

Part II.
Formale Grammatik

7. Generative Grammatik

7.1 Sprache als Untermenge des freien Monoids

7.1.1 Definition von Sprache

Eine Sprache ist eine Menge von Wortfolgen.

7.1.2 Illustration des freien Monoids über $LX = \{a,b\}$

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 Informelle Beschreibung der künstlichen Sprache $a^k b^k$ (with $k \geq 1$)

Wohlgeformte Ausdrücke bestehen aus einer beliebigen Anzahl des Wortes a, gefolgt von der gleichen Anzahl des Wortes b.

7.1.4 Wohlgeformte Ausdrücke von $a^k b^k$

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

7.1.5 Nichtwohlgeformte Ausdrücke von $a^k b^k$

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

7.1.6 PS-grammatische Definition von $a^k b^k$

$$S \rightarrow a S b$$

$$S \rightarrow a b$$

Eine formal Grammatik kann als ein Filter aufgefaßt werden, der aus dem freien Monoid über dem Lexikon der Sprache die wohlgeformten Ausdrücke herausfiltert.

7.1.7 Elementarformalismen der generativen Grammatik

1. Kategorial- oder C-Grammatik
2. Phrasenstruktur- oder PS-Grammatik
3. Linksassoziative oder LA-Grammatik

7.1.8 Algebraische Definition

Die algebraische Definition einer generativen Grammatik (i) zählt die formalen Bausteine des Systems explizit auf und (ii) definiert die strukturellen Bezüge zwischen den Bausteinen mit den Mitteln der Mengentheorie.

7.1.9 Abgeleitete Formalismen der PS-Grammatik

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 Abgeleitete Formalismen der C-Grammatik

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

7.1.11 Beispiele semi-formaler Grammatiken

Abhängigkeitsgrammatik (Tesnière 1959), systemische Grammatik (Halliday 1985), Stratifikationsgrammatik (Lamb 1996)

7.2 Methodische Gründe für generative Grammatik

7.2.1 Grammatisch wohlgeformter Ausdruck

Die kleinen Hunde haben vorhin geschlafen

7.2.2 Grammatisch nicht wohlgeformter Ausdruck

* geschlafen vorhin haben Hunde kleinen die

7.2.3 Methodische Konsequenzen generativer Grammatik

- *Empirisch*: explizite Hypothesenbildung

Die generative Analyse resultiert in formalen Regelsystemen, die explizit festlegen, welche Ausdrücke der Theorie nach grammatikalisch wohlgeformt sind und welche nicht. Diese explizite Hypothesenbildung verschafft Klarheit über die deskriptive Adäquatheit bzw. Inadäquatheit der Analyse und ist Voraussetzung für eine schrittweise Verbesserung der empirischen Beschreibung.

- *Mathematisch*: Bestimmung der formalen Eigenschaften

Nur streng formalisierte Beschreibungen erlauben Aussagen über ihre mathematische Eigenschaften, z. B. Entscheidbarkeit, Komplexität und generative Kapazität. Die mathematischen Eigenschaften eines Grammatikformalismus bestimmen wiederum dessen Eignung für die empirische Arbeit und programmiertechnische Realisierung.

- *Programmiertechnisch*: Deklarative Spezifikation

Nur ein formales Regelsystem ist als deklarative Spezifikation eines zugehörigen Parsers geeignet. Diese stellt seine notwendigen Eigenschaften dar, im Unterschied zu seinen akzidentiellen Eigenschaften, die sich aus der Wahl der Programmierumgebung etc. ergeben. Ein Parser ermöglicht die automatische Analyse von Sprachausdrücken, die wiederum für die Verifikation von Einzelgrammatiken benötigt wird, die im Rahmen des generativen Grammatikformalismus geschrieben wurden.

7.3 Adäquatheit generativer Grammatiken

7.3.1 Desiderata einer generativen Grammatik für natürliche Sprachen

Die generative Analyse einer natürlichen Sprache sollte gleichzeitig

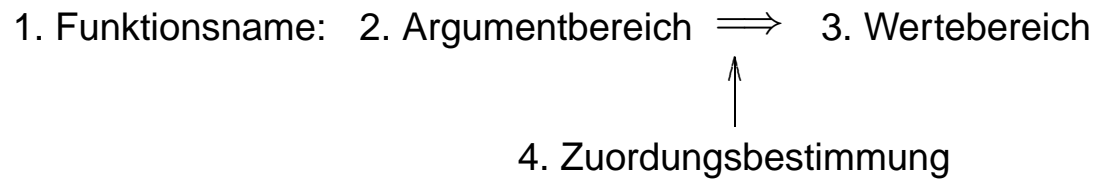
- *mathematisch* als eine formale Theorie mit niedriger Komplexität,
- *funktional* als linguistische Komponente der natürlichen Kommunikationsmechanik und
- *methodisch* als effizient implementiertes Computerprogramm definiert werden, das sowohl die Eigenschaften der formalsprachlichen Theorie als auch der empirischen Analyse natürlicher Sprachen in modularer und transparenter Form verkörpert.

7.4 Formalismus der C-Grammatik

7.4.1 Die historisch erste generative Grammatik

Die Kategorialgrammatik oder C-Grammatik wurde von den polnischen Logikern LEŚNIEWSKI 1929 und AJDUKIEWICZ 1935 mit dem Ziel entwickelt, das Russell-Paradox in der formalen Sprachanalyse zu vermeiden. Auf natürliche Sprachen wurde die C-Grammatik erstmals von BAR-HILLEL 1953 angewandt.

7.4.2 Struktur einer logischen Funktion



7.4.3 Algebraische Definition der C-Grammatik

Eine C-Grammatik ist ein Quintupel $\langle W, C, LX, R, CE \rangle$.

1. W ist eine endliche Menge von Wortformoberflächen.
2. C ist eine Menge von Kategorien der folgenden Form:
 - (a) *Anfangsbedingung*
 u und $v \in C$,
 - (b) *Induktion*
wenn X und $Y \in C$, dann sind auch (X/Y) und $(X \setminus Y) \in C$,
 - (c) *Abschlußbedingung*
Nichts ist in C außer dem, was (a) und (b) entspricht.
3. LX ist eine endliche Menge, wobei $LX \subset (W \times C)$.
4. R ist eine Menge, die die folgenden zwei Regelschemata umfaßt:
$$\alpha_{(Y/X)} \circ \beta_{(Y)} \Rightarrow \alpha\beta_{(X)}$$
$$\beta_{(Y)} \circ \alpha_{(Y \setminus X)} \Rightarrow \beta\alpha_{(X)}$$
5. CE ist die Menge der Kategorien vollständiger Ausdrücke (*complete expressions*), mit $CE \subseteq C$.

7.4.4 Rekursive Definition der unendlichen Menge C

Ausgehend von zwei Anfangselementen u und v in C gehören laut der Induktionsformel auch (u/v) , (v/u) , $(u \setminus v)$ und $(v \setminus u)$ zu C . Damit gehören aber auch $((u/v)/v)$, $((u/v) \setminus v)$, $((u/v)/u)$, $((u/v) \setminus u)$, $(u/(u/v))$, $(v/(u/v))$ etc. zu C .

7.4.5 Definition von LX als endliche Menge geordneter Paare

Jedes geordnete Paar besteht aus (i) einem Element von W und (ii) einem Element von C . Welche Oberflächen (d.h. Elemente von W) welche Elemente von C als ihre Kategorien nehmen wird in LX über explizite Auflistung der geordneten Paare bestimmt.

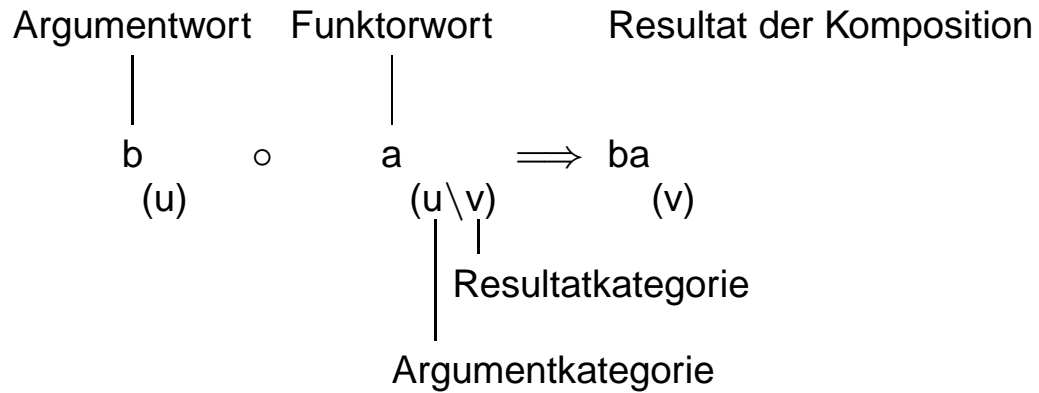
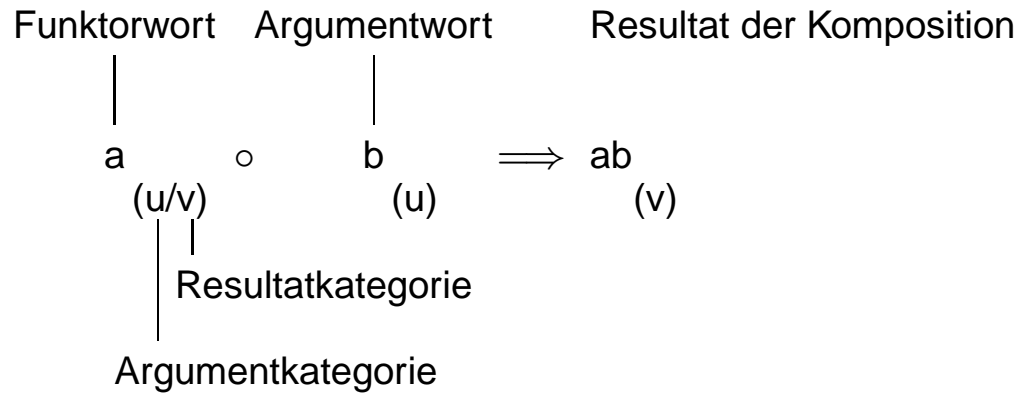
7.4.6 Definition der Menge von Regelschemata R

Die Regelschemata verwenden Variable α für die Oberfläche des Funktors, die Variable β für die Oberfläche des Arguments, und die Variablen X und Y , um deren Kategoriemuster darzustellen.

7.4.7 Definition der Menge der vollständigen Ausdrücke CE

Die Menge CE beschreibt die Kategorien derjenigen Ausdrücke, die als *vollständig* betrachtet werden. Je nach spezifischer Grammatik und Sprache kann die Menge CE endlich sein und als Aufzählung spezifiziert werden oder unendlich sein und über Muster mit Variablen charakterisiert werden.

7.4.8 Implizite Musterabgleichung C-grammatischer Kompositionen



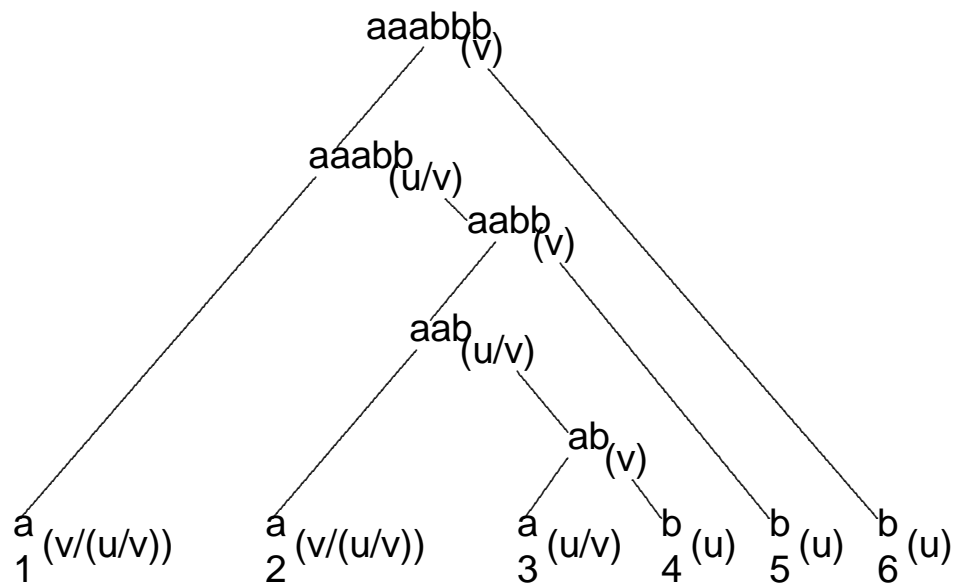
7.4.9 Kategorialgrammatische Definition von $a^k b^k$

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

$$CE =_{def} \{(v)\}$$

Das Wort a ist in 7.4.9 mit zwei verschiedenen Kategorien definiert, nämlich (u/v) und $(v/(u/v))$ – aus Gründen, die in der folgenden Ableitung sichtbar werden.

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



7.5 C-Grammatik für natürliche Sprachen

7.5.1 C-Grammatik für ein winziges Deutsch-Fragment

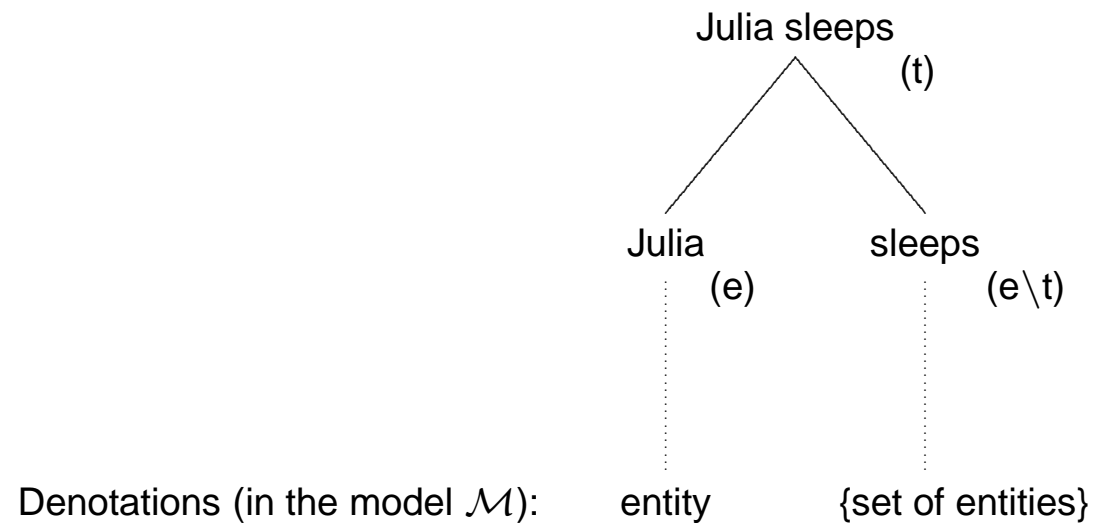
$LX =_{def} \{W_{(e)} \cup W_{(e \setminus t)}\}$, wobei

