Schwach kontextsensitive Grammatik-Formalismen

Petra Schmidt 22.03.2007

Inhalt

- Linguistische Motivation
- Grammatikformalismen
- TAG: Beschreibung, Beispielgrammatik und Analyse
- CCG: Beschreibung, Beispielgrammatik und Analyse
- LIG: Beschreibung, Beispielgrammatik und Analyse
- Beweis Äquivalenz von TAG, CCG und LIG
- Schluss
- Literatur
- Anhang (formale Definitionen)

Linguistische Motivation

Natürliche Sprachen sind nicht kontextfrei

Beispielsatz aus dem Holländischen:
ik haar hem de nijlpaarden zag helpen voeren
(dass) ich sie ihm die Nilpferde sah helfen füttern

Idee: Charakterisierung einer Klasse formaler Grammatikformalismen

- die auch Sprachen wie Holländisch vollständig abdecken,
- deren nicht kontextfreie, überkreuzende syntaktische Abhängigkeiten beschreiben kann
- dabei aber immer noch effizient verarbeitbar sind (polynomielle Parsingalgorithmen)

Grammatikformalismen

- Typische Vertreter der schwach-kontextsensitiven Grammatikformalismen, die auf unterschiedlichen Paradigmen beruhen:
  - TAG: Baumersetzungssystem
  - CCG: CG, die eingeschränkte Komposition von Kategorien hinzufügt
  - LIG: CFG, die die Möglichkeit bietet, die Nichtterminale mit einem Stack zu versehen
- TAG und CCG sind die beiden Formalismen, die in der Computerlinguistik wirklich genutzt werden
- LIG ist eher theoretischer Natur, wird für Beweise und Parsingalgorithmen eingesetzt

TAG: Beschreibung

- TAG wurde hauptsächlich von Aravind Joshi entwickelt (Joshi et al. 1975 und Joshi and Vijay-Shanker 1985)

- besteht aus
  - Initialbäumen und
  - Auxiliarbäumen

- anwendbare Operationen:
  - Adjunktion von Auxiliarbäumen in Auxiliarbäume
  - Substitution eines Blattes A durch Initialbäume mit Root A (durch Adjunktion modellierbar)

TAG: Beispielgrammatik

Beispielsatz (aus Folie 3):

ik haar hem de nijlpaarden zag helpen voeren

Beispielgrammatik

$G = <V_N, V_T, T_{ini}, T_{aux}, S>$

$V_N = \{S, NP, V_1, V_2, V_3\}$

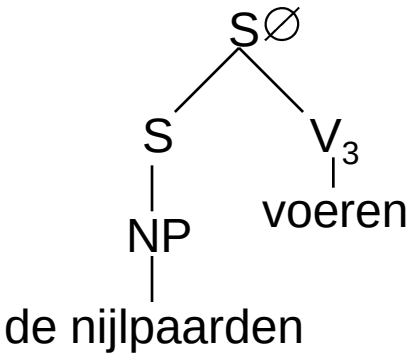
$V_T = \{ik, haar, hem, de\ nilpaarden, zag, helpen, voeren\}$

$T_{ini} = \{\alpha_1\}$

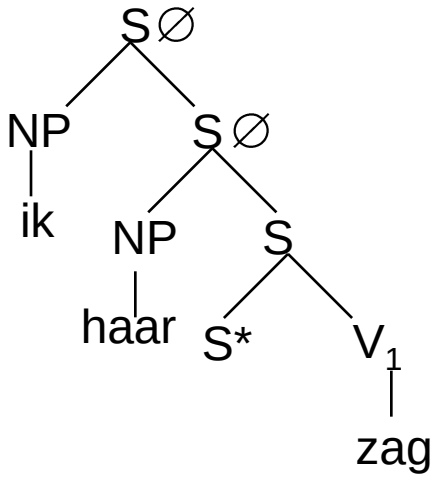
$T_{aux} = \{\beta_1, \beta_2\}$

TAG: Beispielgrammatik

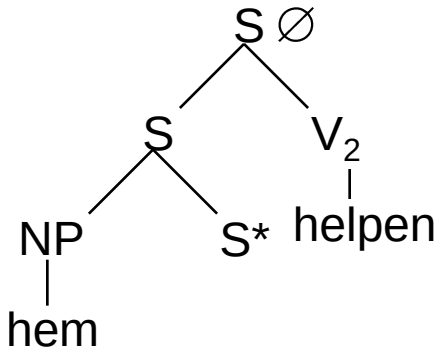
(α_1)



(β_1)

