



bottom-up amalgamating

PS-grammar



top-down expanding

10.2 Formalism of LA-grammar

10.2.1 Algebraic definition of LA-grammar

A left-associative grammar (or LA-grammar) is defined as a 7-tuple $<$W, C, LX, CO, RP, ST_S, ST_F $>$, where

1. W is a finite set of *word surfaces*.
2. C is a finite set of *category segments*.
3. LX $\subset$ (W $\times$ C^+) is a finite set comprising the *lexicon*.
4. CO = (co_0 ... co_{n-1}) is a finite sequence of total recursive functions from (C^* $\times$ C^+) into $C^* \cup \{\perp\}$, called *categorial operations*.
5. RP = (rp_0 ... rp_{n-1}) is an equally long sequence of subsets of n, called *rule packages*.
6. ST_S = {(cat_s rp_s), ...} is a finite set of *initial states*, whereby each rp_s is a subset of n called start rule package and each cat_s ϵ C^+.
7. ST_F = {(cat_f rp_f), ...} is a finite set of *final states*, whereby each cat_f ϵ C^* and each rp_f ϵ RP.

10.2.2 A concrete LA-grammar is specified by

1. a lexicon LX (cf. 3),
2. a set of initial states ST_S (cf. 6),
3. a sequence of rules r_i, each defined as an ordered pair (co_i, rp_i), and
4. a set of final states ST_F.

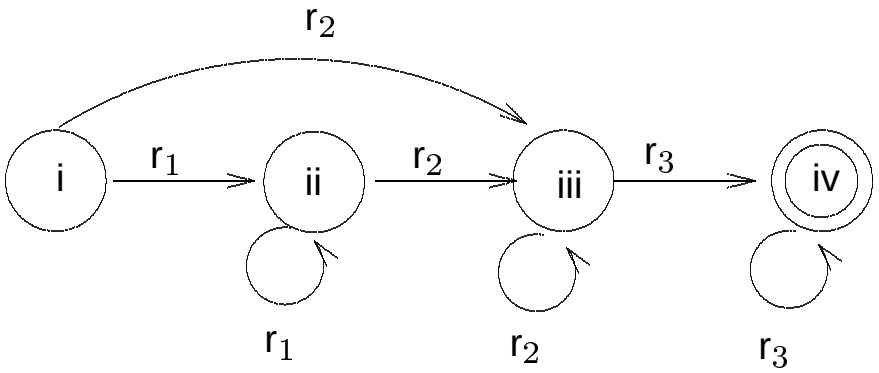
10.2.3 LA-grammar for $a^k b^k$

LX $=_{def}$ {[a (a)], [b (b)]}
$ST_S =_{def}$ {[(a) {r_1, r_2}]}
r_1: (X) (a) $\Rightarrow$ (aX) {r_1, r_2}
r_2: (aX) (b) $\Rightarrow$ (X) {r_2}
$ST_F =_{def}$ {[ε rp_2]}.

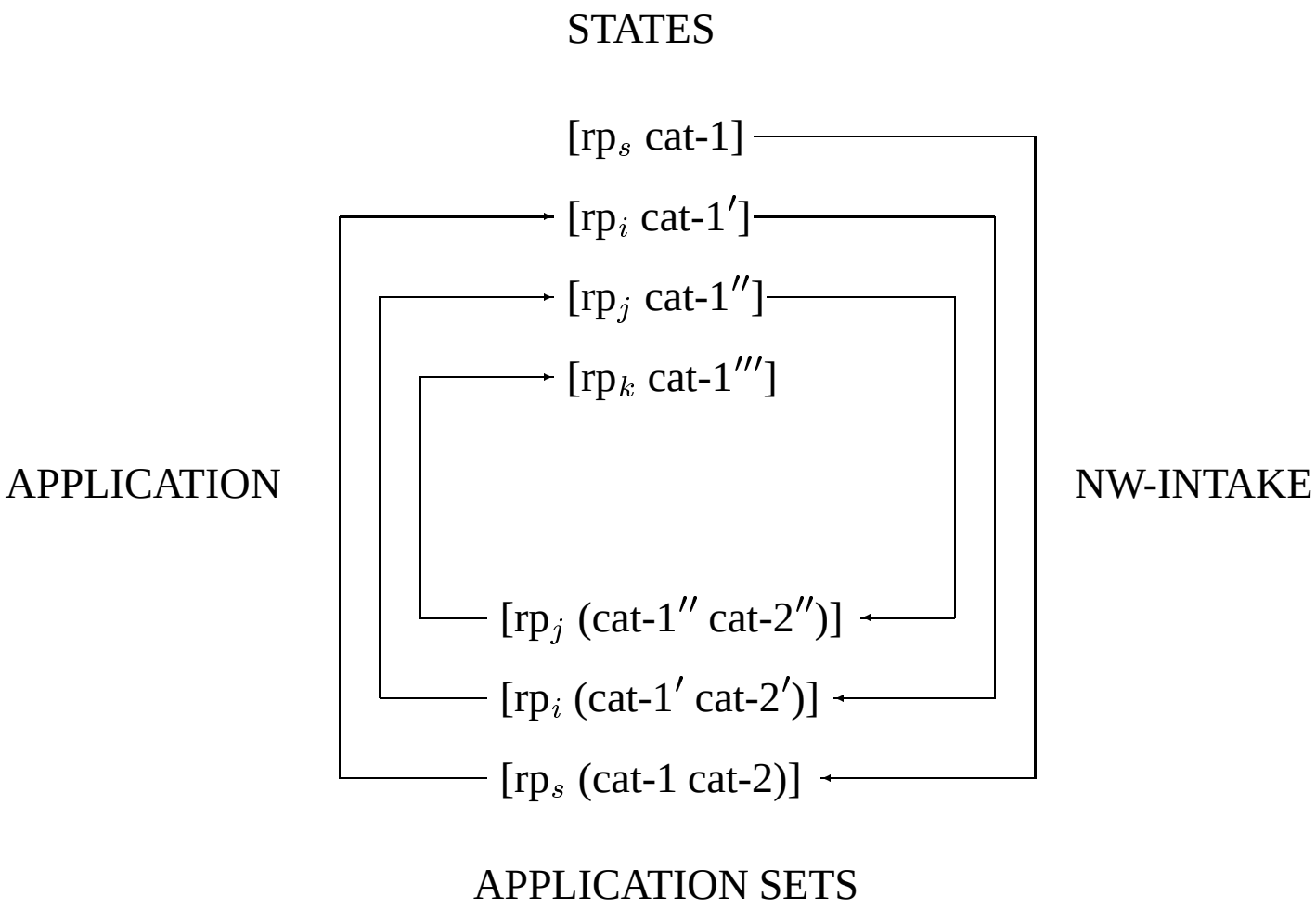
10.2.4 LA-grammar for $a^k b^k c^k$

LX $=_{def}$ {[a (a)], [b (b)], [c (c)]}
$ST_S =_{def}$ {[(a) {r_1, r_2}]}
r_1: (X) (a) $\Rightarrow$ (aX) {r_1, r_2}
r_2: (aX) (b) $\Rightarrow$ (Xb) {r_2, r_3}
r_3: (bX) (c) $\Rightarrow$ (X) {r_3}
$ST_F =_{def}$ {[ε rp_3]}.