$W_{(e)} = \{\text{Julia, Peter, Maria, Fritz, Susi} \dots\}$

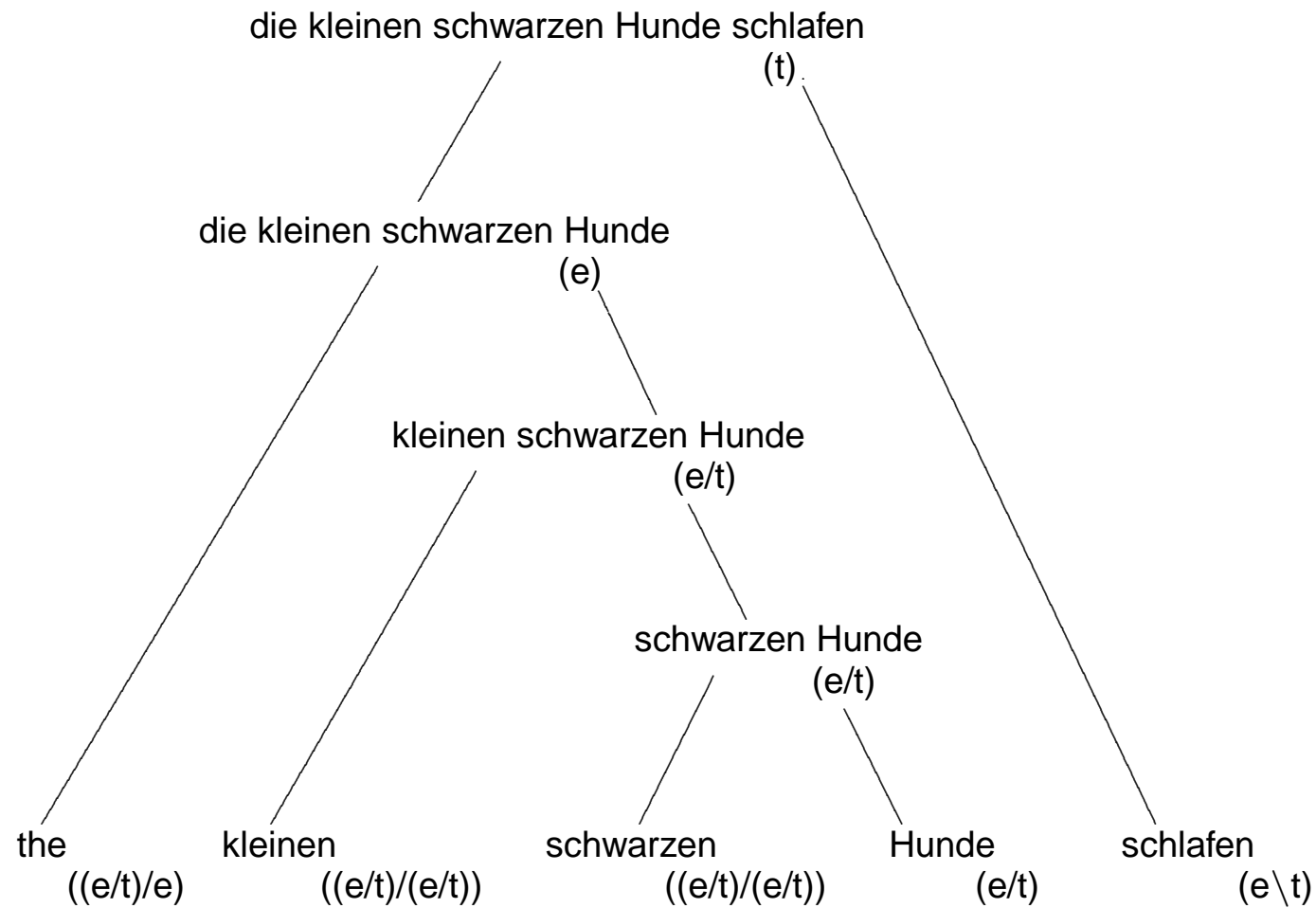
$W_{(e \setminus t)} = \{\text{schläft, lacht, singt} \dots\}$

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

7.5.2 Simultane syntaktische und semantische Analyse



7.5.3 Kategoriale Analyse eines Satzes



7.5.4 C-Grammatik für Satz 7.5.3

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

$W_{(e)} = \{ \text{Julia, Peter, Maria, Fritz, Susi ...} \}$

$W_{(e \setminus t)} = \{ \text{schläft, lacht, singt ... schlafen, lachen, singen ...} \}$

$W_{(e/t)} = \{ \text{Hund, Hunde, Katze, Katzen, Tisch, Tische ...} \}$

$W_{((e/t)/(e/t))} = \{ \text{kleine, kleinen, schwarze, schwarzen ...} \}$

$W_{((e/t)/t)} = \{ \text{der, die, den ...} \}$

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

7.5.5 Empirische Nachteile der C-Grammatik für natürliche Sprache

- Die C-grammatische Ableitung von Ausdrücken hat den Charakter von *problem solving*.
- Die Behandlung alternativer Wortstellungen und Kongruenzbeschränkungen erfordert einen extrem hohen Grad lexikalischer Ambiguitäten.

8. Sprachhierarchien und Komplexität

8.1 Formalismus der PS-grammar

8.1.1 Ursprüngliche Definition

Von dem amerikanischen Logiker E. Post 1936 als *rewrite* oder *Post production system* publiziert, ist dieser Formalismus in der Rekursionstheorie entstanden und steht in enger Beziehung zur Automatentheorie.

8.1.2 Erste Anwendung auf natürliche Sprache

Posts *rewrite systems* wurden von N. Chomsky 1957 erstmals auf die natürlichen Sprachen angewendet, und zwar als sogenannte *phrase structure grammars*.

8.1.3 Algebraische Definition der PS-Grammatik

Eine PS-Grammatik ist ein Quadrupel $\langle V, V_T, S, P \rangle$.

1. V ist eine Menge von Zeichen.
2. V_T ist eine echte Untermenge von V , genannt *terminale Zeichen*.
3. S ist ein Zeichen in V ohne V_T , genannt *Startsymbol*.
4. P ist eine Menge von Ersetzungsregeln der Form $\alpha \rightarrow \beta$, wobei α ein Element von V^+ und β ein Element von V^* ist.

8.1.4 Restriktionstypen des PS-Regelschemas

0. Unbeschränkte PS-Regel:

Bei einer Typ-0 Regel stehen auf der linken und rechten Regelseite beliebige Folgen von Terminalen und Variablen.

1. Kontextsensitive PS-Regel:

Bei einer Typ-1 Regel stehen auf der linken und rechten Regelseite beliebige Folgen von Terminalen und Variablen, wobei die rechte Regelseite mindestens so lang sein muß wie die linke.

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

2. Kontextfreie PS-Regel:

Bei einer Typ-2 Regel steht auf der linken Regelseite genau eine Variable. Auf der rechten Regelseite steht eine Zeichenkette aus V^+ .

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

3. Reguläre PS-Regel:

Bei einer Typ-3 Regel steht auf der linken Regelseite genau eine Variable. Auf der rechten Regelseite steht genau ein Terminal, gefolgt von höchstens einer Variablen.

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

8.2 Sprachklassen und ihre Komplexität

8.2.1 Verschiedene Beschränkungen der generative Regelschemata führen zu

1. unterschiedlichen *Arten von Grammatiken*, die über
2. unterschiedliche *Grade generativer Kapazität*
3. unterschiedliche *Sprachklassen* erzeugen, die wiederum
4. unterschiedliche *Komplexitätsgrade* aufweisen.

8.2.2 Grade der Komplexität

1. *Lineare Komplexität*
 $n, 2n, 3n$ etc.
2. *Polynomiale Komplexität*
 n^2, n^3, n^4 etc.
3. *Exponentielle Komplexität*
 $2^n, 3^n, 4^n$ etc.
4. *Unentscheidbar*
 $n \cdot \infty$

8.2.3 Komplexität polynomialer und exponentieller Algorithmen

	Problemgröße n		
Zeit-komplexität	10	50	100
n^3	.001 Sekunden	.125 Sekunden	1.0 Sekunden
2^n	.001 Sekunden	35.7 Jahre	10^{15} Jahrhunderte

8.2.4 Anwendung auf natürliche Sprache

Das Limaskorpus enthält insgesamt 71 148 Sätze. Von diesen bestehen genau 50 aus 100 Wortformen oder mehr, wobei der längste Satz im Korpus aus 165 Wörtern besteht.

8.2.5 PS-grammatische Hierarchie der formalen Sprachen (Chomsky-Hierarchie)

Regel Beschränkung	Unterklassen der PS-Grammatik	Sprachklassen	Komplexitätsgrad
Typ-3	reguläre PSG	reguläre Spr.	linear
Typ-2	kontextfreie PSG	kontextfreie Spr.	polynomial
Typ-1	kontextsensitive PSG	kontextsensitive Spr.	exponentiell
Typ-0	unbeschränkte PSG	rek. enumerable Spr.	unentscheidbar

8.3 Generative Kapazität und formale Sprachklassen

8.3.1 Linguistische Hauptfrage an die PS-grammatik

Gibt es einen Typ der PS-Grammatik der genau die Strukturen erzeugt, die für die natürlichen Sprachen charakteristisch sind?

8.3.2 Struktureigenschaften der regulären PS-Grammatik

Die generative Kapazität der regulären PS-Grammatik erlaubt die rekursive Wiederholung einzelner Wörter, aber ohne irgendwelche rekursive Korrespondenzen.

8.3.3 Reguläre PS-Grammatik für 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 Reguläre PS-Grammatik für $\{a, b\}^+$

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

$$V_T =_{def} \{a, b\}$$

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

8.3.5 Reguläre PS-Grammatik für $a^m b^k$ ($k, m \geq 1$)

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 Struktureigenschaften der kontextfreien PS-Grammatik

Die generative Kapazität der kontextfreien PS-Grammatik erlaubt die rekursive Erzeugung von invers-paarweisen Korrespondenzen, z. B. $a b c \dots c b a$.

8.3.7 Kontextfreie PS-Grammatik für $a^k b^{3k}$

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

$$V_T =_{def} \{a, b\}$$

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

8.3.8 Kontextfreie PS-Grammatik für 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 Warum WW die generative Kapazität der kontextfreien PS-Grammatik übersteigt

aa
 abab
 abcabc
 abcdabcd
 ...

haben keine *inverse* Struktur. Deshalb ist es trotz der Ähnlichkeit zwischen WW^R und WW unmöglich, eine kontextfreie PS-Grammatik wie 8.3.8 für WW zu schreiben.

8.3.10 Warum $a^k b^k c^k$ die generative Kapazität der kontextfreien PS-Grammatik übersteigt

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

kann nicht von einer kontextfreien PS-Grammatik generiert werden, weil Korrespondenzen zwischen *drei* verschiedenen Bereichen aufrecht erhalten werden müssen – was die *paarweis* inverse Struktur der kontextfreien Sprachen, wie sie z. B. von den Sprachen $a^k b^k$ und WW^R illustriert wird, übersteigt.

8.3.11 Struktureigenschaften der kontextsensitiven PS-Grammatik

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.

[Fast jede erdenkliche Sprache ist kontextsensitiv; die einzigen bekannten Beweise, daß bestimmte Sprachen nicht kontextsensitiv sind, beruhen letztlich auf Diagonalisierung.]

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

8.3.12 PS-Grammatik für kontextsensitives $a^k b^k c^k$

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

$$V_T =_{def} \{a, b, c\}$$

$$P =_{def} \left\{ \begin{array}{ll} S \rightarrow a S B C, & \text{rule 1} \\ S \rightarrow a b C, & \text{rule 2} \\ C B \rightarrow D_1 B, & \text{rule 3a} \\ D_1 B \rightarrow D_1 D_2, & \text{rule 3b} \\ D_1 D_2 \rightarrow B D_2, & \text{rule 3c} \\ B D_2 \rightarrow B C, & \text{rule 3d} \\ b B \rightarrow b b, & \text{rule 4} \\ b C \rightarrow b c, & \text{rule 5} \\ c C \rightarrow c c & \text{rule 6} \end{array} \right.$$

Die Regeln 3a bis 3d haben zusammen denselben Effekt wie die

Regel 3 $C B \rightarrow B C$.

8.3.13 Ableitung von $a a a b b b c c c$

	Zwischenketten	Regeln
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 Struktureigenschaften der rekursiven Sprachen

Die kontextsensitiven Sprachen sind eine echte Untermenge der rekursiven Sprachen. Die Klasse der rekursiven Sprachen kann in der PS-grammatischen Hierarchie nicht dargestellt werden. Der Grund dafür ist, daß das PS-grammatische Regelschema keinen Restriktionstyp (vgl. 8.1.4) bereithält, dessen zugehörige PS-Grammatiken genau die rekursiven Sprachen generieren würden.

Eine Sprache ist genau dann rekursiv, wenn sie entscheidbar ist, d. h., wenn es einen Algorithmus gibt, der für jede beliebige Eingabe in endlich vielen Schritten entscheiden kann, ob die Eingabe zur Sprache gehört oder nicht. Eine rekursive Sprache, die nicht kontextsensitiv ist (weil sie die generative Kapazität der kontextsensitiven Grammatiken überfordert), ist die sogenannte Ackermann-Funktion.

8.3.15 Struktureigenschaften der unbeschränkten PS-Grammatik

In unbeschränkten PS-Grammatiken kann die rechte Regelseite kürzer als die linke sein, wodurch die Möglichkeit besteht, bereits erzeugte Sequenzen wieder zu *tilgen*. Deshalb ist die Klasse der rekursiv aufzählbaren Sprachen unentscheidbar.

8.4 PS-Grammatik für natürliche Sprachen

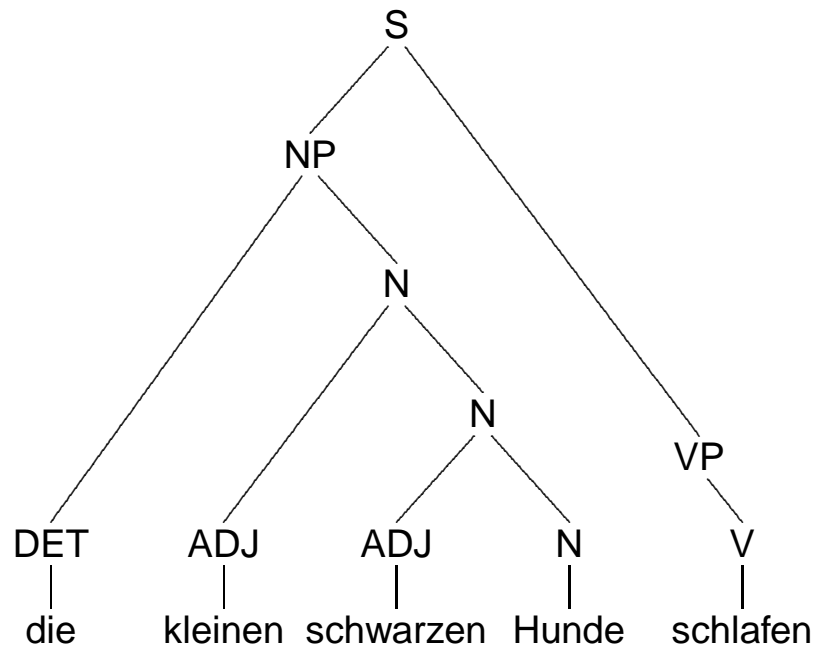
8.4.1 Eine PS-Grammatik für Beispiel 7.5.4

$V =_{def} \{S, NP, VP, V, N, DET, ADJ, \text{die}, \text{Hunde}, \text{kleinen}, \text{schlafen}, \text{schwarzen}\}$

$V_T =_{def} \{\text{die}, \text{Hunde}, \text{kleinen}, \text{schlafen}, \text{schwarzen}\}$

$P =_{def} \{$
 $S \rightarrow NP VP,$
 $VP \rightarrow V,$
 $NP \rightarrow DET N,$
 $N \rightarrow ADJ N,$
 $N \rightarrow \text{Hunde},$
 $ADJ \rightarrow \text{kleinen},$
 $ADJ \rightarrow \text{schwarzen},$
 $DET \rightarrow \text{die},$
 $V \rightarrow \text{schlafen}\}$