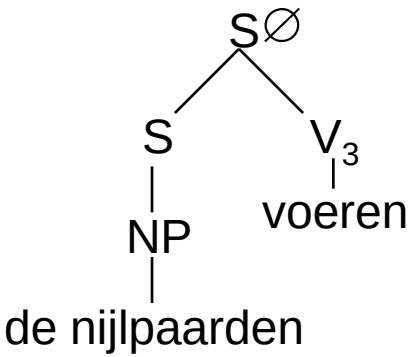


(β_2)

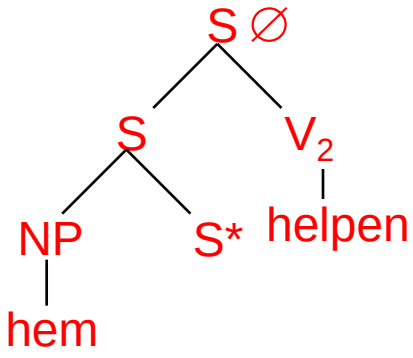


TAG: Analyse

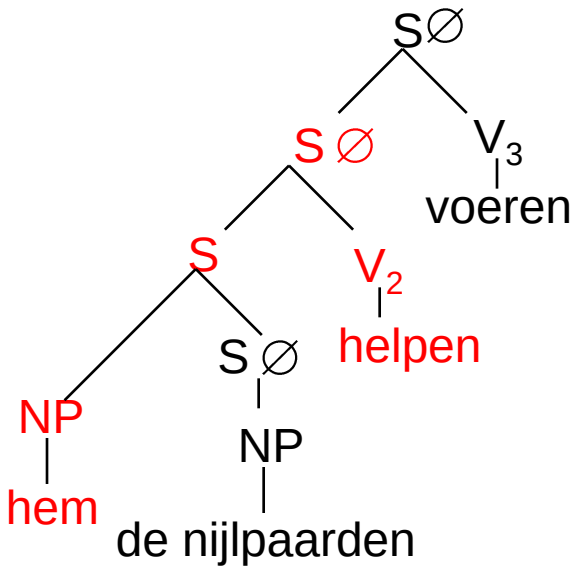
(α_1)



(β_2)

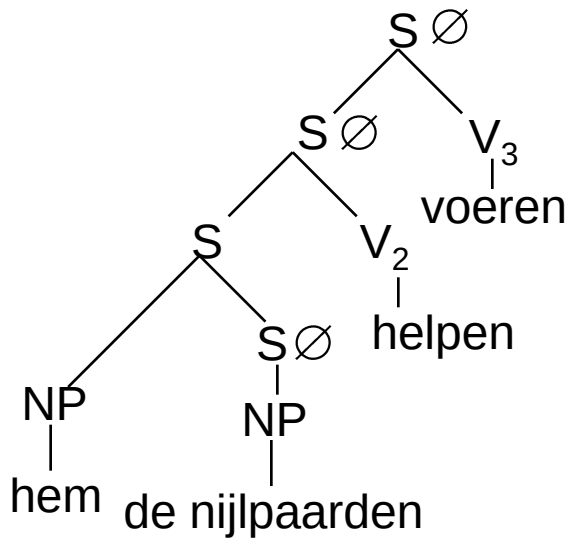


(γ_1)