10.2.5 The finite state backbone of the LA-grammar for $a^k b^k c^k$



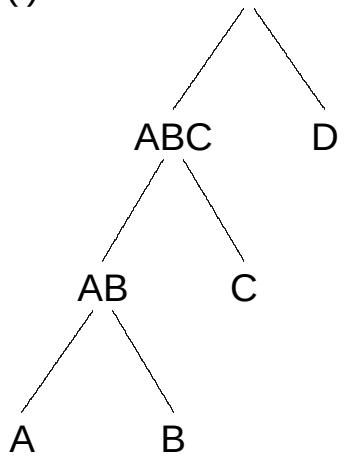
10.2.6 Recursion of left-associative algorithm



10.3 Time-linear analysis

10.3.1 LA-trees as structured lists

(i)



(ii)

ABCD
(D
ABC)
(C
AB)
(B
A)

(iii)

(A
B)
(AB
C)
(ABC
D)
ABCD

10.3.2 LA-grammar derivation of $a^k b^k$ for k =3

```
NEWCAT>  a a a b b b
    *START-0
    1
       (A) A
       (A) A
    *RULE-1
    2
       (A A) A A
       (A) A
    *RULE-1
    3
       (A A A) A A A
       (B) B
    *RULE-2
    4
       (A A) A A A B
       (B) B
    *RULE-2
    5
       (A) A A A B B
       (B) B
    *RULE-2
    6
       (NIL) A A A B B B
```

10.3.3 Interpretation of a history section

```
active rule package:          *START-0
composition number:           1
sentence start:               (A) A
next word:                    (A) A
successful rule:             *RULE-1
next composition number:      2
result:                       (A A) A A
```

10.3.4 Overlap between history sections

```
active rule package:          *RULE-1
composition number:           2
sentence start :              (A A) A A
next word:                    (A) A
successful rule :             *RULE-1
next composition number:      3
result:                       (A A A) A A A
```

10.4 Absolute type transparency of LA-grammar

10.4.1 Parsing aaabbbccc with active rule counter

```
NEWCAT>  a a a b b b c c c
;  1: Applying rules (RULE-1 RULE-2)
;  2: Applying rules (RULE-1 RULE-2)
;  3: Applying rules (RULE-1 RULE-2)
;  4: Applying rules (RULE-2 RULE-3)
;  5: Applying rules (RULE-2 RULE-3)
;  6: Applying rules (RULE-2 RULE-3)
;  7: Applying rules (RULE-3)
;  8: Applying rules (RULE-3)
; Number of rule applications: 14.

    *START-0
    1
       (A) A
       (A) A
    *RULE-1
    2
       (A A) A A
       (A) A
    *RULE-1
    3
       (A A A) A A A
       (B) B
    *RULE-2
    4
       (A A B) A A A B
       (B) B
    *RULE-2
    5
       (A B B) A A A B B
       (B) B
    *RULE-2
    6
        (B B B) A A A B B B
        (C) C
    *RULE-3
    7
        (C C) A A A B B B C
        (C) C
    *RULE-3
    8
        (C) A A A B B B C C
        (C) C
    *RULE-3
    9
        (NIL) A A A B B B C C C
```

10.4.2 Generating a representative sample in $a^k b^k c^k$

```
NEWCAT> (gram-gen 3 '(a b c))

Parses of length 2:
 A B
   2   (B)
 A A
   1   (A A)

Parses of length 3:
 A B C
   2 3   (NIL)
 A A B
   1 2   (A B)
 A A A
   1 1   (A A A)

Parses of length 4:
 A A B B
   1 2 2   (B B)
 A A A B
   1 1 2   (A A B)
 A A A A
   1 1 1   (A A A A)

Parses of length 5:
 A A B B C
   1 2 2 3   (B)
 A A A B B
   1 1 2 2   (A B B)
 A A A A B
   1 1 1 2   (A A A B)

Parses of length 6:
A A B B C C
   1 2 2 3 3   (NIL)
A A A B B B
   1 1 2 2 2   (B B B)
A A A A B B
   1 1 1 2 2   (A A B B)

Parses of length 7:
A A A B B B C
   1 1 2 2 2 3   (B B)
A A A A B B B
   1 1 1 2 2 2   (A B B B)

Parses of length 8:
A A A B B B C C
   1 1 2 2 2 3 3   (C)
A A A A B B B B
   1 1 1 2 2 2 2   (B B B B)

Parses of length 9:
A A A B B B C C C
```

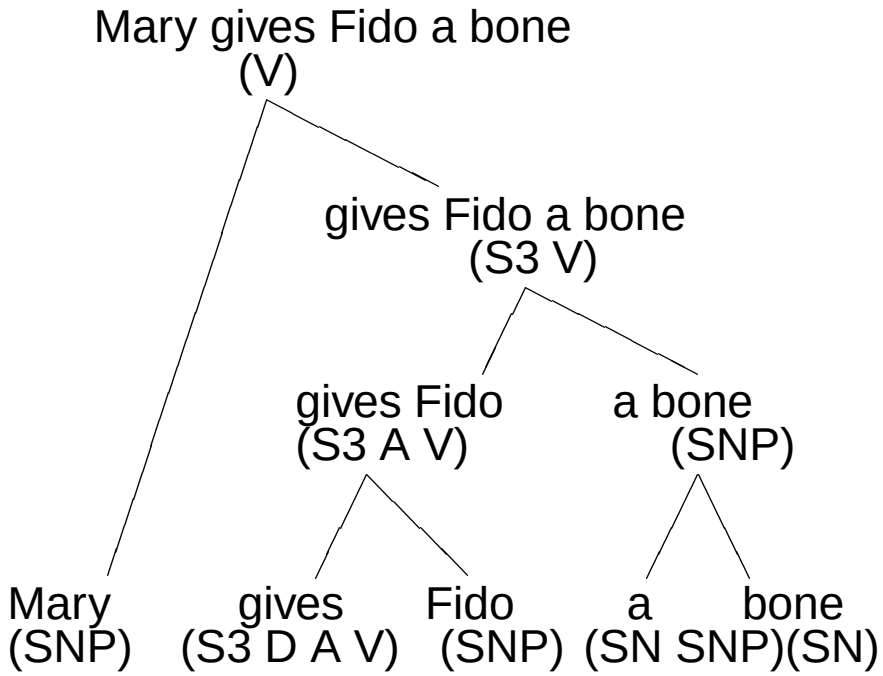
```
   1 1 2 2 2 3 3 3   (NIL)
 A A A A B B B B C
   1 1 1 2 2 2 2 3   (B B B)

Parses of length 10:
 A A A A B B B B C C
   1 1 1 2 2 2 2 3 3   (B B)

Parses of length 11:
 A A A A B B B B C C C
   1 1 1 2 2 2 2 3 3 3   (B)

Parses of length 12:
 A A A A B B B B C C C C
   1 1 1 2 2 2 2 3 3 3 3   (NIL)
```