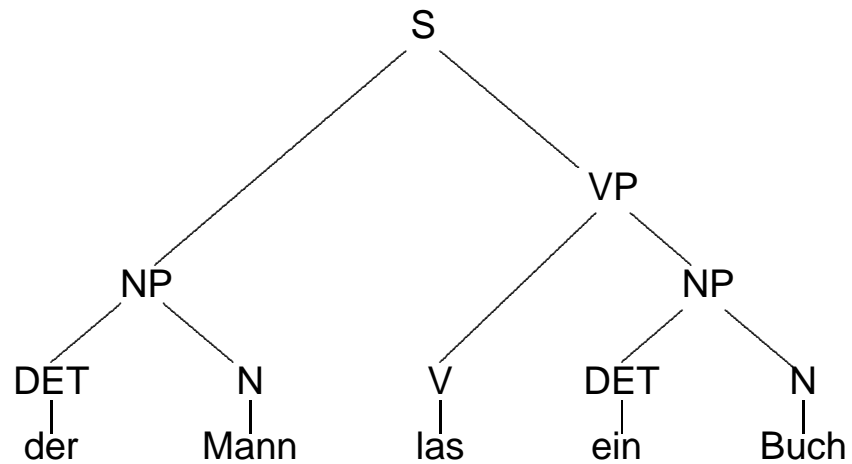
8.4.2 PS-grammatische Analyse von Beispiel 7.5.4



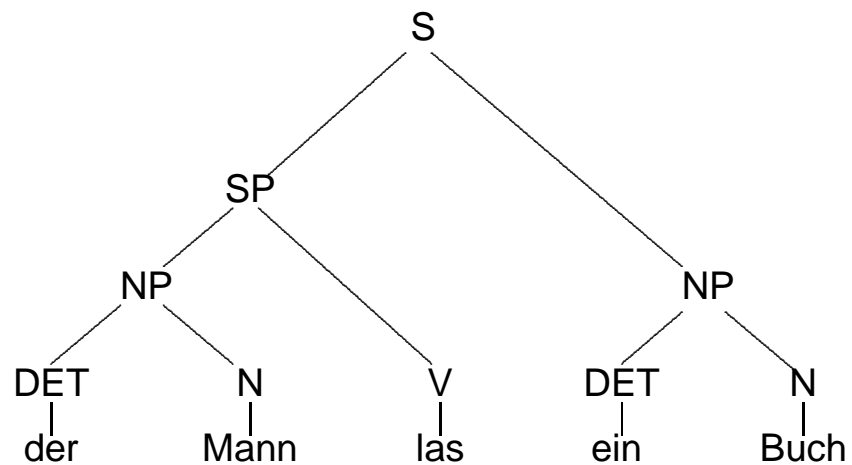
8.4.3 Definition der Konstituentenstruktur

1. Wörter oder Konstituenten, die semantisch zusammengehören, müssen direkt und exhaustiv von einem Knoten dominiert werden.
2. Die Linien einer Konstituentenstruktur dürfen sich nicht überkreuzen (*nontangling condition*).

8.4.4 Akzeptable Konstituentenstrukturanalyse



8.4.5 Nichtakzeptable Konstituentenstrukturanalyse

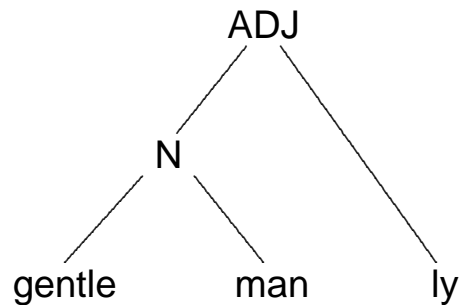


8.4.6 Ursprung der Konstituentenstruktur

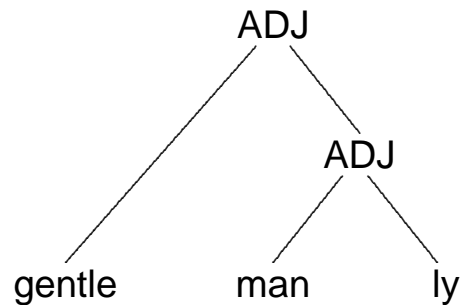
Historisch hat sich der Begriff der Konstituentenstruktur aus der *immediate constituent analysis* des amerikanischen Strukturalisten L. BLOOMFIELD (1887–1949) und den Distributionstests seines Schülers Z. Harris entwickelt.

8.4.7 Immediate constituents in PS-grammar:

Korrekt:



Falsch:



8.4.8 Substitutionsprobe

Akzeptable Substitution:

Susanne liest [ein gutes Buch]

↓

[eine dicke Zeitung]

Nicht-akzeptable Substitution:

Susanne liest ein [gutes Buch]

↓

*Susanne liest ein [dicke Zeitung]

8.4.9 Bewegungsprobe

Akzeptable Bewegung:

[der kleine Hund] sieht Julia \implies sieht [der kleine Hund] Julia (?)

Nicht-akzeptable Bewegung:

der [kleine Hund] sieht Julia \implies *der sieht [kleine Hund] Julia

8.4.10 Beabsichtigter Zweck der Konstituentenstruktur

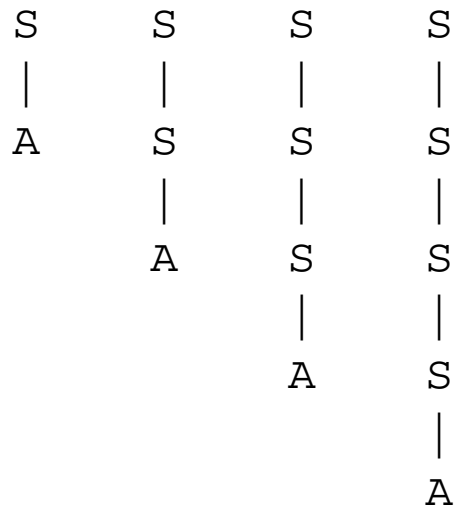
Distributionstests erschienen den amerikanischen Strukturalisten methodisch wichtig zu sein, um ihre Intuitionen über die korrekte Zerlegung (Segmentierung) von Sätzen zu objektivieren. Die Unterscheidung zwischen linguistisch wohlmotivierten und unakzeptablen *immediate-constituent*-Analysen schien wiederum notwendig, weil jeder endlichen terminalen Kette (also jeder Sequenz von Wortformen) *unendlich* viele verschiedene Baumstrukturen zugrunde gelegt werden können.

8.4.11 Unendliche Anzahl von Bäumen über einem einzigen Wort

Kontextfreie Regeln: $S \rightarrow S$, $S \rightarrow A$

Indizierte Klammerung: $(A)_S$, $((A)_S)_S$, $((((A)_S)_S)_S)$, $(((((A)_S)_S)_S)_S)$, etc.

Entsprechende Bäume:

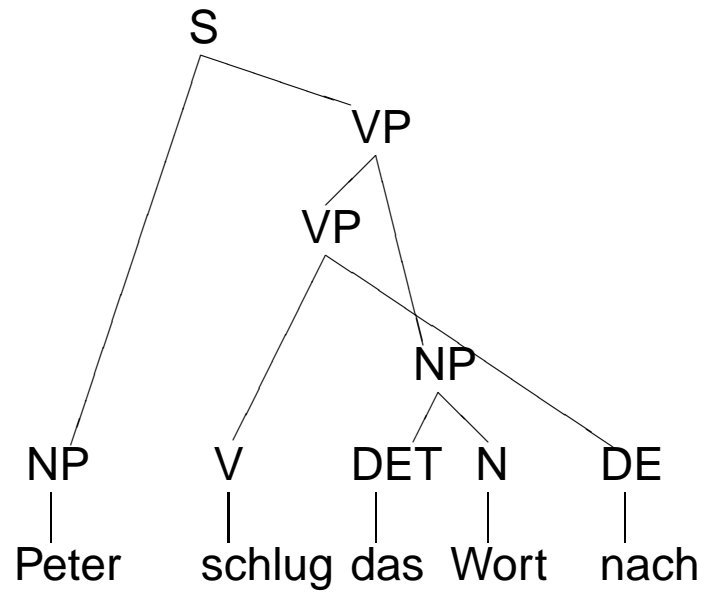


8.5 Konstituentenstrukturparadox

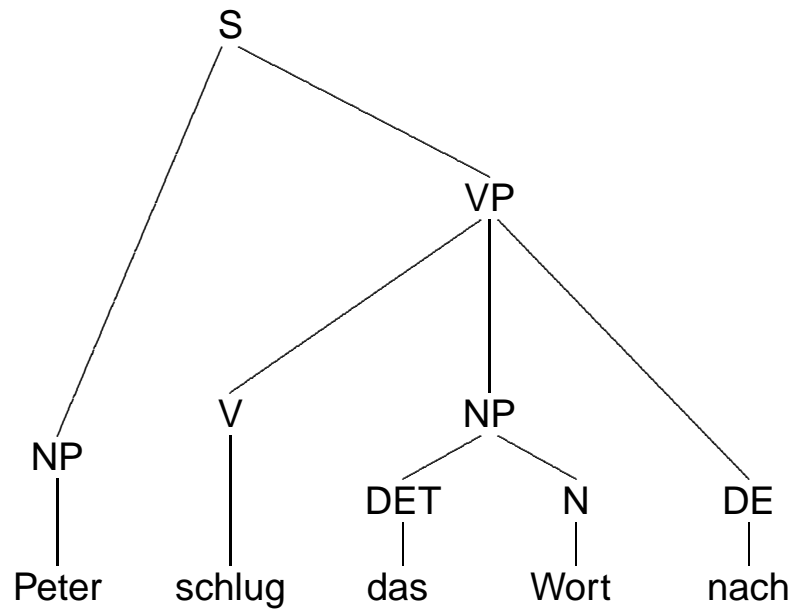
8.5.1 Konstituentenstrukturen aus der Sicht der SLIM-Sprachtheorie

- Konstituentenstrukturen und die Distributionstests, die ihnen zugrundegelegt werden, widersprechen der zeitlinearen Struktur der natürlichen Sprachen.
- Die aus der Konstituentenstrukturanalyse resultierenden Phrasenstrukturbäume haben keinerlei kommunikative Funktion.
- Die Prinzipien der Konstituentenstruktur können bei der empirischen Analyse natürlicher Sprachen nicht immer erfüllt werden.

8.5.2 Verletzung der zweiten Bedingung



8.5.3 Verletzung der ersten Bedingung



8.5.4 Annahmen der Transformationsgrammatik

Um die Konstituentenstruktur als angeboren zu erhalten, unterscheidet die Transformationsgrammatik zwischen hypothetischen Tiefenstrukturen, die angeblich universal sind, und den konkreten sprachabhängigen Oberflächenstrukturen. Dabei wird angenommen,

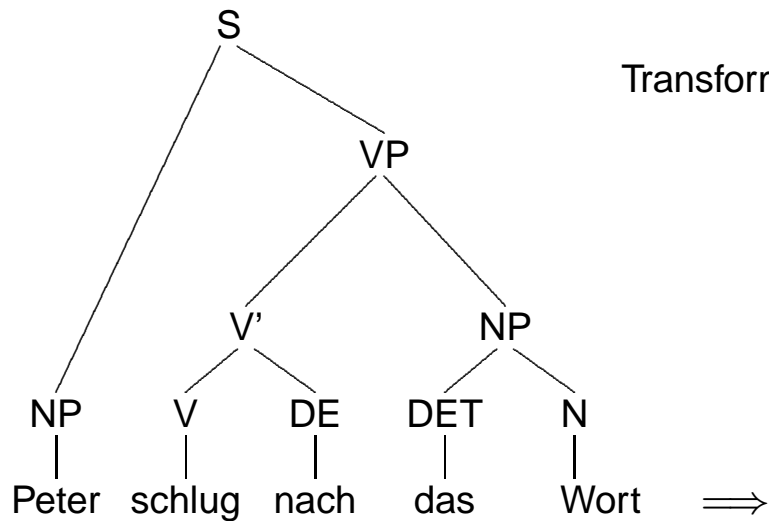
- daß die beiden Ebenen semantisch äquivalent sind,
- daß die Tiefenstrukturen nicht grammatisch wohlgeformt sein müssen, aber die Bedingungen der Konstituentenstruktur erfüllen müssen, und
- daß die Oberflächenstrukturen grammatisch sein müssen, aber nicht die Bedingungen der Konstituentenstruktur erfüllen müssen.

8.5.5 Beispiel einer formalen Transformation

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

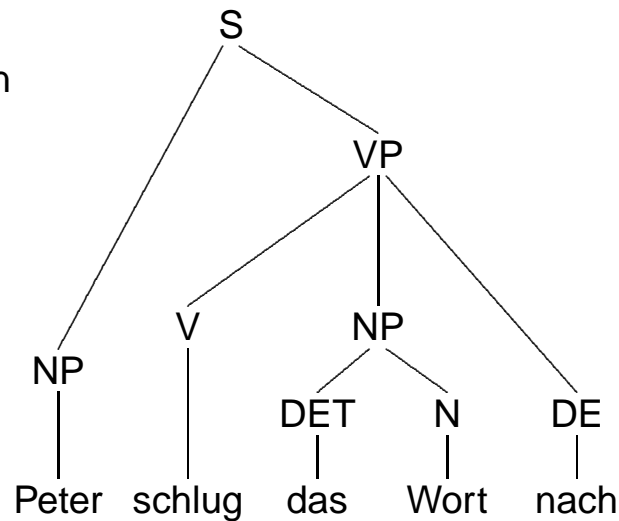
8.5.6 Anwendung der Transformation 8.5.5

Tiefenstruktur:



Transformation

Oberflächenstruktur:



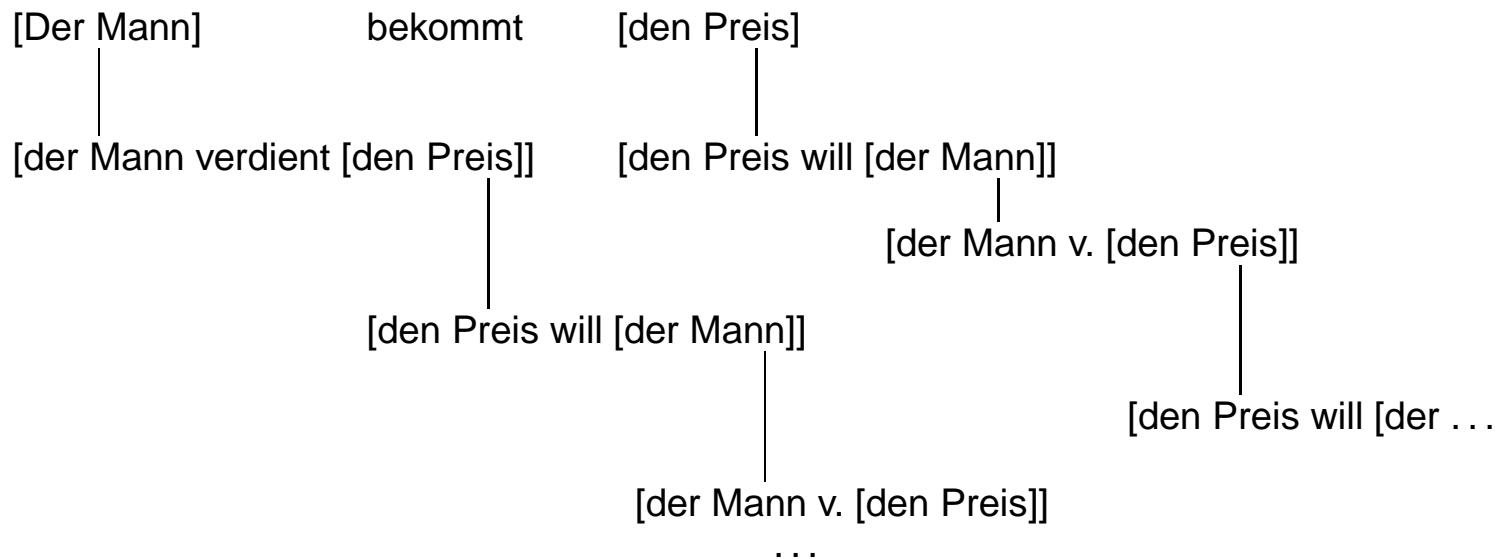
8.5.7 Mathematische Folgen aus dem Einbau von Transformationen in die PS-Grammatik

Während die kontextfreie Tiefenstruktur von niedriger polynomialer Komplexität ist (n^3), hebt der Einbau von Transformationen die Komplexität zu rekursiv aufzählbar. M.a.W., die Transformationsgrammatik ist unentscheidbar.