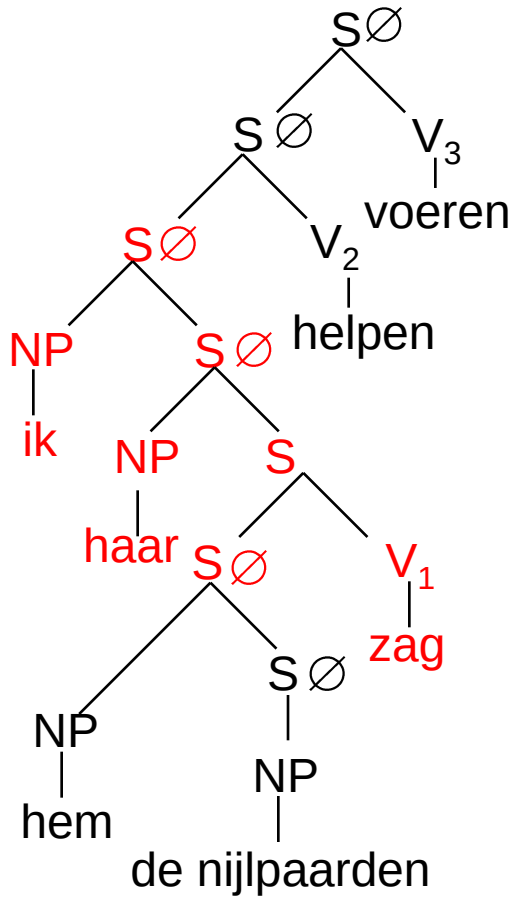


TAG: Analyse

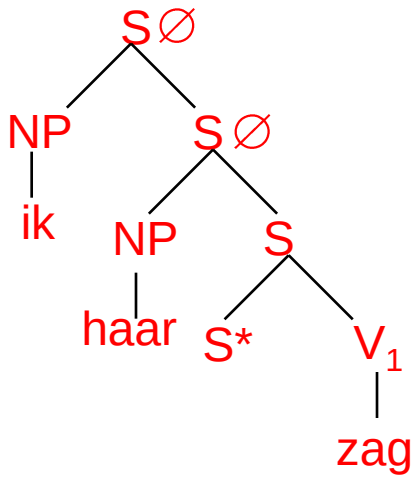
(γ_1)



(γ_2)



(β_1)



CCG: Beschreibung

- CCG wurde von Steedman entwickelt (erste Papiere 80er Jahre, „The syntactic process“ 2000)

- besteht aus:
  - einer endl. Menge von atomaren Kategorien
    - (z.B. s für Satz, n für Namen…)
  - Konnektoren (/, \) mittels denen aus atomaren Kategorien rekursiv komplexe Kategorien gebildet werden können
    - (z.B. s\n für fliegt)
  - einem Lexikon, das Wörter auf Kategorien abbildet
    - z.B. Hans := n, fliegt := s\n

CCG:Beschreibung

- syntaktische Regeln (Kombinationsregeln), die die Kombination von Kategorien definieren
  - Applikation
    - Rückwärtsapplikation (<): y x\y $\Rightarrow$ x (Bsp: Hans fliegt: n s\n $\Rightarrow$ s)
    - Vorwärtsapplikation (>): x/y y $\Rightarrow$ x
  - Komposition
    - Rückwärtskomposition ($B_<$): x\y y\z $\Rightarrow$ x\z
    - Vorwärtskomposition ($B_>$): x/y y/z $\Rightarrow$ x/z
    - Rückwärtskreuzkomposition ($B_<$*): x\y y/z $\Rightarrow$ x/z
    - Vorwärtskreuzkomposition ($B_>$*): x/y y\z $\Rightarrow$ x\z

CCG: Beschreibung

- syntaktische Regeln (Kombinationsregeln), die die Kombination von Kategorien definieren
  - Applikation
    - Rückwärtsapplikation (<): y x\y $\Rightarrow$ x (Bsp: Hans fliegt: n s\n $\Rightarrow$ s)
    - Vorwärtsapplikation (>): x/y y $\Rightarrow$ x
  - Komposition
    - Rückwärtskomposition ($B_<$): x\y y\z $\Rightarrow$ x\z
    - Vorwärtskomposition ($B_>$): x/y y/z $\Rightarrow$ x/z
    - Rückwärtskreuzkomposition ($B_<$*): x\y y/z $\Rightarrow$ x/z
    - Vorwärtskreuzkomposition ($B_>$*): x/y y\z $\Rightarrow$ x\z

CCG: Beispielgrammatik

Beispielsatz (aus Folie 3):

ik haar hem de nijlpaarden zag helpen voeren

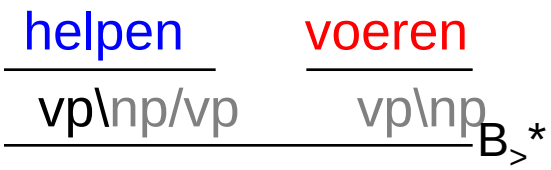
Lexikon

- ik, haar, hem, de nilpaarden := np
- zag := s\np\np/vp
- helpen := vp\np/vp
- voeren := vp\np