10.4.3 Complete well-formed expression in $a^k b^k c^k$

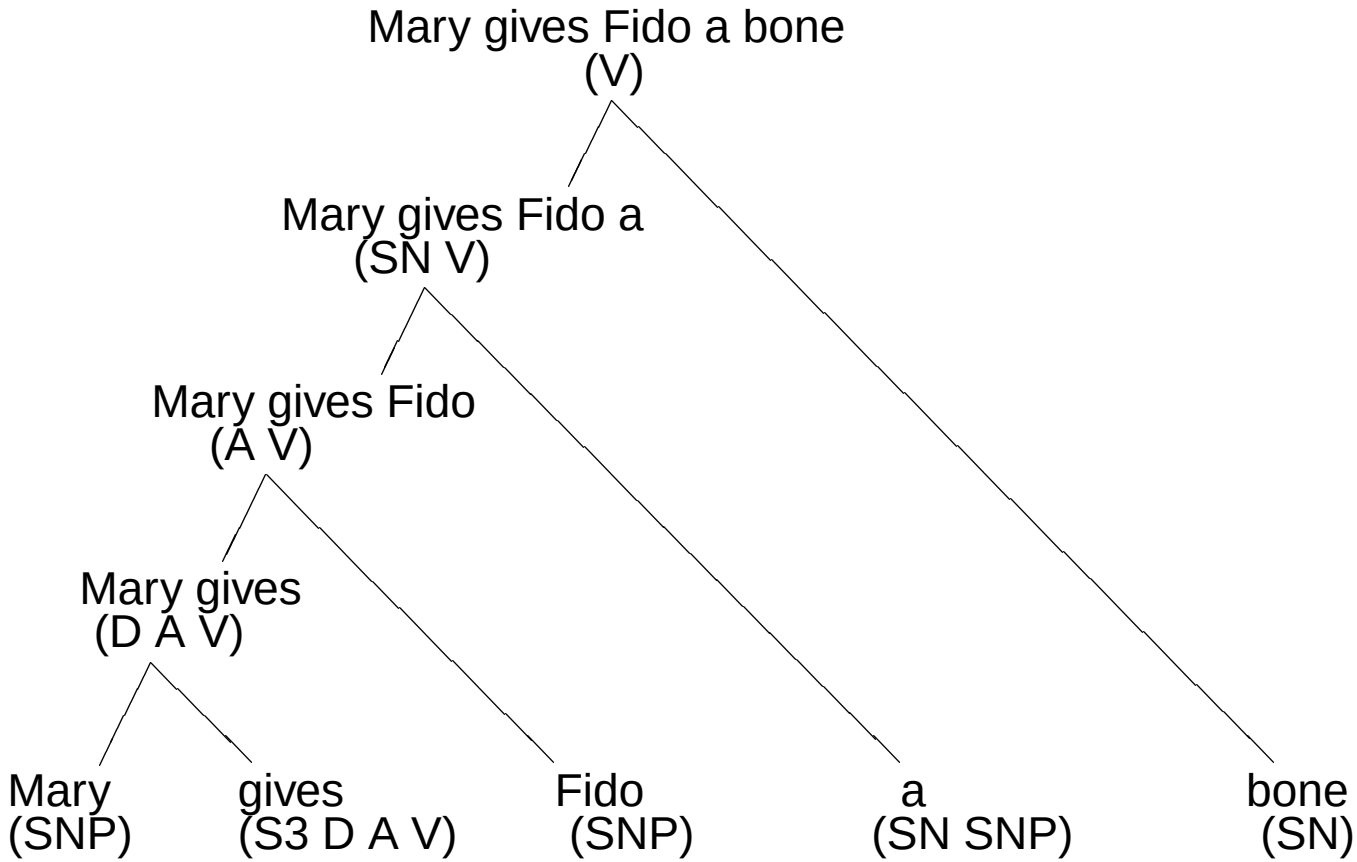
```
    A A A B B B C C C
      1 1 2 2 2 3 3 3   (NIL)
```

10.5 LA-grammar for natural language

10.5.1 Constituent structure analysis in C-grammar



10.5.2 Time-linear analysis in LA-grammar



10.5.3 Categorial operation combining Mary and gives

(SNP) (N D A V) ⇒ (D A V)

10.5.4 Categorial operation combining Mary gives and Fido

(D A V) (SNP) ⇒ (A V)

10.5.5 Categorial operation combining Mary gives Fido and a

(A V) (SN SNP) ⇒ (SN V)

10.5.6 Categorial operation combining Mary gives Fido a and book

(SN V) (SN) ⇒ (V)

10.5.7 Left-associative parsing of example 10.5.2

```
NEWCAT>  Mary gives Fido a bone \.

    *START
    1
       (SNP) MARY
       (S3 D A V) GIVES
    *NOM+FVERB
    2
       (D A V) MARY GIVES
       (SNP) FIDO
    *FVERB+MAIN
    3
       (A V) MARY GIVES FIDO
       (SN SNP) A
    *FVERB+MAIN
    4
       (SN V) MARY GIVES FIDO A
       (SN) BONE
    *DET+NOUN
    5
       (V) MARY GIVES FIDO A BONE
       (V DECL) .
    *CMPLT
    6
       (DECL) MARY GIVES FIDO A BONE .
```

10.5.8 Analysis of a discontinuous element

```
NEWCAT>  Fido dug the bone up \.

    *START
    1
       (SNP) FIDO
       (N A UP V) DUG
    *NOM+FVERB
    2
       (A UP V) FIDO DUG
       (SN SNP) THE
    *FVERB+MAIN
    3
       (SN UP V) FIDO DUG THE
       (SN) BONE
    *DET+NOUN
    4
       (UP V) FIDO DUG THE BONE
       (UP) UP
    *FVERB+MAIN
    5
       (V) FIDO DUG THE BONE UP
       (V DECL) .
    *CMPLT
    6
       (DECL) FIDO DUG THE BONE UP .
```

10.5.9 LA-analysis of ungrammatical input

```
NEWCAT>  the young girl give Fido the bone \.

ERROR
Ungrammatical continuation at: "GIVE"

    *START
    1
       (SN SNP) THE
       (ADJ) YOUNG
    *DET+ADJ
    2
       (SN SNP) THE YOUNG
       (SN) GIRL
     *DET+NOUN
    3
       (SNP) THE YOUNG GIRL
```

11. Hierarchy of LA-grammar

11.1 Generative capacity of unrestricted LAG

11.1.1 Generative capacity of unrestricted LA-grammar

Unrestricted LA-grammar accepts and generates all and only the recursive languages.

11.1.2 Theorem 1

Unrestricted LA-grammar accepts and generates *only* the recursive languages.

Proof: Assume an input string of finite length n. Each word in the input string has a finite number of readings (> 0).

Combination step 1: The finite set of start states ST_S and all readings of the first word w_1 result in a finite set of well-formed expressions $WE_1 = \{(ss' rp_S) \mid ss' \epsilon (W^+ \times C^+)\}$.

Combination step n: Combination step k-1, $k > 1$, has produced a finite set of well-formed expressions $WE_k = \{(ss' rp_i) \mid i \epsilon n, ss' \epsilon (W^+ \times C^*)$ and the surface of each ss' has length k$\}$. The next word w_{k+1} has a finite number of readings.

Therefore, the Cartesian product of all elements of WE_k and all readings of the current next word will be a finite set of pairs. Each pair is associated with a rule package containing a finite set of rules. Therefore, combination step k will produce only finitely many new sentence starts. The derivation of this finite set of new sentence starts is decidable because the categorial operations are defined to be total recursive functions.

Q.E.D.

11.1.3 Theorem 2

Unrestricted LA-grammar accepts and generates *all* recursive languages.

Proof: Let L be a recursive language with the alphabet W. Because L is recursive, there is a total recursive function ϱ: $W^* \rightarrow \{0,1\}$, i.e., the characteristic function of L. Let LAG^L be an LA-grammar defined as follows:

The set of word surfaces of LAG^L is W.

The set of category segments $C =_{def} W \cup \{0,1\}$.

For arbitrary e, f ϵ W^+, [e (f)] ϵ LX if and only if e = f.

$LX =_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)], . . . }
$ST_S =_{def}$ {[(seg_c) {r_1, r_2}]}, where seg_c ϵ {a, b, c, d, . . . }
r_1: (X) (seg_c) $\Rightarrow$ (X seg_c) {r_1, r_2}
r_2: (X) (seg_c) $\Rightarrow$ ϱ (X seg_c) { }
$ST_F =_{def}$ {[(1) rp_2]}

After any given combination step, the rule package rp_1 offers two choices: application of r_1 to continue reading the input string, or application of r_2 to test whether the input read so far is a well-formed expression of L. In the

latter case, the function ϱ is applied to the concatenation of the input categories, which are identical to the input surfaces. If the result of applying r_2 is [(1) rp_2], the input surface is accepted; if it is [(0) rp_2], it is rejected.

Since the categorial operations of LAG^L can be any total recursive function, LAG^L may be based on ϱ, the characteristic function of L. Therefore, LAG^L accepts and generates any recursive language.

Q.E.D.

11.1.4 Definition of the class of A-LAGs.

The class of A-LAGs consists of unrestricted LA-grammars and generates *all* recursive languages.

11.2 LA-hierarchy of A-, B-, and C-LAGs

11.2.1 Parameters of complexity

- The *amount* of computation per rule application required in the worst case.
- The *number* of rule applications relative to the length of the input needed in the worst case.

11.2.2 Main approaches to restricting LA-grammar

R1: Restrictions on the form of categorial operations in order to limit the maximal amount of computation required by arbitrary rule applications.
R2: Restrictions on the degree of ambiguity in order to limit the maximal number of possible rule applications.

11.2.3 Possible restrictions on categorial operations

R1.1: Specifying upper bounds for the *length* of categories;
R1.2: Specifying restrictions on *patterns* used in the definition of categorial operations.