8.5.8 Beispiel eines Bach-Peters-Satzes

Der Mann, der ihn verdient, bekommt den Preis, den er will.

8.5.9 Tiefenstruktur eines Bach-Peters-Satzes



9. Grundbegriffe des Parsens

9.1 Deklarative und prozedurale Aspekte des Parsens

9.1.1 Deklarative und prozedurale Aspekte der Sprachanalyse

- Der *deklarative* Aspekt einer computerlinguistischen Analyse wird durch die generative Grammatik repräsentiert, die für die zu analysierende Sprache im Rahmen eines mathematisch wohldefinierten Formalismus geschrieben wurde.
- Der *prozedurale* Aspekt einer computerlinguistischen Analyse umfaßt alle Teilbereiche der Programmierung, die den generativen Formalismus bei der automatischen Analyse sprachlicher Eingaben umsetzen und verwenden.

9.1.2 Unterspezifikation der deklarative Definition bzgl. Ableitungsordnung

Regel 1: $A \rightarrow B C$

Regel 2: $B \rightarrow c d$

Regel 3: $C \rightarrow e f$

9.2 Anpassen von Grammatik auf Sprache

9.2.1 Eine kontextfreie Struktur im Deutschen

Der Mann, schläft.
der die Frau, liebt,
die das Kind, sieht,
das die Katze füttert,

9.2.2 Kontextsensitive Struktur im Schweizerdeutsch

mer em Hans es huus hälfed aastriiche
wir dem Hans das Haus helfen anstreichen

9.2.3 Mögliche Schlüsse aus der Annahme, daß die natürlichen Sprachen nicht kontextfrei sind

1. Die PS-Grammatik ist der einzige Elementarformalismus der generativen Grammatik, weshalb man sich damit abfinden muß, daß die natürlichen Sprachen von einer hohen mathematischen Komplexität (und daher *computationally intractible*) sind.
2. Die PS-Grammatik ist nicht der einzige Elementarformalismus. Vielmehr gibt es andere generative Grammatiken, die alternative Sprachhierarchien definieren, die quer (orthogonal) zur PS-Hierarchie liegen.

9.2.4 Mögliche Beziehungen zwischen zwei Formalismen

- *Keine Äquivalenz*

Zwei Grammatikformalisten sind nicht äquivalent, wenn sie unterschiedliche Sprachklassen erzeugen und erkennen; das heißt, die beiden Formalismen sind von unterschiedlicher generativer Kapazität.

- *Schwache Äquivalenz*

Zwei Grammatikformalisten sind schwach äquivalent, wenn sie dieselben Sprachen erzeugen und erkennen; das heißt, die beiden Formalismen haben dieselbe generative Kapazität.

- *Starke Äquivalenz*

Zwei Grammatikformalisten sind stark äquivalent, wenn sie (i) schwach äquivalent sind und (ii) darüber hinaus dieselben Strukturbeschreibungen liefern; das heißt, die beiden Formalismen sind lediglich *notationelle Varianten*.

9.2.5 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 Categorical Grammar *à 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].

[Es stellte sich das Problem, die genaue Beziehung zwischen diesen Typen der [PS-]Grammatik und der C-Grammatik zu bestimmen. Ich vermutete 1958, daß die bidirektionale C-Grammatik [siehe Definition 7.4.1] ungefähr dieselbe generative Kapazität hat [wie die kontextfreie PS-Grammatik]. Der Beweis ihrer Äquivalenz wurde im Juni 1959 von Gaifman gefunden. ... Die Äquivalenz dieser beiden Grammatiktypen sollte nicht zu sehr überraschen. Jeder von ihnen war mit der Absicht entwickelt worden, den Begriff *immediate constituent grammars* präzise zu explizieren. Dieser Begriff hat viele Jahre lang als die favorisierte Form der deskriptiven Linguistik in Amerika gedient, wie die bekannten Bücher von Harris 1951 und Hockett 1958 zeigen.]

Y. Bar-Hillel 1960 [1964, p. 103]

9.2.6 Allgemeinen Beziehungen zwischen den Begriffen

Sprache, generative Grammatik, Untertypen von Grammatiken, Sprachklassen, Parser und Komplexität

- *Sprachen* existieren unabhängig von generativen Grammatiken. Eine Sprache kann von verschiedenen Grammatiken aus verschiedenen Formalismen beschrieben werden.
- Eine *generative Grammatik* ist einerseits ein allgemeiner formaler Rahmen, andererseits ein spezifisches Regelsystem, das innerhalb des allgemeinen Rahmens für eine bestimmte Sprache geschrieben wurde.
- *Untertypen von generativen Grammatiken* werden über verschiedene Beschränkungen des verwendeten Grammatiktyps (insbesondere seiner Regelstruktur) definiert.
- *Sprachklassen* entstehen aus den Untertypen eines generativen Grammatikformalismus.
Nota bene: *Sprachen* existieren unabhängig von den tatsächlichen oder möglichen formalen Grammatiken, die sie generieren. *Sprachklassen* werden dagegen über spezifische Beschränkungen spezifischer Grammatikformalismen definiert.
- *Parser* sind automatische Analyseprogramme, die für einen ganzen Untertyp einer generativen Grammatik funktionieren.
- Die *Komplexität* eines Grammatiktyps wird über die Zahl der Rechenschritte (*primitive operations*) bestimmt, die schlimmstenfalls von äquivalenten abstrakten Automaten oder Parsingprogrammen bei der Analyse zugehöriger Sprachausdrücke benötigt wird.

9.3 Typentransparenz zwischen Grammatik und Parser

9.3.1 Die natürliche Auffassung eines Parsers als Motor einer Grammatik

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*.

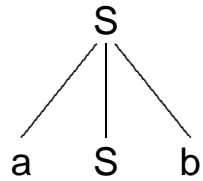
[Der ursprüngliche Vorschlag von Miller und Chomsky (1963) war eigentlich, daß Grammatiken mehr oder weniger direkt als Parsingalgorithmen zu realisieren seien. Wir können dies zu einem methodischen Prinzip machen. Das heißt, wir verlangen, daß die logische Organisation der Regeln und die Struktur der Grammatik in der Organisation des Parsingmechanismus sehr genau wiedergespiegelt wird. Dieses methodische Prinzip nennen wir *Typentransparenz*.]

R.C. Berwick & A.S. Weinberg 1984, p. 39.

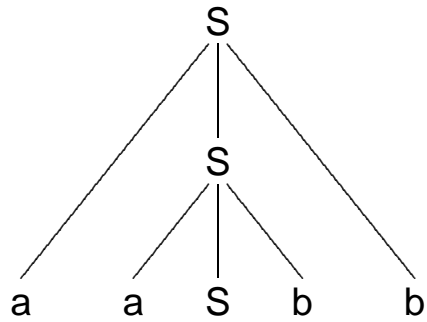
9.3.2 Definition der absoluten Typentransparenz

- Für eine gegebene Sprache verwenden Parser und Generator dieselbe formale Grammatik,
- wobei Parser und Generator die Regeln der Grammatik direkt anwenden.
- Das heißt insbesondere, daß Parser und Generator die Regeln in derselben Reihenfolge anwenden wie die grammatische Ableitung,
- daß Parser und Generator bei jeder Regelanwendung dieselben Eingaben nehmen wie die Grammatik und
- daß Parser und Generator bei jeder Regelanwendung dieselben Ausgaben ergeben wie die Grammatik.

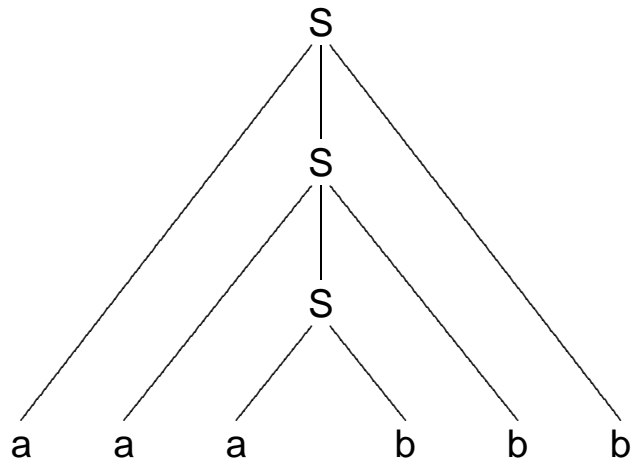
9.3.3 Top-down Ableitungsstruktur von a a a b b b



$S \longrightarrow a S b$ Schritt 1

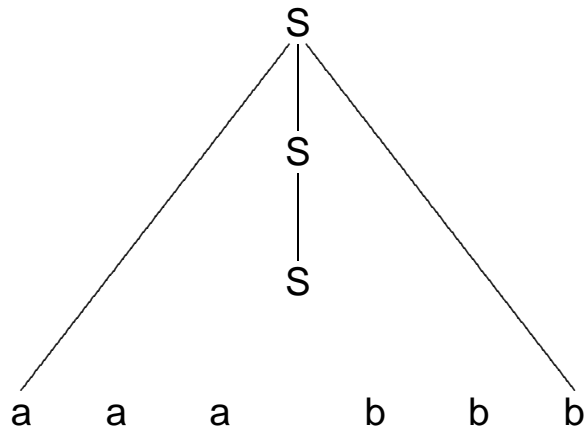


$S \longrightarrow a S b$ Schritt 2

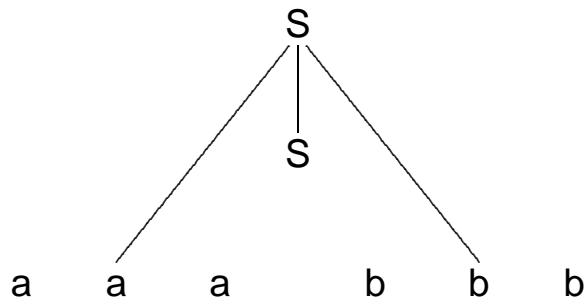


$S \longrightarrow a b$ Schritt 3

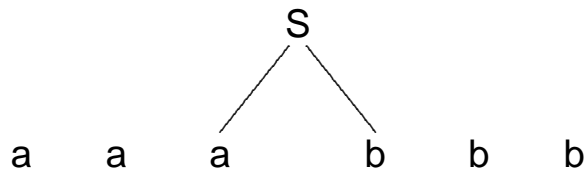
9.3.4 Bottom-up Ableitungsstruktur von a a a b b b



$S \leftarrow a S b$ Schritt 3



$S \leftarrow a S b$ Schritt 2



$S \leftarrow a b$ Schritt 1

9.3.5 Der Earley-Algorithmus am Beispiel von $a^k b^k$

.aaabbb

.S

| a.aabbb

.ab -> a.b

.aSb -> a.Sb

| aa.aabbb

a.abbb -> aa.bb

a.aSbbb -> aa.Sbbb

| aaa.bbb aaab.bb

aa.aabbb -> aaa.bbb -> aaab.bb -> ...

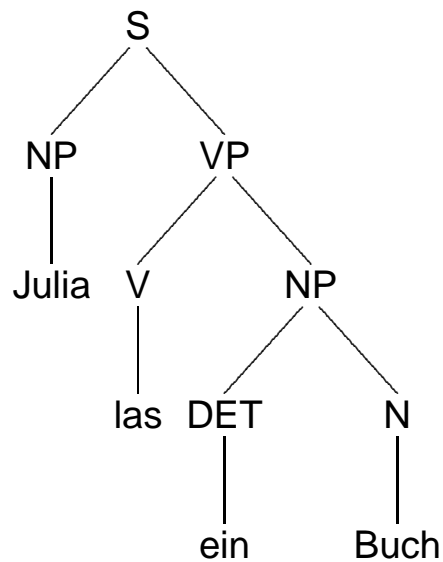
aa.aSbbb -> aaa.Sbbb

9.4 Ein-Ausgabeäquivalenz mit dem Sprecher-Hörer

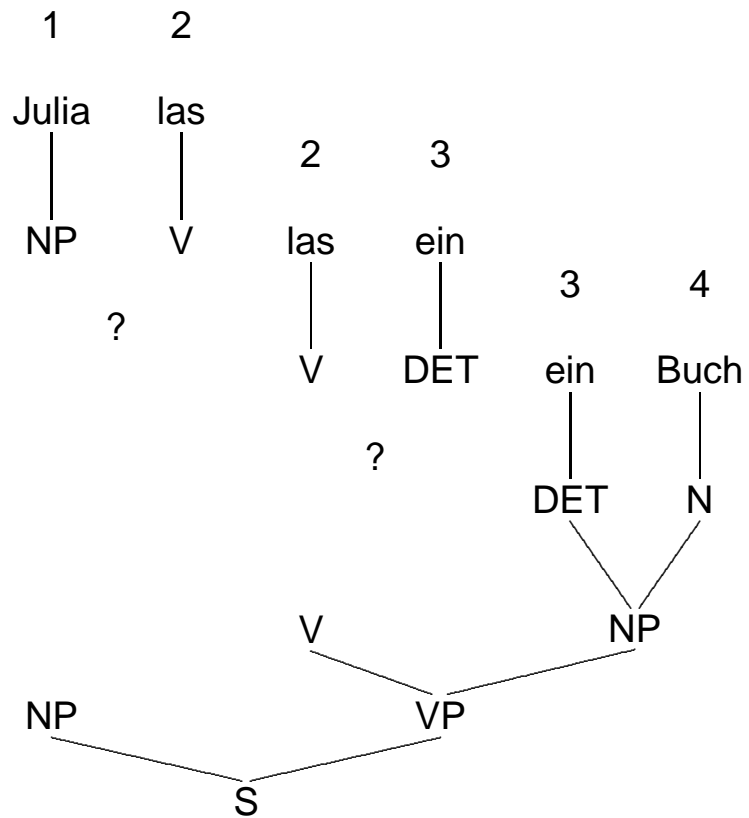
9.4.1 Eine kontextfreie PS-Grammatik

- | | | | |
|-------|---------|--------|--------|
| 1. S | → NP VP | 5. V | → las |
| 2. NP | → DET N | 6. DET | → ein |
| 3. VP | → V NP | 7. N | → Buch |
| 4. NP | → Julia | | |

9.4.2 PS-grammatische Analyse (*top-down-Ableitung*)



9.4.3 Zeitlinearer Analyse-Versuch in der PS-Grammatik



9.5 Konvergenz-Desiderata an einen Grammatikformalismus

9.5.1 Symptome fehlender Konvergenz im Nativismus

- Statt einer Konsolidierung des ursprünglichen Elementarsystems werden ständig neue abgeleitete Systeme entwickelt.
- Die als notwendig erachtete Einführung zusätzlicher Mechanismen (Transformationen, Metaregeln etc.) führt ausnahmslos zu einer Verschlechterung der mathematischen und programmiertechnischen Eigenschaften des ursprünglichen Formalismus kontextfreier PS-Grammatik.
- Die empirische Beschreibung natürlicher Sprache führt ständig zu Problemen vom Typ ‘deskriptive Aporie’ und ‘Qual der Wahl’.
- Die theoretischen Konstrukte des Nativismus werden von praktischen Systemen der Sprachverarbeitung höchstens mit Lippenbekenntnissen beachtet, meist aber ganz ignoriert.