11.2.4 Definition of the class of B-LAGs.

The class of *bounded* LA-grammars, or B-LAGs, consists of grammars where for any complete well-formed expression E the length of intermediate sentence start categories is bounded by $k \cdot n$, where n is the length of E and k is a constant.

11.2.5 Rule schemata with constant categorial operations

r_i: (seg$_1$...seg$_k$ X) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$
r_i: (X seg$_1$...seg$_k$) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$
r_i: (seg$_1$...seg$_m$ X seg$_{m+1}$...seg$_k$) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$

11.2.6 Rule schema with nonconstant categorial operation

r_i: (X seg$_1$...seg$_k$ Y) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$

11.2.7 Definition of the class of C-LAGs.

The class of *constant* LA-grammars, or C-LAGs, consists of grammars in which no categorial operation co$_i$ looks at more than k segments in the sentence start categories, for a finite constant k.

11.2.8 The hierarchy of A-LAGs, B-LAGs, and C-LAGs

The class of A-LAGs accepts and generates all recursive languages, the class of B-LAGs accepts and generates all context-sensitive languages, and the class of C-LAGs accepts and generates many context-sensitive, all context-free, and all regular languages.

11.3 Ambiguity in LA-grammar

11.3.1 Factors determining the number of rule applications

The number of rule application in an LA-derivation depends on

1. the length of the input;
2. the number of rules in the rule package to be applied in a certain combination to the analyzed input pair;
3. the number of readings existing at each combination step.

11.3.2 Impact on complexity

- Factor 1 is grammar-independent and used as the length n in formulas characterizing complexity .
- Factor 2 is a grammar-dependent constant.
- Only factor 3 may push the total number of rule applications beyond a linear increase. Whether for a given input more than one rule in a rule package may be successful depends on the input conditions of the rules.

11.3.3 Regarding factor 3: Possible relations between the input conditions of two rules

1. *Incompatible* input conditions: if there exist no input pairs which are accepted by both rules.

   Examples: (a X) (b) (a X) (b)
   (c X) (b) (a X) (c)

2. *Compatible* input conditions: if there exists at least one input pair accepted by both rules and there exists at least one input pair accepted by one rule, but not the other.

   Examples: (a X) (b)
   (X a) (b)

3. *Identical* input conditions: if all input pairs are either accepted by both rules or rejected by both rules.

11.3.4 Definition of unambiguous LA-grammars

An LA-grammar is unambiguous if and only if (i) it holds for all rule packages that their rules have *incompatible* input conditions and (ii) there are no lexical ambiguities.

11.3.5 Definition of syntactically ambiguous LA-grammars

An LA-grammar is syntactically ambiguous if and only if (i) it has at least one rule package containing at least two rules with *compatible* input conditions and (ii) there are no lexical ambiguities.

11.3.6 +global syntactic ambiguity

A syntactic ambiguity is called +global if it is a property of the whole sentence.

Example: Flying air planes can be dangerous.

11.3.7 –global syntactic ambiguity

A syntactic ambiguity is called -global if it is a property of only part of the sentence.

Example: The horse raced by the barn fell.

11.3.8 Role of the ±global distinction

In LA-grammar, the difference between +global and –global ambiguities consists in whether more than one reading survives to the end of the sentence (example 11.3.6) or not (example 11.3.7). The ±global distinction has no impact on complexity in LA-grammar and is made mainly for linguistic reasons.

11.3.9 +recursive syntactic ambiguity

An ambiguity is +recursive, if it originates within a recursive loop of rule applications.

Examples: the C-LAGs for WW^R (cf. 11.5.6) and WW (cf. 11.5.8), which are –global, and for SubsetSum (cf. 11.5.10), which are +global.

11.3.10 -recursive syntactic ambiguity

An ambiguity is –recursive, if none of the branches produced in the ambiguity split returns to the state which caused the ambiguity.

Examples: the C-LAG for $a^k b^k c^m d^m \cup a^k b^m c^m d^k$ (cf. 11.5.3), which is +global, and the C-LAGs for natural language in Chapter 17 and 18, which exhibit both +global and –global ambiguities.

11.3.11 Role of the ±recursive distinction

The ±recursive distinction is crucial for the analysis of complexity because it can be shown that in LA-grammars with nonrecursive ambiguities the maximal number of rule applications per combination step is limited by a grammar-dependent constant.

11.3.12 Theorem 3

The maximal number of rule applications in LA-grammar with only –recursive ambiguities is

$$(n - (R - 2)) \cdot 2^{(R-2)}$$

for n > (R - 2), where n is the length of the input and R is the number of rules in the grammar.

Proof: Parsing an input of length n requires (n – 1) combination steps. If an LA-grammar has R rules, then one of these rules has to be reapplied after R combination steps at the latest. Furthermore, the maximal number of rule applications in a combination step for a given reading is R.

According to the definition of –recursive ambiguity, rules causing a syntactic ambiguity may not be reapplied in a time-linear derivation path (reading). The first ambiguity-causing rule may produce a maximum of R-1 new branches (assuming its rule package contains all R rules except for itself), the second ambiguity causing rule may produce a maximum of R – 2 new branches, etc. If the different rules of the LA-grammar are defined with their maximally possible rule packages, then after R – 2 combination steps a maximum of $2^{(R-2)}$ readings is reached.

Q.E.D.

11.4 Complexity of grammars and automata

11.4.1 Choosing the primitive operation

> The Griffith and Petrick data is not in terms of actual time, but in terms of "primitive operations." They have expressed their algorithms as sets of nondeterministic rewriting rules for a Turing-machine-like device. Each application of one of these is a primitive operation. We have chosen as our primitive operation the act of adding a state to a state set (or attempting to add one which is already there). We feel that this is comparable to their primitive operation because both are in some sense the most complex operation performed by the algorithm whose complexity is independent of the size of the grammar and the input string.
>
> J. Earley 1970, p. 100

11.4.2 Primitive operation of the C-LAGs

The primitive operation of C-LAGs is a rule application (also counting unsuccessful attempts).

11.5 Subhierarchy of C1-, C2-, and C3-LAGs

11.5.1 The subclass of C1-LAGs

A C-LAG is a C1-LAG if it is not recursively ambiguous. The class of C1-languages parses in linear time and contains all deterministic context-free languages which can be recognized by a DPDA without ε-moves, plus context-free languages with –recursive ambiguities, e.g. $a^k b^k c^m d^m \cup a^k b^m c^m d^k$, as well as many context-sensitive languages, e.g. $a^k b^k c^k$, $a^k b^k c^k d^k e^k$, $\{a^k b^k c^k\}^*$, L_{square}, L^k_{hast}, a^{2^i}, $a^k b^m c^{k \cdot m}$, and $a^{i!}$, whereby the last one is not even an index language.

11.5.2 C1-LAG for context-sensitive a^{2^i}

LX $=_{def}$ {[a (a)]}
$ST_S =_{def}$ {[(a) {r_1}]}
r_1: (a) (a) ⇒ (aa) {r_2}
r_2: (aX) (a) ⇒ (Xbb) {r_2, r_3}
r_3: (bX) (a) ⇒ (Xaa) {r_2, r_3}
$ST_F =_{def}$ {[(aa) rp_1], [(bXb) rp_2], [(aXa) rp_3]}.

11.5.3 C1-LAG for ambiguous $a^k b^k c^m d^m \cup a^k b^m c^m d^k$

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)]}
$ST_S =_{def}$ {[(a) {r_1, r_2, r_5}]}
r_1: (X) (a) $\Rightarrow$ (a X) {r_1, r_2, r_5}
r_2: (a X) (b) $\Rightarrow$ (X) {r_2, r_3}
r_3: (X) (c) $\Rightarrow$ (c X) {r_3, r_4}
r_4: (c X) (d) $\Rightarrow$ (X) {r_4}
r_5: (X) (b) $\Rightarrow$ (b X) {r_5, r_6}
r_6: (b X) (c) $\Rightarrow$ (X) {r_6, r_7}
r_7: (a X) (d) $\Rightarrow$ (X) {r_7}
$ST_F =_{def}$ {[ε rp_4], [ε rp_7]}

11.5.4 The Single Return Principle (SRP)

A +recursive ambiguity is single return, if exactly one of the parallel paths returns into the state resulting in the ambiguity in question.

11.5.5 The subclass of C2-LAGs

A C-LAG is a C2-LAG if it is SR-recursively ambiguous. The class of C2-languages parses in polynomial time and contains certain nondeterministic context-free languages like WW^{R} and $\mathsf{L}^{\infty}_{hast}$, plus context-sensitive languages like WW, $\mathsf{W}^{\mathrm{k}\geq 3}$, $\{\mathsf{WWW}\}^{*}$, and $\mathsf{W}_1\mathsf{W}_2\mathsf{W}_1^{\mathrm{R}}\mathsf{W}_2^{\mathrm{R}}$.

11.5.6 C2-LAG for context-free WW^R

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)] . . . }
$ST_S =_{def}$ {[(seg_c) {r_1, r_2}]}, where $seg_c \in$ {a, b, c, d, . . . }
r_1: (X) (seg_c) $\Rightarrow$ (seg_c X) {r_1, r_2}
r_2: (seg_c X) (seg_c) $\Rightarrow$ (X) {r_2}
$ST_F =_{def}$ {[ε rp_2]}

11.5.7 Derivation structure of the worst case in WW^R

rules:	analyses:
2	a $ a
1 2 2	a a $ a a
1 1 2 2 2	a a a $ a a a
1 1 1 2 2	a a a a $ a a
1 1 1 1 2	a a a a a $ a
1 1 1 1 1	a a a a a a $

11.5.8 C2-LAG for context-sensitive WW

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)] . . . }
$ST_S =_{def}$ {[(seg_c) {r_1, r_2}]}, where $seg_c \in$ {a, b, c, d, . . . }
r_1: (X) (seg_c) $\Rightarrow$ (X seg_c) {r_1, r_2}
r_2: (seg_c X) (seg_c) $\Rightarrow$ (X) {r_2}
$ST_F =_{def}$ {[ε rp_2]}

11.5.9 C2-LAG for context-sensitive $W_1W_2W_1^RW_2^R$

LX $=_{def}$ {[a (a)], [b (b)]}
$ST_S =_{def}$ {[(seg_c) {r_{1a}}], [(seg_c) {r_{1b}}]}, where seg_c, $seg_d \in$ {a, b}
r_{1a}: (seg_c) (seg_d) $\Rightarrow$ (# seg_c seg_d) {r_2, r_3}
r_{1b}: (seg_c) (seg_d) $\Rightarrow$ (seg_d # seg_c) {r_3, r_4}
r_2: (X) (seg_c) $\Rightarrow$ (X seg_c) {r_2, r_3}
r_3: (X) (seg_c) $\Rightarrow$ (seg_c X) {r_3, r_4}
r_4: (X seg_c) (seg_c) $\Rightarrow$ (X) {r_4, r_5}
r_5: (seg_c X #) (seg_c) $\Rightarrow$ (X) {r_6}
r_6: (seg_c X) (seg_c) $\Rightarrow$ (X) {r_6}
$ST_F =_{def}$ {[ε rp_5], [ε rp_6]}

11.5.10 C3-LAG for SubsetSum.

LX $=_{def}$ {[0 (0)], [1 (1)], [# (#)]}
$ST_S =_{def}$ {[(seg$_c$) {r$_1$, r$_2$}]}, where seg$_c$ ϵ {0, 1}
seg$_c$ ε {0, 1}

r$_1$: (X)	(seg$_c$)	$\Rightarrow$	(seg$_c$ X)	{r$_1$, r$_2$}
r$_2$: (X)	(#)	$\Rightarrow$	(# X)	{r$_3$, r$_4$, r$_6$, r$_7$, r$_{12}$, r$_{14}$}
r$_3$: (X seg$_c$)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_3$, r$_4$, r$_6$, r$_7$}
r$_4$: (X #)	(#)	$\Rightarrow$	(# X)	{r$_3$, r$_4$, r$_6$, r$_7$, r$_{12}$, r$_{14}$}
r$_5$: (X seg$_c$)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_5$, r$_6$, r$_7$, r$_{11}$}
r$_6$: (X 1)	(0)	$\Rightarrow$	(1 X)	{r$_5$, r$_6$, r$_7$, r$_{11}$}
r$_7$: (X 0)	(1)	$\Rightarrow$	(1 X)	{r$_8$, r$_9$, r$_{10}$}
r$_8$: (X seg$_c$)	(seg$_c$)	$\Rightarrow$	(1 X)	{r$_8$, r$_9$, r$_{10}$}
r$_9$: (X 1)	(0)	$\Rightarrow$	(0 X)	{r$_5$, r$_6$, r$_7$, r$_{11}$}
r$_{10}$: (X 0)	(1)	$\Rightarrow$	(0 X)	{r$_8$, r$_9$, r$_{10}$}
r$_{11}$: (X #)	(#)	$\Rightarrow$	(# X)	{r$_3$, r$_4$, r$_6$, r$_7$, r$_{12}$, r$_{14}$}
r$_{12}$: (X 0)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_4$, r$_{12}$, r$_{14}$}
r$_{13}$: (X 0)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_{11}$, r$_{13}$, r$_{14}$}
r$_{14}$: (X 1)	(seg$_c$)	$\Rightarrow$	(1 X)	{r$_{11}$, r$_{13}$ r$_{14}$}

$ST_F =_{def}$ {[(X) rp$_4$]}

11.5.11 Types of restriction in LA-grammar

0. LA-type A: no restriction
1. LA-type B: The length of the categories of intermediate expressions is limited by $k \cdot n$, where k is a constant and n is the length of the input ($R1.1$, amount).
2. LA-type C3: The form of the category patterns results in a constant limit on the operations required by the categorial operations ($R1.2$, amount).
3. LA-type C2: LA-type C3 and the grammar is at most SR-recursively ambiguous ($R2$, number).
4. LA-type C1: LA-type C3 and the grammar is at most –recursively ambiguous ($R2$, number).

11.5.12 LA-grammar hierarchy of formal languages

restrictions	types of LAG	languages	complexity
LA-type C1	C1-LAGs	C1 languages	linear
LA-type C2	C2-LAGs	C2 languages	polynomial
LA-type C3	C3-LAGs	C3 languages	exponential
LA-type B	B-LAGs	B languages	exponential
LA-type A	A-LAGs	A languages	exponential

12. LA- and PS-hierarchies in comparison

12.1 Language classes of LA- and PS-grammar

12.1.1 Complexity degrees of the LA- and PS-hierarchy

	LA-grammar	*PS-grammar*
undecidable	—	recursively enumerable languages
exponential	A-languages B-languages C3-languages	context-sensitive languages
polynomial	C2-languages	context-free languages
linear	C1-languages	regular languages

12.2 Subset relations in the two hierarchies

12.2.1 Subset relations in the PS-hierarchy

regular lang. $\subset$ context-free lang. $\subset$ context-sensitive lang. $\subset$ rec. enum. languages

12.2.2 Subset relations in the LA-hierarchy

C1-languages $\subseteq$ C2-languages $\subseteq$ C3-languages $\subseteq$ B-languages $\subset$ A-languages

12.3 Non-equivalence of the LA- and PS-hierarchy

12.3.1 Languages which are in the same class in PS-grammar, but in different classes in LA-grammar

$a^k b^k$ and WW^R are in the same class in PS-grammar (i.e. context-free), but in different classes in LA-grammar: $a^k b^k$ is a C1-LAG parsing in linear time, while WW^R is a C2-LAG parsing in n^2.

12.3.2 Languages which are in the same class in LA-grammar, but in different classes in PS-grammar

$a^k b^k$ and $a^k b^k c^k$ are in the same class in LA-grammar (i.e. C1-LAGs), but in different classes in PS-grammar: $a^k b^k$ is context-free, while $a^k b^k c^k$ is context-sensitive.

12.3.3 Inherent complexity

The inherent complexity of a language is based on the number of operations required in the worst case on an abstract machine (e.g. a Turing or register machine). This form of analysis occurs on a very low level corresponding to machine or assembler code.