9.5.2 Ursachen der fehlenden Konvergenz im Nativismus

- Der Nativismus ist empirisch unterspezifiziert, weil er keine funktionale Sprachtheorie enthält.
- Der vom Nativismus verwendete Formalismus der PS-Grammatik ist mit den Ein-Ausgabebedingungen des Sprecher-Hörers nicht kompatibel.

9.5.3 Eigenschaften der PS-Grammatik

- *Mathematisch:*

Praktisch verwendbare Parsingalgorithmen existieren nur für die kontextfreie PS-Grammatik. Diese ist zwar von ausreichend niedriger Komplexität (n^3), aber nicht von ausreichend hoher generativer Kapazität für die natürlichen Sprachen. Erweiterungen der generativen Kapazität sind dagegen mathematisch so komplex (unentscheidbar oder exponentiell), daß es für sie keine praktisch verwendbaren Parsingalgorithmen geben kann.

- *Programmiertechnisch:*

Die PS-Grammatik ist nicht typentransparent. Dies erschwert die Fehlersuche und Erweiterung von Grammatiken mit automatisch erstellten Parsingprotokollen. Außerdem erfordert die indirekte Beziehung zwischen Grammatik und Parsingalgorithmus den Einsatz aufwendiger Routinen und umfangreicher Zwischenstrukturen.

- *Empirisch:*

Die substitutionsbasierte Ableitungsordnung der PS-Grammatik widerspricht der zeitlinearen Grundstruktur natürlicher Sprachen und ist mit den Bedingungen der natürlichen Kommunikation nicht in Einklang zu bringen.

9.5.4 Desiderata an generative Grammatikformalismen

1. Der Formalismus sollte mathematisch einwandfrei definiert sein und somit
2. eine *deklarative* Spezifikation der Strukturen in natürlichen und künstlichen Sprachen erlauben.
3. Dabei sollte der Formalismus *rekursiv* (und damit entscheidbar)
4. sowie *typentransparent* in bezug auf seine Parser und Generatoren sein.
5. Der Formalismus sollte über strukturell offensichtliche Einschränkungen des Regelapparats eine *Hierarchie verschiedener Sprachklassen* definieren, analog – aber orthogonal – zur PS-grammatischen Hierarchie,
6. wobei diese Hierarchie eine Sprachklasse von niedriger, möglichst linearer Komplexität enthält, deren *generative Kapazität* für die Analyse der natürlichen Sprachen ausreicht.
7. Der Formalismus sollte *ein-ausgabeäquivalent* mit dem Sprecher-Hörer sein und somit eine zeitlineare Ableitungsordnung verwenden.
8. Der Formalismus sollte für *Produktion* (im Sinne einer Abbildung von Bedeutungen auf Oberflächen) und *Interpretation* (im Sinne einer Abbildung von Oberflächen auf Bedeutungen) gleichermaßen geeignet sein.

10. Linksassoziative Grammatik (LAG)

10.1 Regeltypen und Ableitungsordnung

10.1.1 Der Begriff *linksassoziativ*

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 $(a + (b + c))$.

[Wenn wir Operatoren kombinieren, um Ausdrücke zu bilden, ist die Ordnung, in der die Operatoren anzuwenden sind, nicht immer offensichtlich. Zum Beispiel kann $a + b + c$ interpretiert werden als $((a + b) + c)$ oder als $(a + (b + c))$. Wir sagen, daß $+$ linksassoziativ ist, wenn die Teilausdrücke von links nach rechts in der Form der folgenden Klammerung erscheinen: $((a + b) + c)$. Wir nennen einen Operator rechtsassoziativ, wenn er die Teilausdrücke in der umgekehrten Richtung gruppiert: $(a + (b + c))$.]

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

10.1.2 Inkrementelle links- oder rechts-assoziative Ableitung

linksassoziativ:

$$\begin{array}{l}
 a \\
 (a + b) \\
 ((a + b) + c) \\
 (((a + b) + c) + d) \\
 \dots \\
 \implies
 \end{array}$$

rechtsassoziativ:

$$\begin{array}{l}
 a \\
 (b + a) \\
 (c + (b + a)) \\
 (d + (c + (b + a))) \\
 \dots \\
 \impliedby
 \end{array}$$

10.1.3 Linksassoziative Ableitungsordnung

Ableitung basiert auf dem Prinzip der möglichen *Fortsetzungen*. Modelliert die zeitlineare Struktur der Sprache.

10.1.4 Unregelmäßige Klammerstrukturen, die den Bäumen der C- und PS-Grammatik entsprechen

$$\begin{array}{l}
 (((a + b) + (c + d)) + e) \\
 ((a + b) + ((c + d) + e)) \\
 (a + ((b + c) + (d + e))) \\
 ((a + (b + c)) + (d + e)) \\
 (((a + b) + c) + (d + e)) \\
 \dots
 \end{array}$$

Die Zahl der unregelmäßigen Klammerstrukturen wächst exponentiell mit der Länge der Kette – oder ist bei Klammerungen wie (a), ((a)), (((a))) etc. sogar unendlich.

10.1.5 Unregelmäßige Klammerstrukturen

Ableitung basiert auf dem Prinzip der möglichen *Substitutionen*. Modelliert Konstituentenstrukturen.

10.1.6 Das Prinzip der möglichen Fortsetzungen

Beginnend mit dem ersten Wort des Satzes beschreibt die LAGrammatik für jeden wohlgeformten Satzanfang die möglichen Fortsetzungen, indem sie die Regeln angibt, die für die jeweils nächste Komposition (d. h. das Anhängen des nächsten Wortes) grammatisch möglich sind.

10.1.7 Schema of left-associative rule in LA-grammar

$$r_i: \text{cat}_1 \text{ cat}_2 \Rightarrow \text{cat}_3 \text{ rp}_i$$

10.1.8 Schema of a canceling rule in C-grammar

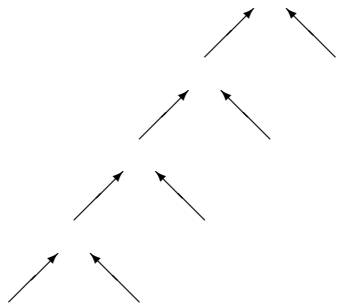
$$\alpha_{(Y|X)} \circ \beta_{(Y)} \Rightarrow \alpha\beta_{(X)}$$

10.1.9 Schema of a rewrite rule in PS-grammar

$$A \rightarrow B C$$

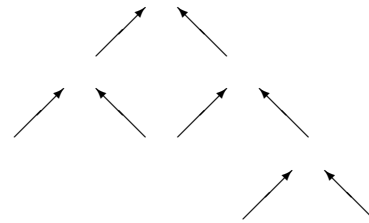
10.1.10 Three conceptual derivation orders

LA-Grammatik



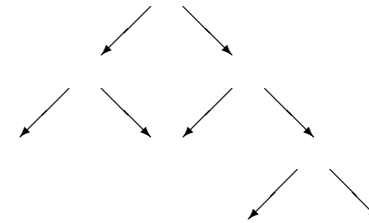
bot.-up left-associative

C-Grammatik



bottom-up amalgamating

PS-Grammatik



top-down expanding

10.2 Formalismus der LA-Grammatik

10.2.1 Algebraische Definition der LA-Grammatik

Eine LA-Grammatik ist als 7-Tupel $\langle W, C, LX, CO, RP, ST_S, ST_F \rangle$ definiert.

Dabei gilt

1. W ist eine endliche Menge von *Wortformoberflächen*.
2. C ist eine endliche Menge von *Kategoriesegmenten*.
3. $LX \subset (W \times C^+)$ ist eine endliche Menge, die das *Lexikon* umfaßt.
4. $CO = (co_0 \dots co_{n-1})$ ist eine endliche Sequenz aus total rekursiven Funktionen aus $(C^* \times C^+)$ in $C^* \cup \{\perp\}$, genannt *kategoriale Operationen*.
5. $RP = (rp_0 \dots rp_{n-1})$ ist eine ebenso lange Sequenz von Untermengen von n , genannt *Regelpakete*.
6. $ST_S = \{(cat_s \ rp_s), \dots\}$ ist eine endliche Menge von *Anfangszuständen*, wobei jedes rp_s , genannt *Anfangsregelpaket*, eine Untergruppe von n ist und jedes $cat_s \in C^+$.
7. $ST_F = \{(cat_f \ rp_f), \dots\}$ ist eine endliche Menge von *Endzuständen*, wobei jedes $cat_f \in C^*$ und jedes $rp_f \in RP$.

10.2.2 Eine konkrete LA-Grammatik wird spezifiziert durch

1. ein Lexikon LX (siehe 3),
2. eine Menge von Anfangszuständen ST_S (siehe 6),
3. eine Sequenz von Regeln r_i , definiert als (co_i, rp_i) , (siehe 4 und 5) und
4. eine Menge von Endzuständen ST_F (siehe 7)

10.2.3 LA-Grammatik für $a^k b^k$

$$LX =_{def} \{[a(a)], [b(b)]\}$$

$$ST_S =_{def} \{[(a) \{r_1, r_2\}]\}$$

$$r_1: (X) (a) \Rightarrow (aX) \{r_1, r_2\}$$

$$r_2: (aX) (b) \Rightarrow (X) \{r_2\}$$

$$ST_F =_{def} \{[\varepsilon rp_2]\}.$$

10.2.4 LA-Grammatik für $a^k b^k c^k$

$LX =_{def} \{[a(a)], [b(b)], [c(c)]\}$

$ST_S =_{def} \{[(a) \{r_1, r_2\}]\}$

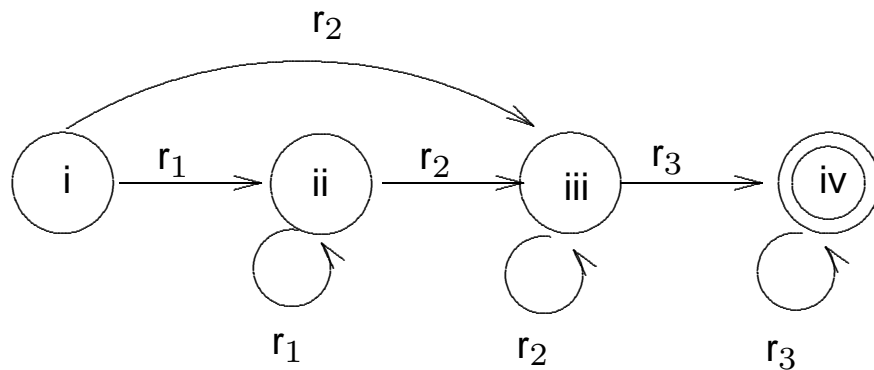
$r_1: (X) (a) \Rightarrow (aX) \{r_1, r_2\}$

$r_2: (aX) (b) \Rightarrow (Xb) \{r_2, r_3\}$

$r_3: (bX) (c) \Rightarrow (X) \{r_3\}$

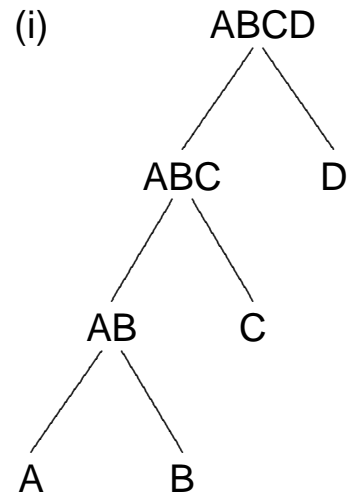
$ST_F =_{def} \{[\varepsilon rp_3]\}$.

10.2.5 Das *finite-state*-Grundgerüst von LA- $a^k b^k c^k$



10.3 Zeitlineare Analyse

10.3.1 LA-Bäume als strukturierte Listen



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

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

10.3.2 LA-grammatische Ableitung von $a^k b^k$ 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 einer ‘history section’

aktives Regelpaket:	*START-0
Kompositionsnummer:	1
Satzanfang:	(A) A
nächstes Wort:	(A) A
Erfolgreiche Regel	*RULE-1
nächste Kompositionsnummer:	2
Ergebnis:	(A A) A A

10.3.4 Überlappung zwischen ‘history sections’

aktives Regelpaket:	*RULE-1
Kompositionsnummer:	2
Satzanfang:	(A A) A A
nächstes Wort:	(A) A
Erfolgreiche Regel:	*RULE-1
nächste Kompositionsnummer:	3
Ergebnis:	(A A A) A A A

10.4 Absolute Typentransparenz der LA-Grammatik

10.4.1 Parsen von aaabbbccc mit aktivem Regelzähler

```

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

*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

```

10.4.2 Erzeugung ‘Charakteristischer Beispiele’ 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:

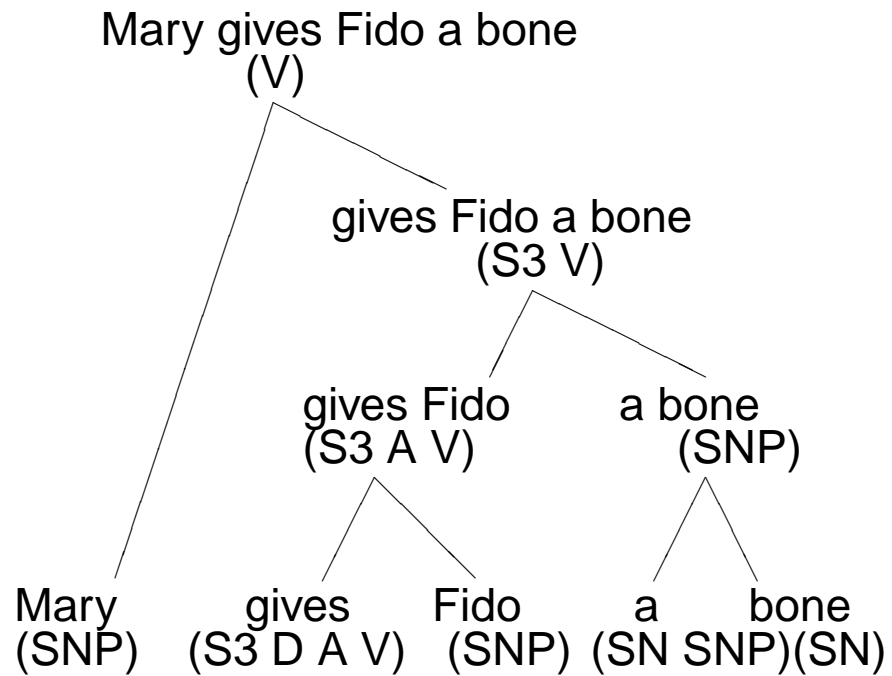
A A A A B B B B C C C C
 1 1 1 2 2 2 2 3 3 3 3 (NIL)