12.3.4 Class assigned complexity

The complexity of artificial and natural languages is usually analyzed at the abstraction level of grammar formalisms, whereby complexity is determined for the grammar type and its language class as a whole.

12.3.5 Difference between the two types of complexity

Languages which are inherently of high complexity (e.g. 3SAT and SUBSET SUM) are necessarily in a high complexity class (here exponential) in any possible grammar formalism.

Languages which are inherently of low complexity (e.g. $a^k b^k c^k$) may be assigned high or low class complexity, depending on the formalism.

12.3.6 PS-Grammar of L_{no}

$S \rightarrow 1S1$ $\quad S \rightarrow 1S$ $\quad S \rightarrow \#$

$S \rightarrow 0S0$ $\quad S \rightarrow 0S$

12.3.7 PS-grammar derivation of 10010#101 in L_{no}

derivation tree:	generated chains:	states:	
S			
		1.S1	1S1.
1 S 1	1S1	1.S	
		0.S0	0S0.
0 S 0	10S01	0.S	
		0.S0	
0 S	100S01	0.S	0S.
		1.S1	1S1.
1 S 1	1001S101	1.S	
		0.S0	
0 S	10010S101	0.S	0S.
#	10010#101	#.	

12.3.8 C3-LAG for L_{no}

LX $=_{def}$ {[0 (0)], [1 (1)], [# (#)]}
$ST_S =_{def}$ {[(seg_c) {r_1, r_2, r_3, r_4, r_5}] }, where seg_c, seg_d ϵ {0, 1}.

r_1: (seg_c)(seg_d)	$\Rightarrow \varepsilon$	{r_1, r_2, r_3, r_4, r_5}
r_2: (seg_c)(seg_d)	$\Rightarrow$ (seg_d)	{r_1, r_2, r_3, r_4, r_5}
r_3: (X)(seg_c)	$\Rightarrow$ (X)	{r_1 r_2, r_3, r_4, r_5}
r_4: (X)(seg_c)	$\Rightarrow$ (seg_c X)	{r_1 r_2, r_3, r_4, r_5}
r_5: (X)(#)	$\Rightarrow$ (X)	{r_6}
r_6: (seg_c X)(seg_c)	$\Rightarrow$ (X)	{r_6}

$ST_F =_{def}$ {[ε rp_6]}

12.4 Comparing the lower LA- and PS-classes

Context-free PS-grammar has been widely used because it provides the greatest amount of generative capacity within the PS-grammar hierarchy while being computationally tractable.

12.4.1 How suitable is context-free grammar for describing natural and programming languages?

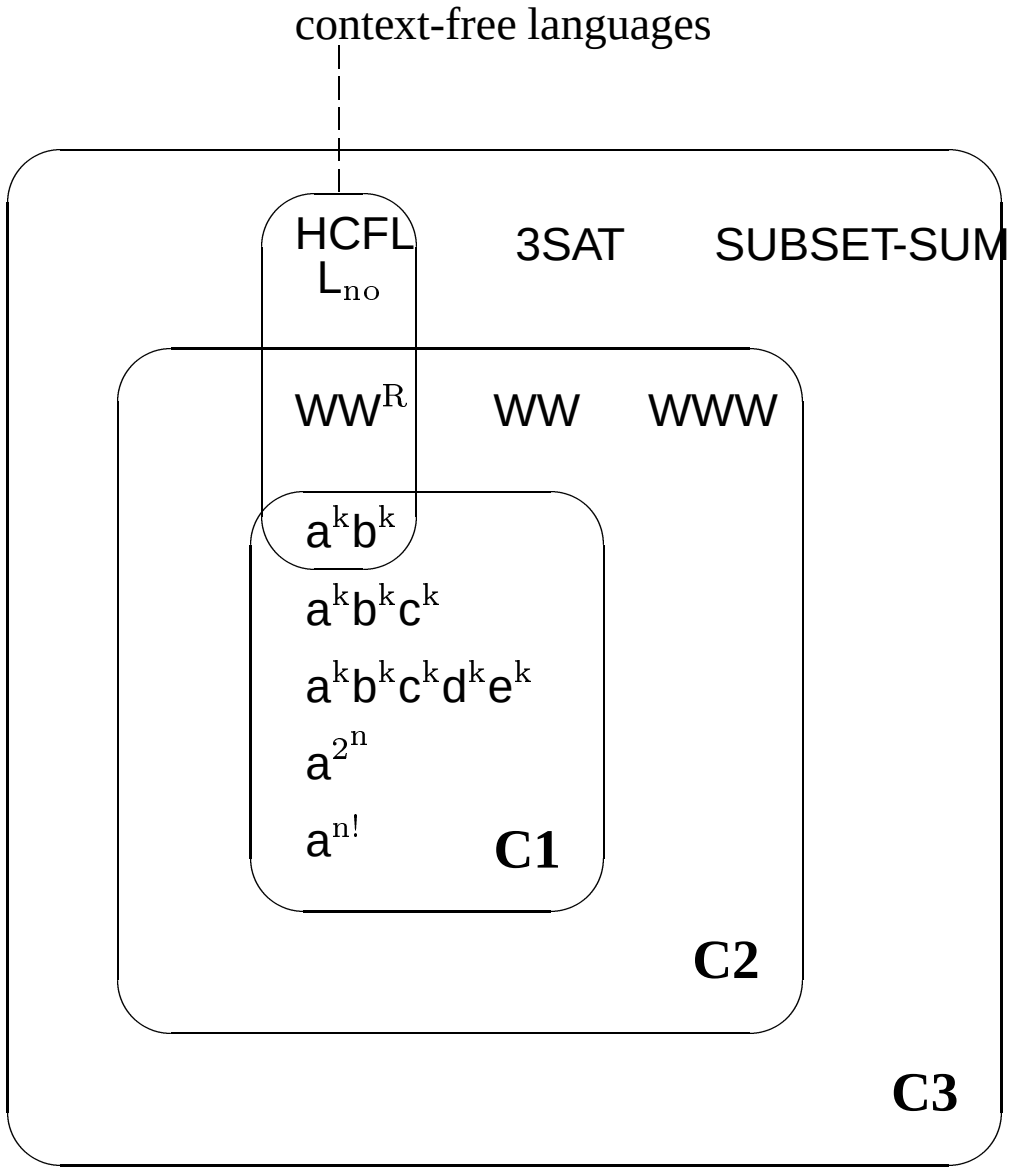
There is general agreement in linguistics that context-free PS-grammar does not properly fit the structures characteristic of natural language. The same holds for computer science:

> It is no secret that context-free grammars are only a first order approximation to the various mechanisms used for specifying the syntax of modern programming languages.
>
> S. Ginsberg 1980, p.7

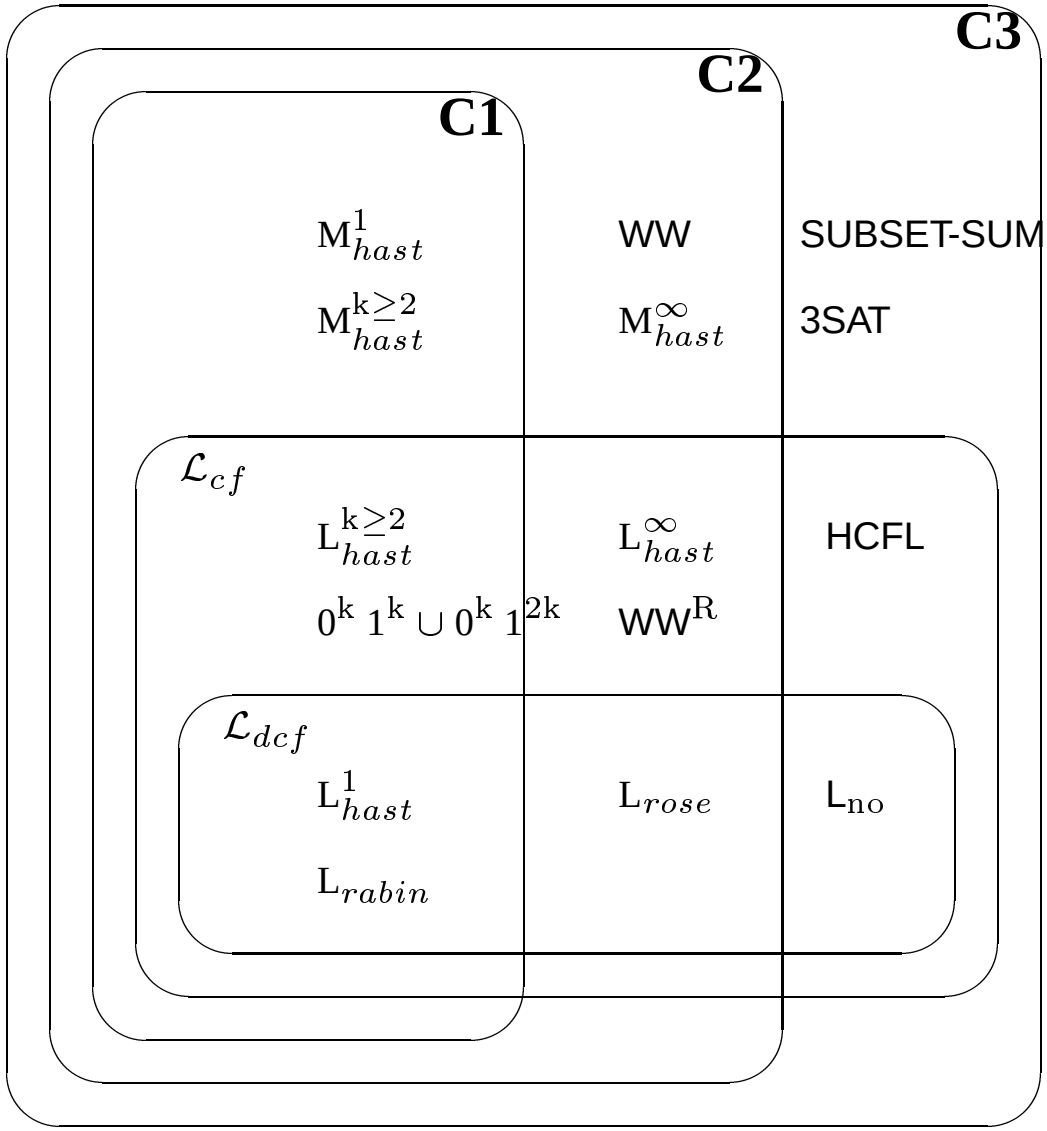
12.4.2 Conservative extensions of the PS-grammar hierarchy

regular lang. $\subset$ context-free lang. $\subset$ TAL $\subset$ index lang. $\subset$ context-sensitive lang. $\subset$ r.e. languages

12.4.3 Orthogonal relation between C- and cf-languages



12.4.4 Orthogonal $\mathcal{L}_{dcf}$, $\mathcal{L}_{cf}$, C_1, C_2, and C_3 classifications



12.5 Linear complexity of natural language

12.5.1 Why the natural languages are likely to be C-languages

In a context-sensitive language which is not a C-language, the category length would have to grow just within the LBA-definition of context-sensitive languages, but grow faster than the pattern-based categorial operations of the C-LAGs would permit.

That this type of language should be characteristic for the structure of natural language is highly improbable.

12.5.2 If the natural languages are C-LAGs

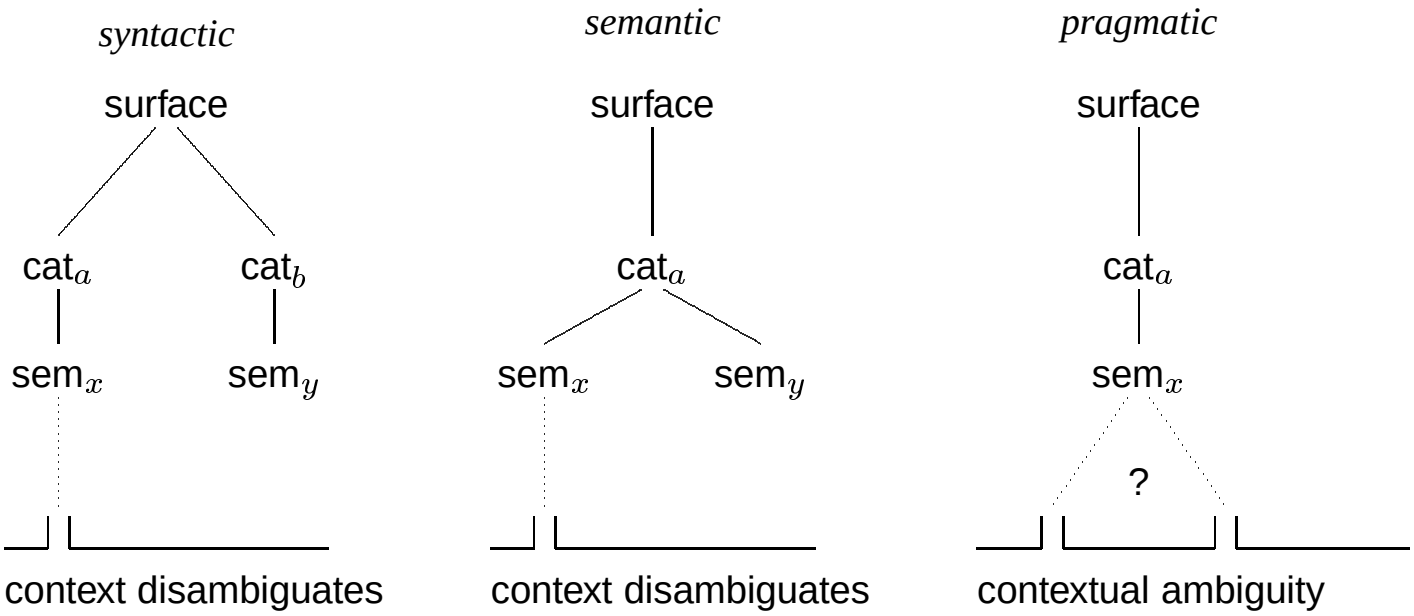
then the following two questions are equivalent:

(i) *How complex are the natural languages?*

(ii) *How ambiguous are the natural languages?*

This is because the C-LAG subclasses differ solely in their degrees of ambiguity.