10.4.3 Vollständiger wohlgeformter Ausdruck in $a^k b^k c^k$

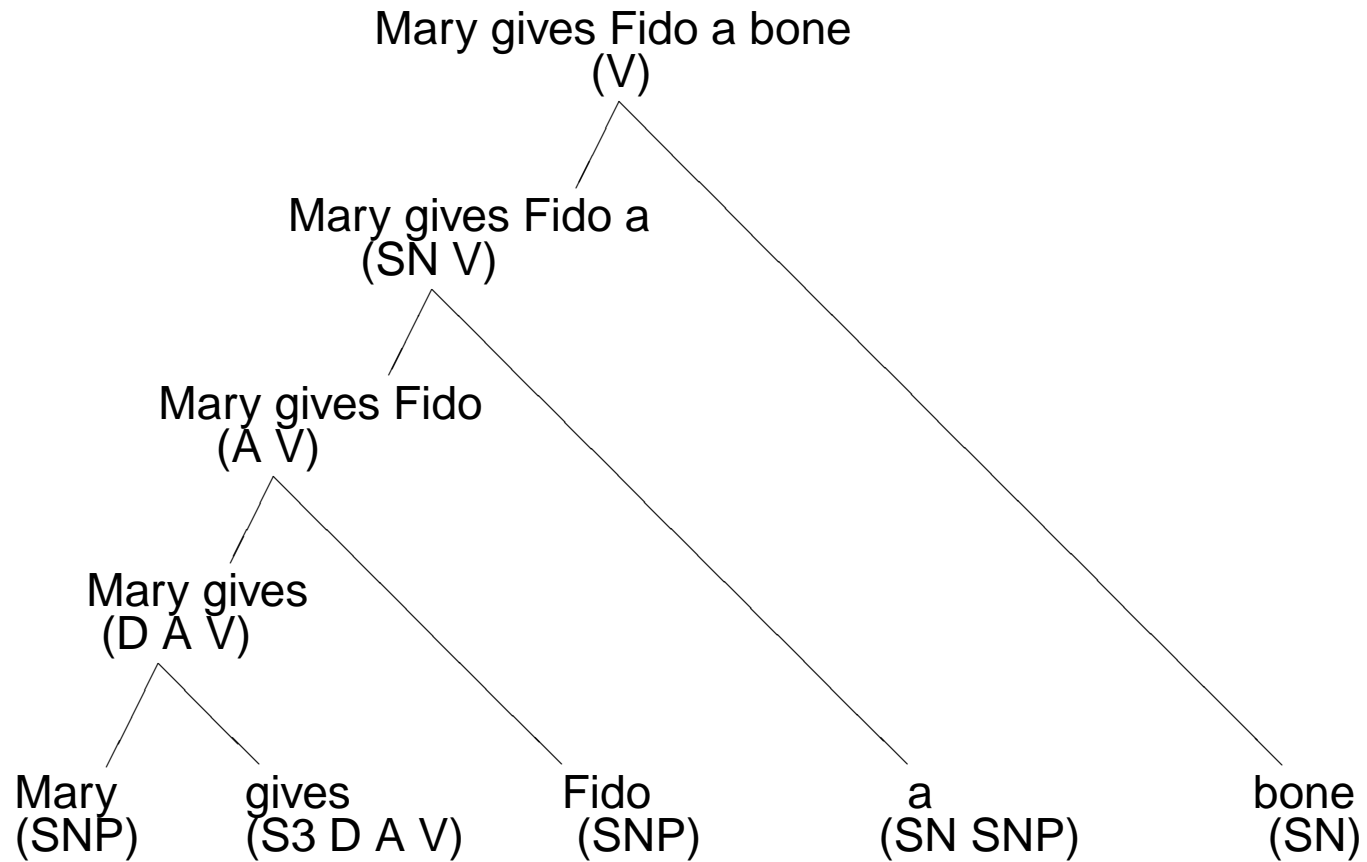
A A A B B B C C C
 1 1 2 2 2 3 3 3 (NIL)

10.5 LA-Grammatik für natürliche Sprachen

10.5.1 Konstituentenstrukturanalyse in der C-Grammatik



10.5.2 Zeitlineare Analyse in der LA-Grammatik



10.5.3 Kategoriale Operation der Komposition von Mary und gives $(\text{SNP}) (\text{N D A V}) \Rightarrow (\text{D A V})$ **10.5.4 Kategoriale Operation der Komposition von Mary gives and Fido** $(\text{D A V}) (\text{SNP}) \Rightarrow (\text{A V})$ **10.5.5 Kategoriale Operation der Komposition von Mary gives Fido and a** $(\text{A V}) (\text{SN SNP}) \Rightarrow (\text{SN V})$ **10.5.6 Kategoriale Operation der Komposition von Mary gives Fido a and book** $(\text{SN V}) (\text{SN}) \Rightarrow (\text{V})$

10.5.7 Automatische Parsinganalyse von Beispiel 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 Analyse eines diskontinuierlichen Elements

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 Analyse einer ungrammatischen Eingabe

```
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. Hierarchie der LA-Grammatik

11.1 Generative Kapazität der unbeschränkten LAG

11.1.1 Grundeigenschaft der unbeschränkten LAG

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

11.1.2 Theorem 1

Die LA-Grammatik in ihrer unbeschränkten Grundform analysiert und generiert *nur* die rekursiven Sprachen.

Beweis: Gegeben sei eine Eingabekette von endlicher Länge n . Jedes Wort in dieser Eingabekette hat endlich viele lexikalische Lesarten (> 0).

Ableitungslänge 1: Die endliche Menge der Anfangszustände ST_S und alle Lesarten des ersten Wortes w_1 ergeben eine endliche Menge wohlgeformter Ausdrücke $WE_1 = \{(ss' rp_S) \mid ss' \in (W^* \times C^+)\}$.

Kombination n: Kombination $k - 1$, $k > 1$, hat eine endliche Menge wohlgeformter Ausdrücke

$WE_k = \{(ss' rp_i) \mid i \in n, ss' \in (W^* \times C^+), \text{ und die Oberfläche von } ss' \text{ hat Länge } k\}$
hervorgebracht. Das nächste Wort w_{k+1} hat endlich viele Lesarten.

Deshalb ist das kartesische Produkt aller Elemente von WE_k und aller Lesarten des nächsten Wortes w_{k+1} eine endliche Menge von geordneten Paaren. Jedes Paar enthält ein Regelpaket mit endlich vielen Regeln. Deshalb ergibt Kombination k nur endlich viele neue Satzanfänge. Die Ableitung dieser endlichen Menge neuer Satzanfänge ist entscheidbar, weil die kategorialen Operationen als totale rekursive Funktionen definiert wurden.

Q.E.D.

11.1.3 Theorem 2

Der Formalismus der LA-Grammatik analysiert und generiert *alle* rekursiven Sprachen.

Beweis: L sei eine rekursive Sprache mit der Wortmenge W . Weil L rekursiv ist, gibt es eine totale rekursive Funktion $\varrho: W^* \rightarrow \{0,1\}$, also die charakteristische Funktion von L . LAG^L sei eine LA-Grammatik mit der folgenden Definition:

Die Menge der Wortoberflächen von LAG^L ist W .

Die Menge der Kategorie-segmente $C =_{def} W \cup \{0,1\}$

Für beliebige $e, f \in W^+$ gilt, $[e (f)] \in LX$ dann und nur dann wenn $e = f$.

$$LX =_{def} \{[a (a)], [b (b)], [c (c)], [d (d)], \dots\}$$

$$ST_S =_{def} \{[(seg_c) \{r_1, r_2\}]\}, \text{ where } seg_c \in \{a, b, c, d, \dots\}$$

$$r_1: (X) (seg_c) \Rightarrow (X seg_c) \quad \{r_1, r_2\}$$

$$r_2: (X) (seg_c) \Rightarrow \varrho (X seg_c) \quad \{ \}$$

$$ST_F =_{def} \{[(1) rp_2]\}$$

Nach jedem Kombinationschritt erlaubt rp_1 zwei mögliche Fortsetzungen: Anwendung von r_1 um die Eingabekette weiterzulesen (und in die Resultatkategorie zu kopieren) oder Anwendung von r_2 , um zu testen, ob die

bisher gelesene Eingabe einen wohlgeformten Ausdruck von L darstellt. In letzterem Falle wird die Funktion ϱ auf die bisher angesammelten Kategorie-segmente der Eingabe angewendet (die der Oberfläche der Eingabe gleichen). Wenn das Ergebnis der Anwendung von r_2 in dem Zustand $[(1) rp_2]$ resultiert, wird die Oberfläche von der Grammatik akzeptiert. Ist das Ergebnis dagegen $[(0) rp_2]$, ist die Eingabe kein wohlgeformter Ausdruck von L .

Da die kategorialen Operationen von LAG^L jede totale rekursive Funktion sein kann, kann LAG^L auf ϱ , also der charakteristischen Funktion von L , basieren. Deshalb akzeptiert und generiert LAG^L alle rekursiven Sprachen.

Q.E.D.

11.1.4 Definition der Klasse der A-LAGs.

Die Klasse der A-LAGs besteht aus unbeschränkten LA-Grammatiken und generiert *alle* rekursiven Sprachen.

11.2 LA-Hierarchie der A-, B- und C-LAGs

11.2.1 Parameter der Komplexität

- Der *Aufwand* an Berechnung pro Regelanwendung, der im schlimmsten Fall benötigt wird.
- Die *Anzahl* der Regelanwendungen, abhängig von der Länge der Eingabe, die bei einer Ableitung im schlimmsten Fall benötigt werden.

11.2.2 Hauptansatzpunkte zur Beschränkung von LAGs

R1: Beschränkungen der Form kategorialer Operationen, um den möglichen Rechenaufwand beliebiger Regelanwendungen zu begrenzen.

R2: Beschränkungen des Ambiguitätsgrades, um die Anzahl der möglichen Regelanwendungen zu begrenzen.

11.2.3 Mögliche Einschränkungen kategorialer Operationen

R1.1: Beschränkung der Kategorielänge.

R1.2: Beschränkung der Kategoriemuster, die bei der Definition kategorialer Operationen verwendet werden dürfen.

11.2.4 Definition der Klasse der B-LAGs

Die Klasse der B-LAGs besteht aus LA-Grammatiken, für die eine Konstante B existiert, so daß bei beliebigen Eingaben der Länge n die Kategorien der Ableitungszwischenschritte höchstens die Länge $B \cdot n$ haben.

11.2.5 Konstante kategoriale Operationen

$$r_i: (\text{seg}_1 \dots \text{seg}_k \text{ X}) \text{ cat}_2 \Rightarrow \text{cat}_3 \text{ rp}_i$$

$$r_i: (\text{X seg}_1 \dots \text{seg}_k) \text{ cat}_2 \Rightarrow \text{cat}_3 \text{ rp}_i$$

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

11.2.6 Nichtkonstante kategoriale Operation

$$r_i: (\text{X seg}_1 \dots \text{seg}_k \text{ Y}) \text{ cat}_2 \Rightarrow \text{cat}_3 \text{ rp}_i$$

11.2.7 Definition der Klasse der C-LAGs

Die Klasse der C-LAGs besteht aus B-LAGs, für die eine endliche Konstante C existiert, so daß keine kategoriale Operation co_i mehr als C Segmente in den Satzanfangskategorien überprüft.

11.2.8 Die Hierarchie der A-LAGs, B-LAGs, und C-LAGs

Die Klasse der A-LAGs analysiert und generiert alle rekursiven Sprachen, die Klasse der B-LAGs analysiert und generiert alle kontextsensitiven Sprachen, und die Klasse der C-LAGs analysiert und generiert viele kontextsensitive Sprachen, alle kontextfreien Sprachen und alle regulären Sprachen.

11.3 Ambiguität in der LA-Grammatik

11.3.1 Faktoren für die Anzahl der Regelanwendungen

1. Länge der Eingabe.
2. Zahl der Regeln in einem Regelpaket, das bei einer Kombination auf ein Eingabepaar (Lesart) angewendet wird.
3. Zahl der Lesarten, die bei einem einzelnen Kombinationsschritt jeweils vorhanden sind.

11.3.2 Einfluß auf die Komplexität

- Faktor 1 ist grammatikunabhängig und wird als die Länge n in der Formel zu Charakterisierung der Komplexität verwendet.
- Faktor 2 ist eine grammatikabhängige Konstante.
- Nur Faktor 3 kann die Zahl der Regelanwendungen über das lineare Anwachsen mit der Länge der Eingabe hinaus in die Höhe treiben. Ob bei einer Eingabe mehrere Regeln eines Regelpakets erfolgreich sein können, hängt von den Eingabebedingungen der Regeln ab.

11.3.3 Mögliche Beziehungen zwischen Eingabebedingungen

1. *Inkompatible* Eingabebedingungen: Zwei Regeln haben inkompatible Eingabebedingungen, wenn es keine Eingabepaare gibt, die von beiden Regeln akzeptiert werden.
2. *Kompatible* Eingabebedingungen: Zwei Regeln haben kompatible Eingabebedingungen, wenn es mindestens ein Eingabepaar gibt, das von beiden Regeln akzeptiert wird, und mindestens ein Eingabepaar, das von der einen, aber nicht von der anderen Regel akzeptiert wird.
3. *Identische* Eingabebedingungen: Zwei Regeln haben identische Eingabebedingungen, wenn für alle Eingabepaare gilt, daß sie entweder von beiden Regeln akzeptiert oder von beiden Regeln abgelehnt werden.

11.3.4 Definition unambiger LA-Grammatiken

Eine LA-Grammatik ist unambig dann und nur dann, wenn (i) für alle Regelpakete gilt, daß deren Regeln *inkompatible* Eingabebedingungen haben, und (ii) es keine lexikalischen Ambiguitäten gibt.

11.3.5 Definition syntaktisch ambiger LA-Grammatiken

Eine LA-Grammatik ist syntaktisch ambig dann und nur dann, wenn (i) es mindestens ein Regelpaket gibt, das mindestens zwei Regeln mit *kompatiblen* Eingabebedingungen enthält, und (ii) es keine lexikalischen Ambiguitäten gibt.

11.3.6 Globale syntaktische Ambiguität

Eine syntaktische Ambiguität ist +global, wenn sie eine Eigenschaft des ganzen Satzes ist.

Beispiel: Flying air planes can be dangerous.

11.3.7 Lokale syntaktische Ambiguität

Eine syntaktische Ambiguität ist –global, wenn sie eine Eigenschaft von nur einem Teil des Satzes ist.

Beispiel: The horse raced by the barn fell.

11.3.8 Rolle der \pm global Unterscheidung

In der LA-Grammatik besteht der Unterschied zwischen +globalen und –globalen Ambiguitäten allein darin, ob mehr als eine Lesart bis zum Satzende ‘überlebt’ oder nicht. Die \pm global Unterscheidung hat keinen Einfluß auf die Komplexität und wird hauptsächlich aus linguistischen Gründen gemacht.

11.3.9 Recursive syntaktische Ambiguität

Eine Ambiguität ist +rekursiv, wenn sie innerhalb einer rekursiven Regelschleife entsteht.

Beispiele: Die C-LAGs für WW^R (siehe 11.5.6) und WW (siehe 11.5.8), die –global sind, und für `SubsetSum` (siehe 11.5.11), die +global ist.

11.3.10 Nicht-rekursive syntaktische Ambiguität

Eine Ambiguität ist –rekursiv, wenn keiner der Ableitungszweige in den ambiguitätsverursachenden Zustand zurückführen kann.

Beispiele: Die C-LAGs für $a^k b^k c^m d^m \cup a^k b^m c^m d^k$ (siehe 11.5.3), die +global ist, und die C-LAGs für natürliche Sprachen in den Kapiteln 17 und 18, die sowohl +global als auch –global ambig sind.

11.3.11 Rolle der \pm rekursiv Unterscheidung

Die \pm recursive Unterscheidung ist entscheidend für die Komplexitätsanalyse, weil gezeigt werden kann, daß bei LA-Grammatiken mit nicht-rekursiven Ambiguitäten die maximale Anzahl der Regelanwendungen pro Kombination durch eine grammatikabhängige Konstante begrenzt ist.

11.3.12 Theorem 3

Wenn eine LA-Grammatik nur –rekursive Ambiguitäten enthält, dann ist die maximale Anzahl der Regelanwendungen

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

für $n > (R - 2)$, wobei n die Länge der Eingabe und R die Anzahl der Regeln der LA-Grammatik ist.

Beweis: Das Parsen einer Eingabe der Länge n erfordert $(n - 1)$ Kombinationsschritte. Wenn eine LA-Grammatik aus R Regeln besteht, dann muß spätestens nach R Kombinationen eine dieser Regeln erneut angewendet werden. Außerdem können in einer Kombination maximal R Regelanwendungen pro Lesart durchgeführt werden.

Gemäß der Definition –rekursiver Ambiguitäten dürfen Regeln, die eine syntaktische Ambiguität verursacht haben, in einem zeitlinearen Ableitungspfad (Lesart) nicht erneut angewendet werden. Bei ihrer ersten Anwendung kann eine Regel r_i also ein Maximum von $R - 1$ neuen Pfaden erzeugen (unter der Annahme, daß das Regelpaket von r_i alle R Regeln der Grammatik außer r_i selbst enthält). Bei der Fortsetzung eines dieser Pfade kann die zweite Ambiguität erzeugende Regel r_j ein Maximum von $R - 2$ neuen Pfaden erzeugen, usw. Wenn die verschiedenen Regeln der LA-Grammatik mit den jeweils noch möglichen maximalen Regelpaketen definiert werden, dann wird nach $R - 2$ Kombinationsschritten ein Maximum von $2^{(R-2)}$ Lesarten erreicht.

Q.E.D.

11.4 Komplexität von Grammatiken und Automaten

11.4.1 Wahl der primitiven 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.

[Griffith und Petrick machen ihre Komplexitätsangaben nicht in Form der tatsächlichen Zeitmessung, sondern auf der Grundlage “primitiver Operationen.” Sie formulierten ihre Algorithmen als Mengen nichtdeterministischer Ersetzungsregeln für ein Turing-Maschinen-artiges Gerät. Dabei ist jede Regelanwendung eine primitive Operation. Wir haben als unsere primitive Operation den Vorgang des Hinzufügens eines Zustands zur Zustandsmenge gewählt (bzw. den Versuch des Hinzufügens eines Zustands, der bereits vorhanden ist). Wir meinen, daß dies mit deren primitiver Operation vergleichbar ist: Beide sind gewissermaßen die komplexeste Operation der jeweiligen Algorithmen, die unabhängig von der Größe der Grammatik und der Länge der Eingabe ist.]

J. Earley 1970, p. 100

11.4.2 Primitive Operation der C-LAGs

Die primitive Operation einer C-LAG bzw. eines C-LAG-Parsers ist eine Regelanwendung (wobei auch ergebnislose Versuche gezählt werden).

11.5 Subhierarchie der C1-, C2- und C3-LAGs

11.5.1 Die Unterklasse der C1-LAGs

Eine C-LAG ist eine C1-LAG wenn sie –rekursiv ambig ist. Die Klasse der C1-LAGs parst in linearer Zeit und enthält alle deterministischen kontextfreien Sprachen, die von einem DPDA ohne λ -Bewegung erkannt werden, plus kontextfreie Sprachen mit –rekursiven Ambiguitäten, wie z. B. $a^k b^k c^m d^m \cup a^k b^m c^m d^k$ sowie viele kontextsensitive Sprachen, z. B. $a^k b^k c^k$, $a^k b^k c^k d^k e^k$, $\{a^k b^k c^k\}^*$, L_{square} , L_{hast}^k , a^{2^i} , $a^k b^m c^{k \cdot m}$ und $a^{i!}$, wobei letztere nicht einmal mehr eine Indexsprache ist.

11.5.2 C1-LAG für kontextsensitives a^{2^i}

$$LX =_{def} \{[a(a)]\}$$

$$ST_S =_{def} \{[(a) \{r_1\}]\}$$