12.5.3 SLIM-theoretic analysis of ambiguity

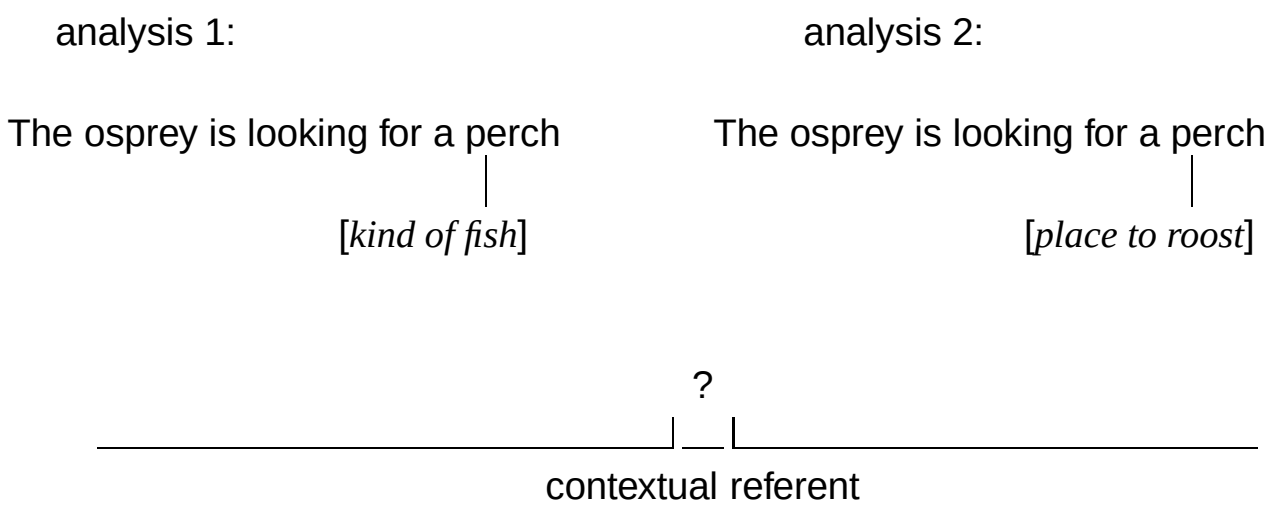


12.5.4 Multiple interpretations of prepositional phrases: a syntactic or a semantic ambiguity?

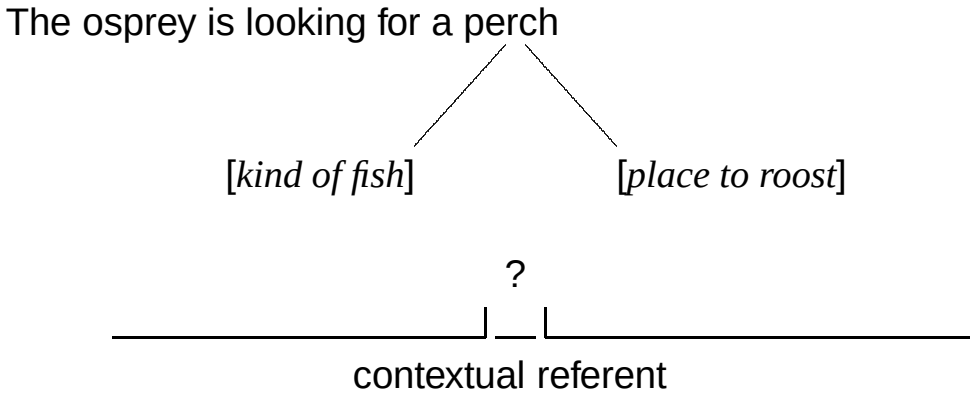
The man saw the girl with the telescope.

Julia ate the apple on the table behind the tree in the garden.

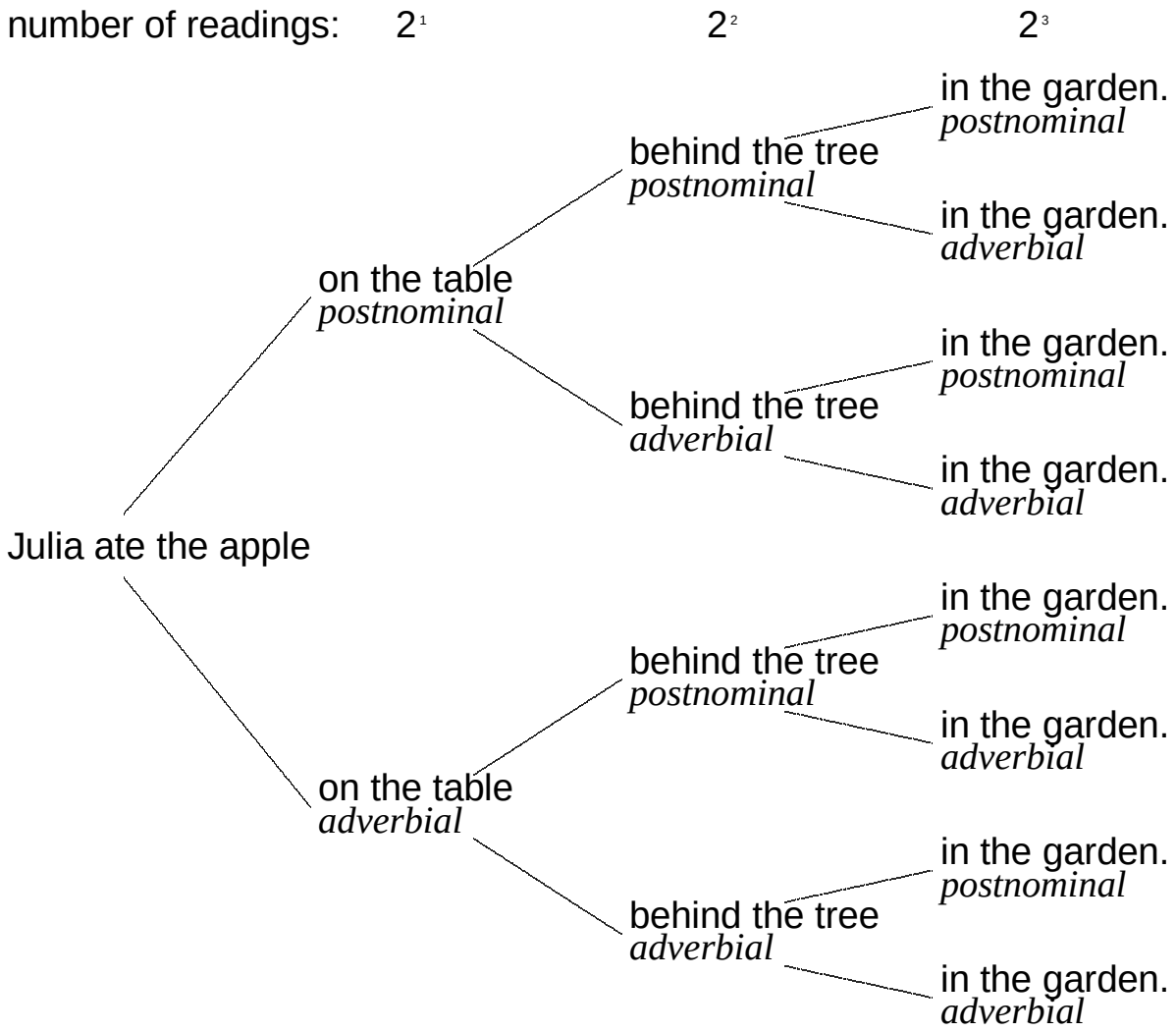
12.5.5 Incorrect analysis of a semantic ambiguity



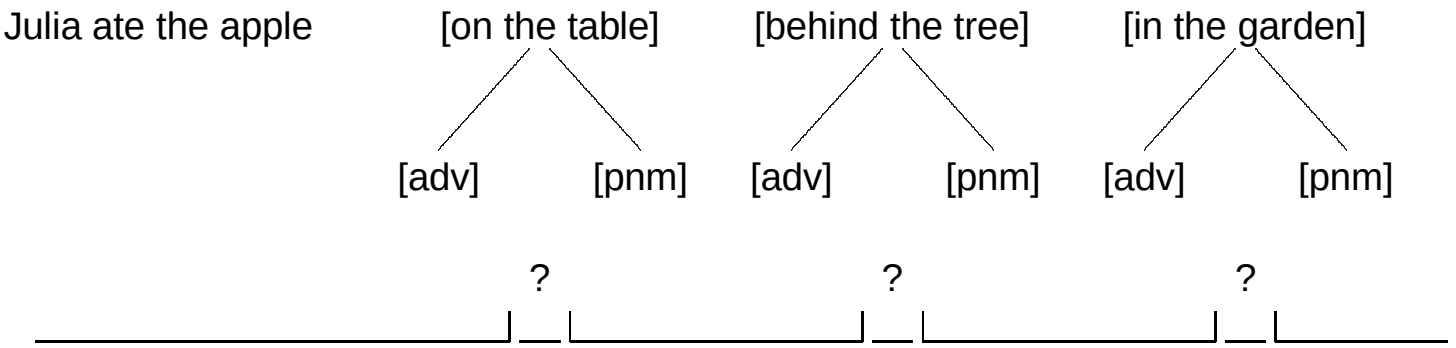
12.5.6 Correct analysis of a semantic ambiguity



12.5.7 Incorrect analysis: a recursive pseudo-ambiguity



12.5.8 Correct analysis with *semantic doubling*



12.5.9 CoNSyx hypothesis (Complexity of natural language syntax)

The natural languages are contained in the class of C1-languages and parse in linear time.

This hypothesis holds as long as no recursive ambiguities are found in natural language.