$$r_1: (a) \quad (a) \Rightarrow (aa) \quad \{r_2\}$$

$$r_2: (aX) \quad (a) \Rightarrow (Xbb) \quad \{r_2, r_3\}$$

$$r_3: (bX) \quad (a) \Rightarrow (Xaa) \quad \{r_2, r_3\}$$

$$ST_F =_{def} \{[(aa) rp_1], [(bXb) rp_2], [(aXa) rp_3]\}.$$

11.5.3 C1-LAG für ambiges $a^k b^k c^m d^m \cup a^k b^m c^m d^k$

$LX =_{def} \{[a(a)], [b(b)], [c(c)], [d(d)]\}$

$ST_S =_{def} \{[(a) \{r_1, r_2, r_5\}]\}$

$r_1: (X) \quad (a) \Rightarrow (a X) \quad \{r_1, r_2, r_5\}$

$r_2: (a X) \quad (b) \Rightarrow (X) \quad \{r_2, r_3\}$

$r_3: (X) \quad (c) \Rightarrow (c X) \quad \{r_3, r_4\}$

$r_4: (c X) \quad (d) \Rightarrow (X) \quad \{r_4\}$

$r_5: (X) \quad (b) \Rightarrow (b X) \quad \{r_5, r_6\}$

$r_6: (b X) \quad (c) \Rightarrow (X) \quad \{r_6, r_7\}$

$r_7: (a X) \quad (d) \Rightarrow (X) \quad \{r_7\}$

$ST_F =_{def} \{[\varepsilon rp_4], [\varepsilon rp_7]\}$

11.5.4 Das Single Return Principle (SRP)

Eine rekursive Ambiguität ist *single return*, wenn genau einer der Fortsetzungszweige der Ambiguität in den Zustand zurückkehrt, der die Ambiguität verursachte.

11.5.5 Die Unterklasse der C2-LAGs

Eine C-LAG ist eine C2-LAG wenn sie (i) rekursive Ambiguitäten erzeugt und (ii) alle Ambiguitäten die Beschränkung des *Single-Return-Prinzips* erfüllen (SR-rekursive Ambiguität). Die Klasse der C2-Sprachen parst in polynomialer Zeit und enthält nondeterministische kontextfreie Sprachen wie WW^R und L_{hast}^∞ , plus kontextsensitive languages wie WW , $W^{k \geq 3}$, $\{WWW\}^*$ und $W_1W_2W_1^RW_2^R$.

11.5.6 C2-LAG für kontextfreies WW^R

$$LX =_{def} \{[a(a)], [b(b)], [c(c)], [d(d)] \dots\}$$

$$ST_S =_{def} \{[(seg_c) \{r_1, r_2\}]\}, \text{ where } seg_c \in \{a, b, c, d, \dots\}$$

$$r_1: (X) (seg_c) \Rightarrow (seg_c X) \{r_1, r_2\}$$

$$r_2: (seg_c X) (seg_c) \Rightarrow (X) \{r_2\}$$

$$ST_F =_{def} \{[\varepsilon rp_2]\}$$

11.5.7 Die Ableitungsstruktur des *worst case* in WW^R

rules:

2

1 2 2

1 1 2 2 2

1 1 1 2 2

1 1 1 1 2

1 1 1 1 1

analyses:

a \$ a

a a \$ a a

a a a \$ a a a

a a a a \$ a a

a a a a a \$ a

a a a a a a \$

11.5.8 C2-LAG für kontextsensitives WW

$$LX =_{def} \{[a(a)], [b(b)], [c(c)], [d(d)] \dots\}$$

$$ST_S =_{def} \{[(seg_c) \{r_1, r_2\}]\}, \text{ where } seg_c \in \{a, b, c, d, \dots\}$$

$$r_1: (X) \quad (seg_c) \Rightarrow (X \text{ } seg_c) \{r_1, r_2\}$$

$$r_2: (seg_c X) \quad (seg_c) \Rightarrow (X) \quad \{r_2\}$$

$$ST_F =_{def} \{[\varepsilon \text{ } rp_2]\}$$

11.5.9 C2-LAG für kontextsensitives $W_1 W_2 W_1^R W_2^R$

$$LX =_{def} \{[a(a)], [b(b)]\}$$

$$ST_S =_{def} \{[(seg_c) \{r_{1a}\}], [(seg_c) \{r_{1b}\}]\}, \text{ where } seg_c, seg_d \in \{a, b\}$$

$$r_{1a}: (seg_c) \quad (seg_d) \Rightarrow (\# seg_c seg_d) \quad \{r_2, r_3\}$$

$$r_{1b}: (seg_c) \quad (seg_d) \Rightarrow (seg_d \# seg_c) \quad \{r_3, r_4\}$$

$$r_2: (X) \quad (seg_c) \Rightarrow (X seg_c) \quad \{r_2, r_3\}$$

$$r_3: (X) \quad (seg_c) \Rightarrow (seg_c X) \quad \{r_3, r_4\}$$

$$r_4: (X seg_c) \quad (seg_c) \Rightarrow (X) \quad \{r_4, r_5\}$$

$$r_5: (seg_c X \#) \quad (seg_c) \Rightarrow (X) \quad \{r_6\}$$

$$r_6: (seg_c X) \quad (seg_c) \Rightarrow (X) \quad \{r_6\}$$

$$ST_F =_{def} \{[\varepsilon rp_5], [\varepsilon rp_6]\}$$

11.5.10 The subclass of C3-LAGs

Eine C-LAG ist eine C3-LAG wenn sie unbeschränkte rekursive Ambiguitäten enthält. Die Unterklasse der C3-LAGs parst in exponentieller Zeit und enthält u.a. die deterministische kontextfreie Sprache L_{no} , die *hardest context-free language* HCFL, plus kontextsensitive Sprachen wie **SubsetSum** and **SAT**, die \mathcal{NP} -vollständig sind.

11.5.11 C3-LAG für SubsetSum.

$$LX =_{def} \{[0 (0)], [1 (1)], [\# (\#)]\}$$

$$ST_S =_{def} \{[(seg_c) \{r_1, r_2\}]\}, \text{ where } seg_c \in \{0, 1\}$$

$$seg_c \in \{0, 1\}$$

$$r_1: (X) \quad (seg_c) \quad \Rightarrow \quad (seg_c X) \{r_1, r_2\}$$

$$r_2: (X) \quad (\#) \quad \Rightarrow \quad (\# X) \quad \{r_3, r_4, r_6, r_7, r_{12}, r_{14}\}$$

$$r_3: (X seg_c) \quad (seg_c) \quad \Rightarrow \quad (0 X) \quad \{r_3, r_4, r_6, r_7\}$$

$$r_4: (X \#) \quad (\#) \quad \Rightarrow \quad (\# X) \quad \{r_3, r_4, r_6, r_7, r_{12}, r_{14}\}$$

$$r_5: (X seg_c) \quad (seg_c) \quad \Rightarrow \quad (0 X) \quad \{r_5, r_6, r_7, r_{11}\}$$

$$r_6: (X 1) \quad (0) \quad \Rightarrow \quad (1 X) \quad \{r_5, r_6, r_7, r_{11}\}$$

$$r_7: (X 0) \quad (1) \quad \Rightarrow \quad (1 X) \quad \{r_8, r_9, r_{10}\}$$

$$r_8: (X seg_c) \quad (seg_c) \quad \Rightarrow \quad (1 X) \quad \{r_8, r_9, r_{10}\}$$

$$r_9: (X 1) \quad (0) \quad \Rightarrow \quad (0 X) \quad \{r_5, r_6, r_7, r_{11}\}$$

$$r_{10}: (X 0) \quad (1) \quad \Rightarrow \quad (0 X) \quad \{r_8, r_9, r_{10}\}$$

$$r_{11}: (X \#) \quad (\#) \quad \Rightarrow \quad (\# X) \quad \{r_3, r_4, r_6, r_7, r_{12}, r_{14}\}$$

$$r_{12}: (X 0) \quad (seg_c) \quad \Rightarrow \quad (0 X) \quad \{r_4, r_{12}, r_{14}\}$$

$$r_{13}: (X 0) \quad (seg_c) \quad \Rightarrow \quad (0 X) \quad \{r_{11}, r_{13}, r_{14}\}$$

$$r_{14}: (X 1) \quad (seg_c) \quad \Rightarrow \quad (1 X) \quad \{r_{11}, r_{13}, r_{14}\}$$

$$ST_F =_{def} \{[(X) rp_4]\}$$

11.5.12 Restriktionstypen der LA-Grammatik

0. Typ-A: keine Beschränkung
1. Typ-B: Die Länge der Zwischenausdrucks-kategorien ist über $B \cdot n$ beschränkt, wobei B eine Konstante und n die Gesamtlänge der Eingabe ist (*R1.1*, Aufwand).
2. Typ-C3: Die Form der Kategoriemuster resultiert in einer konstanten Beschränkung C der kategorialen Operationen (*R1.2*, Aufwand).
3. Typ-C2: Wie LA-Typ-C3 und die Grammatik ist höchstens *single return* rekursiv ambig (*R2*, Anzahl).
4. Typ-C1: Wie LA-Typ-C3 und die Grammatik ist höchstens –rekursiv ambig (*R2*, Anzahl).

11.5.13 LA-grammatische Hierarchie der formalen Sprachen

Beschränkung	Typen der LAG	Sprachklassen	Komplexitätsgrad
LA-Typ-C1	C1-LAGs	C1-Sprachen	linear
LA-Typ-C2	C2-LAGs	C2-Sprachen	polynomial
LA-Typ-C3	C3-LAGs	C3-Sprachen	exponentiell
LA-Typ-B	B-LAGs	B-Sprachen	exponentiell
LA-Typ-A	A-LAGs	A-Sprachen	rekursiv

12. LA- und PS-Hierarchien im Vergleich

12.1 Sprachklassen der LA- und PS-Grammatik

12.1.1 Komplexitätsklassen der LA- und PS-Hierarchie

	<i>LA-Grammatik</i>	<i>PS-Grammatik</i>
<i>unentscheidbar</i>	—	rekursiv aufzählbare Sprachen
<i>rekursiv</i>	A-Sprachen	
<i>exponentiell</i>	B-Sprachen	kontextsensitive Sprachen
<i>exponentiell</i>	C3-Sprachen	
<i>polynomial</i>	C2-Sprachen	kontextfreie Sprachen
<i>linear</i>	C1-Sprachen	reguläre Sprachen

12.2 Inklusionsverhältnisse in den beiden Hierarchien

12.2.1 Inklusionsverhältnisse der PS-Sprachklassen

reguläre Spr. \subset kontextfreie Spr. \subset kontextsensitive Spr. \subset rekursiv aufzählbare Sprachen

12.2.2 Inklusionsverhältnisse der LA-Sprachklassen

C1-Sprachen \subseteq C2-Sprachen \subseteq C3-Sprachen \subseteq B-Sprachen \subset A-Sprachen

12.3 Nichtäquivalenz der LA- und der PS-Hierarchie

12.3.1 Sprachen, die in der PS-Grammatik in der gleichen, aber in der LA-Grammatik in verschiedenen Klassen liegen

$a^k b^k$ und WW^R liegen in der PS-Grammatik in derselben Klasse (nämlich kontextfrei), aber in verschiedenen Klassen in der LA-Grammatik: $a^k b^k$ ist eine C1-LAG, die ein linearer Zeit parst, während WW^R eine C2-LAG ist und in n^2 parst.

12.3.2 Sprachen, die in der PS-Grammatik in verschiedenen, aber in der LA-Grammatik in der gleichen Klasse liegen

$a^k b^k$ und $a^k b^k c^k$ liegen in der LA-Grammatik in der gleichen Klasse (nämlich C1-LAGs), aber in verschiedenen Klassen in der PS-Grammatik: $a^k b^k$ ist kontextfrei, während $a^k b^k c^k$ kontextsensitiv ist.

12.3.3 Inhärente Komplexität

Die inhärente Komplexität einer Sprache basiert auf der Anzahl der Operationen, die im *worst case* auf einer abstrakten Maschine (z.B. einer Turing- oder Register-Maschine) benötigt werden. Diese Form der Analyse findet auf einer sehr niedrigen Ebene statt, die Maschinen- oder Assembler-Code entspricht.

12.3.4 Klassenbestimmte Komplexität

Die Komplexität künstlicher und natürlicher Sprachen wird normalerweise auf der Abstraktionsebene des Grammatikformalismus analysiert, wobei die Komplexität für den Grammatiktyp und die zugehörige Sprachklasse als ganzes bestimmt wird.

12.3.5 Unterschied zwischen den beiden Komplexitätstypen

Sprachen mit einer inhärent hohen Komplexität (z.B. 3SAT und SUBSET SUM) sind notwendigerweise in einer hohen Komplexitätsklasse (hier exponentiell) in jedem möglichen Grammatikformalismus.

Sprachen mit einer inhärent niedrigen Komplexität (z.B. $a^k b^k c^k$) kann eine hohe oder eine niedrige Komplexität zugewiesen werden, abhängig vom gewählten Grammatikformalismus.

12.3.6 PS-Grammatik für L_{no}

$$\begin{array}{lll}
 S \rightarrow 1S1 & S \rightarrow 1S & S \rightarrow \# \\
 S \rightarrow 0S0 & S \rightarrow 0S &
 \end{array}$$

12.3.7 PS-grammatische Ableitung von 10010#101 in L_{no}

derivation tree:	generated chains:	states:
<pre> S / \ 1 S 1 / \ 0 S 0 / \ 0 S 1 / \ 1 S 1 / \ 0 S # </pre>	<p>1S1</p> <p>10S01</p> <p>100S01</p> <p>1001S101</p> <p>10010S101</p> <p>10010#101</p>	<p>1.S1 1S1.</p> <p>1.S</p> <p>0.S0 0S0.</p> <p>0.S</p> <p>0.S0</p> <p>0.S 0S.</p> <p>1.S1 1S1.</p> <p>1.S</p> <p>0.S0</p> <p>0.S 0S.</p> <p>#.</p>

12.3.8 C3-LAG für L_{no}

$LX =_{def} \{[0 (0)], [1 (1)], [# (\#)]\}$

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

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

$r_2: (seg_c)(seg_d) \Rightarrow (seg_d) \quad \{r_1, r_2, r_3, r_4, r_5\}$

$r_3: (X)(seg_c) \Rightarrow (X) \quad \{r_1, r_2, r_3, r_4, r_5\}$

$r_4: (X)(seg_c) \Rightarrow (seg_c X) \quad \{r_1, r_2, r_3, r_4, r_5\}$

$r_5: (X)(\#) \Rightarrow (X) \quad \{r_6\}$

$r_6: (seg_c X)(seg_c) \Rightarrow (X) \quad \{r_6\}$

$ST_F =_{def} \{[\varepsilon rp_6]\}$

12.4 Vergleich der unteren LA- und PS-Sprachklassen

Die kontextfreie PS-Grammatik hat weite Verwendung gefunden, weil sie innerhalb der PS-Hierarchie die größte generative Kapazität hat, die gerade noch praktisch geparkt werden kann.

12.4.1 Wie geeignet ist die kontextfreie Grammatik für die Beschreibung der natürlichen und der Programmiersprachen?

In der Linguistik besteht allgemeine Übereinstimmung, daß die kontextfreie PS-Grammatik nicht auf die Strukturen paßt, die für die natürlichen Sprachen charakteristisch sind. Entsprechendes gilt für die Informatik:

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.

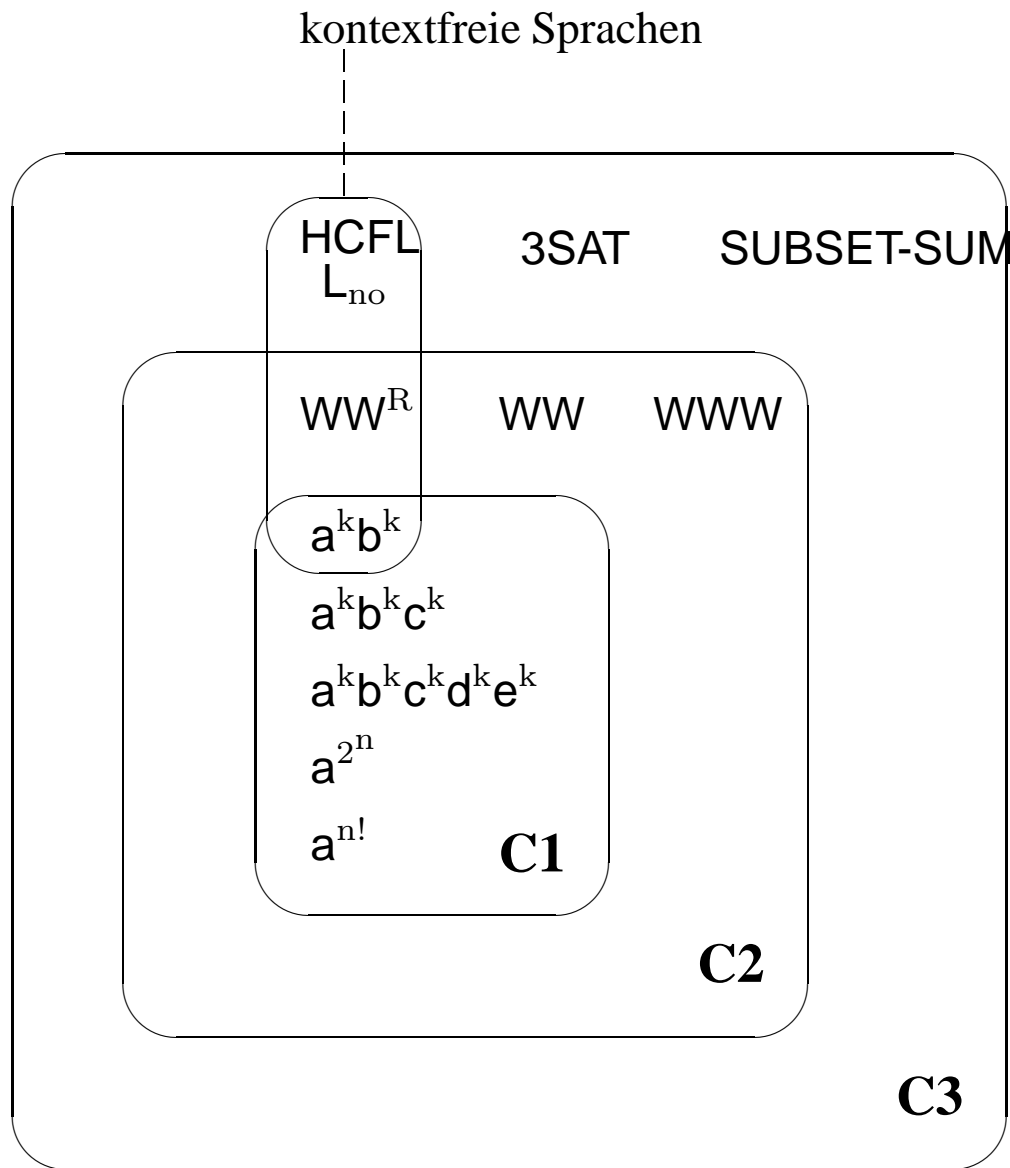
[Es ist kein Geheimnis, daß die kontextfreien Sprachen nur eine allererste Annäherung an die verschiedenen Mechanismen sind, die zur Spezifikation der Syntax moderner Programmiersprachen verwendet werden.]

Ginsberg 1980, S.7

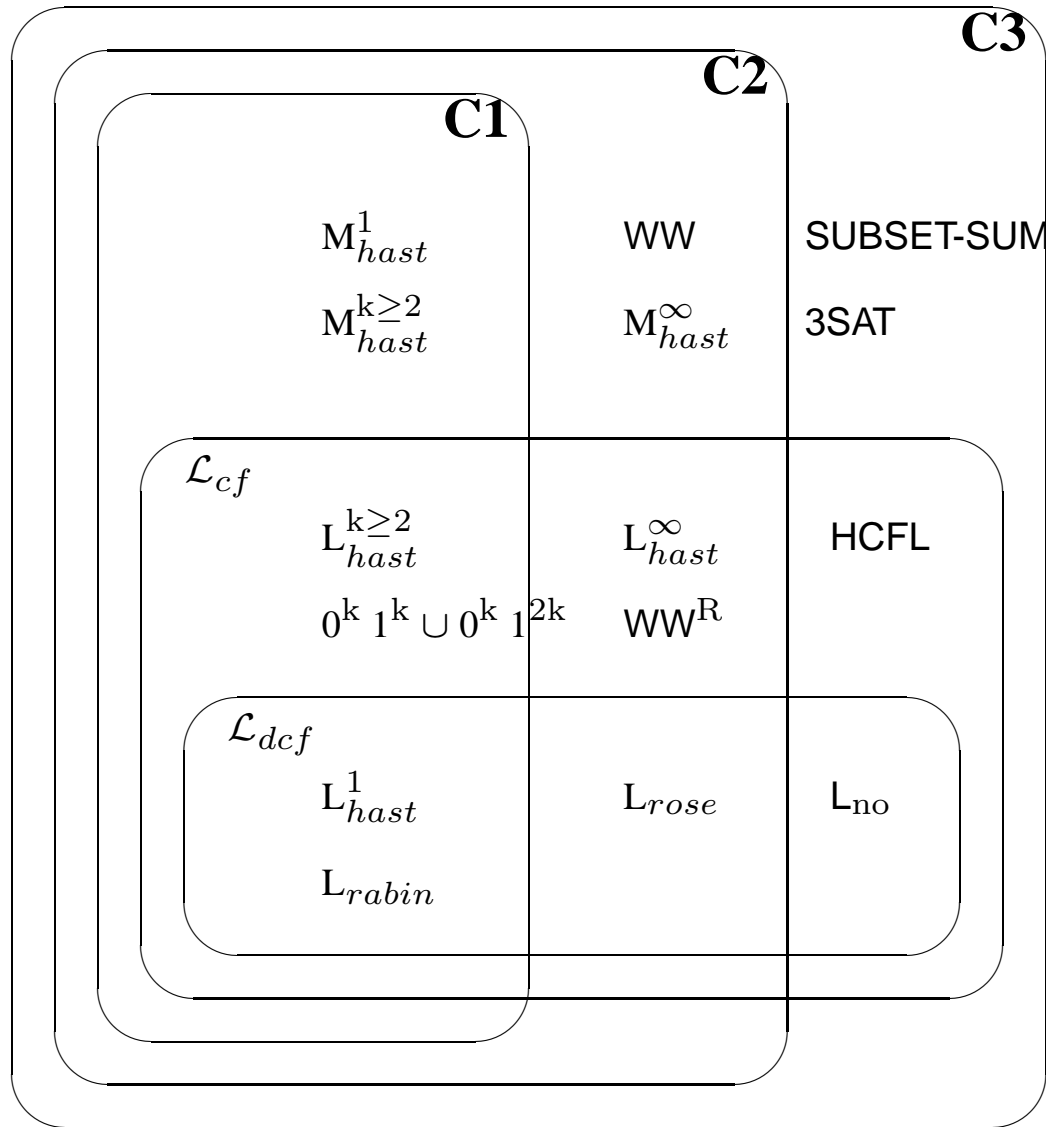
12.4.2 Konservative Erweiterungen der PS-grammatischen Hierarchie

reguläre Spr. \subset kontextfreie Spr. \subset TAL \subset Indexspr. \subset kontextsensitive Spr. \subset rek. aufzählbare Spr.

12.4.3 Orthogonales Verhältnis der C- und CF-Sprachen



12.4.4 Orthogonale Klassifikationen \mathcal{L}_{dcf} , \mathcal{L}_{cf} , C_1 , C_2 und C_3



12.5 Die lineare Komplexität der natürlichen Sprachen

12.5.1 Warum die natürlichen Sprachen wahrscheinlich C-Sprachen sind

In einer kontextsensitiven Sprache, die keine C-Sprache ist, müßte das Wachstum der Kategorienlänge gerade noch in die LBA-Definition der kontextsensitiven Sprachen hineinpassen, aber die musterbasierten kategorialen Operationen der C-LAGs überfordern.

Daß gerade diese Struktur für die natürlichen Sprachen charakteristisch sein soll, ist unwahrscheinlich.

12.5.2 Wenn die natürlichen Sprachen C-LAGs sind,

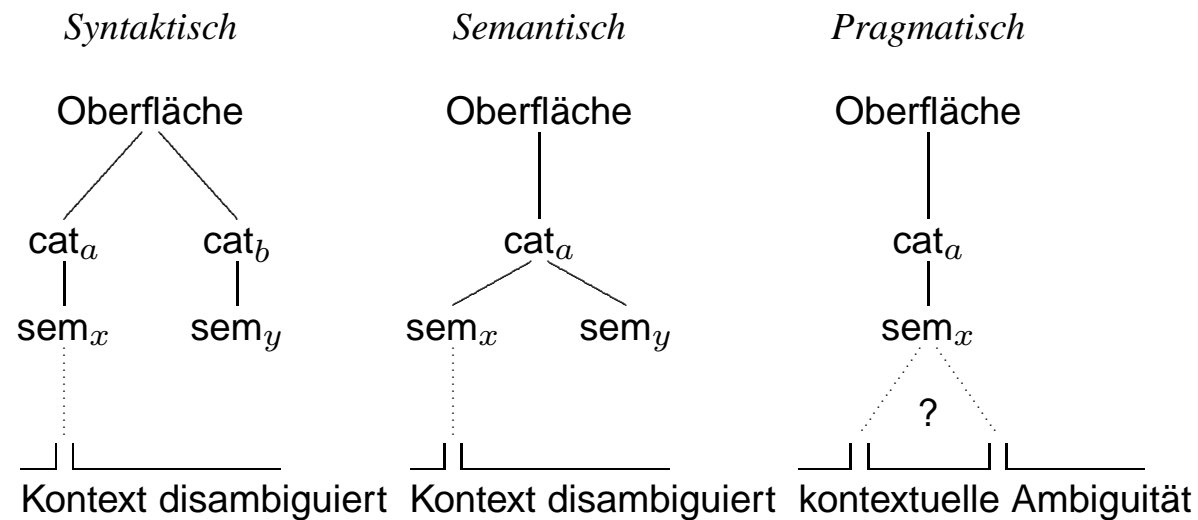
dann sind die folgenden Fragen äquivalent:

(i) *Wie komplex sind die natürlichen Sprachen?*

(ii) *Wie ambig sind die natürlichen Sprachen?*

Dies gilt weil die Unterklassen der C-LAGs sich unterscheiden sich nur in ihren Ambiguitätsgraden unterscheiden.

12.5.3 SLIM-theoretische Analyse von Ambiguität

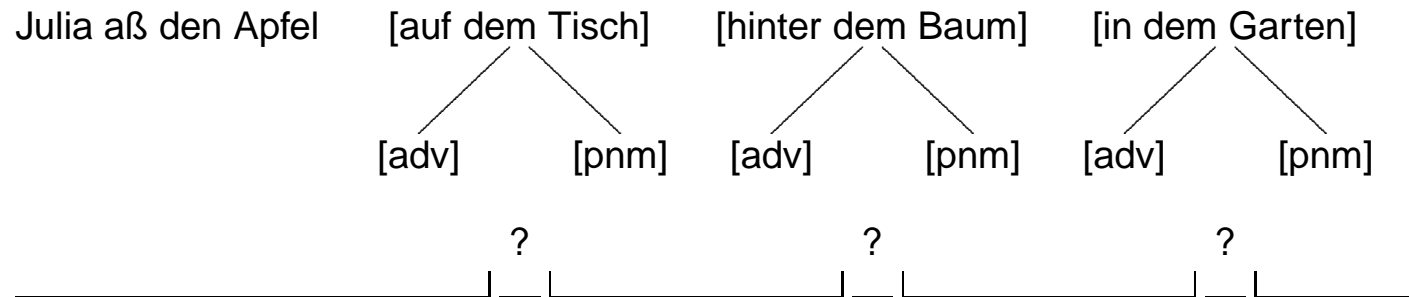


12.5.4 Mehrfachinterpretation bei Präpositionalphrasen

Susanne sah den Mann mit dem Fernrohr.

Julia aß den Apfel auf dem Tisch hinter dem Baum in dem Garten.

12.5.8 Korrekte Analyse mit *semantic doubling*



12.5.9 KoNSyx-Hypothese (Komplexität natürlichsprachlicher Syntax)

Die natürlichen Sprachen liegen in der Klasse der C1-Sprachen und parsen in linearer Zeit.

Diese Hypothese gilt so lange keine rekursiven Ambiguitäten in den natürlichen Sprachen gefunden werden.