CCG: Analyse

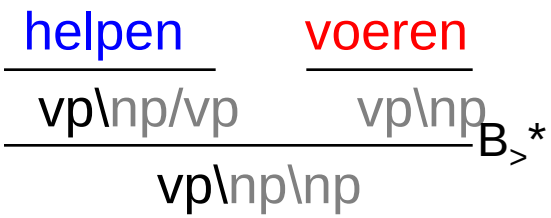
helpen	voeren
vp\np/vp	vp\np

CCG: Analyse



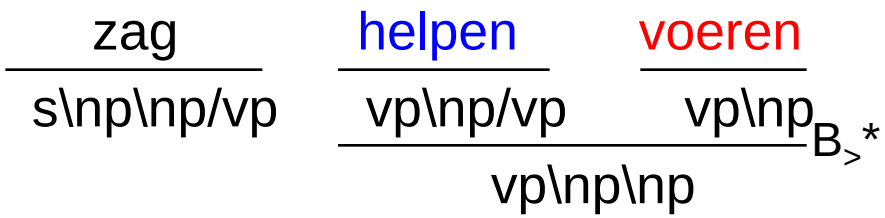
B>*: x/y y\z ⇒ x\z

CCG: Analyse

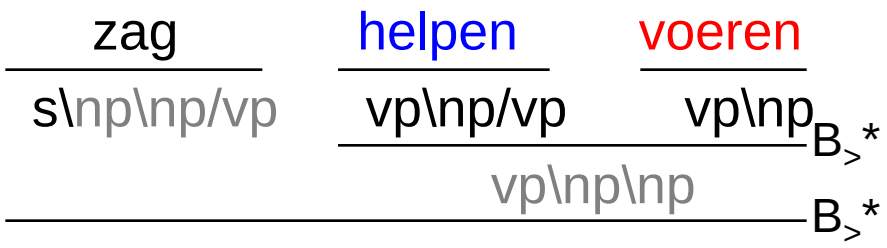


B>*: x/y y\z ⇒ x\z

CCG: Analyse

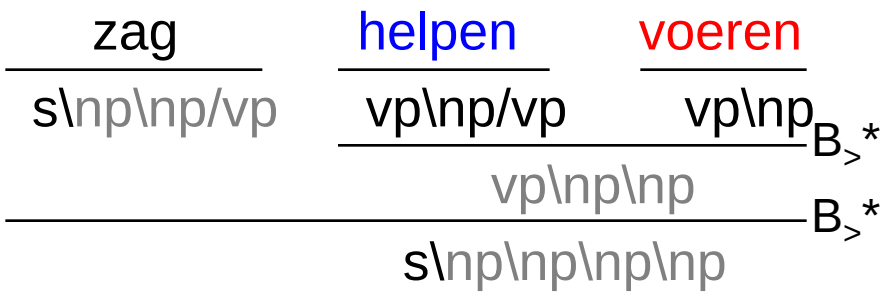


CCG: Analyse



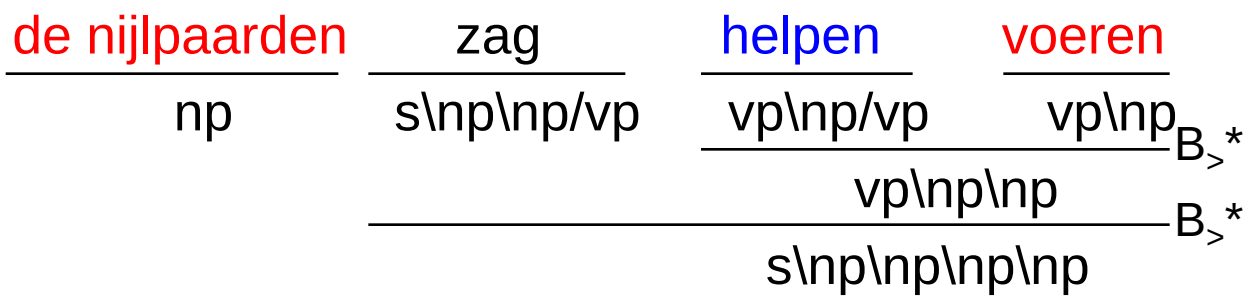
B>*: x/y y\z ⇒ x\z

CCG: Analyse

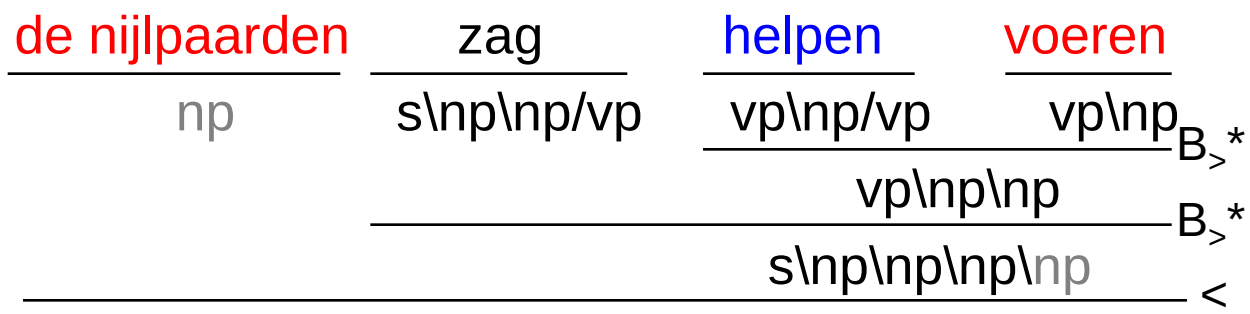


B>*: x/y y\z ⇒ x\z

CCG: Analyse

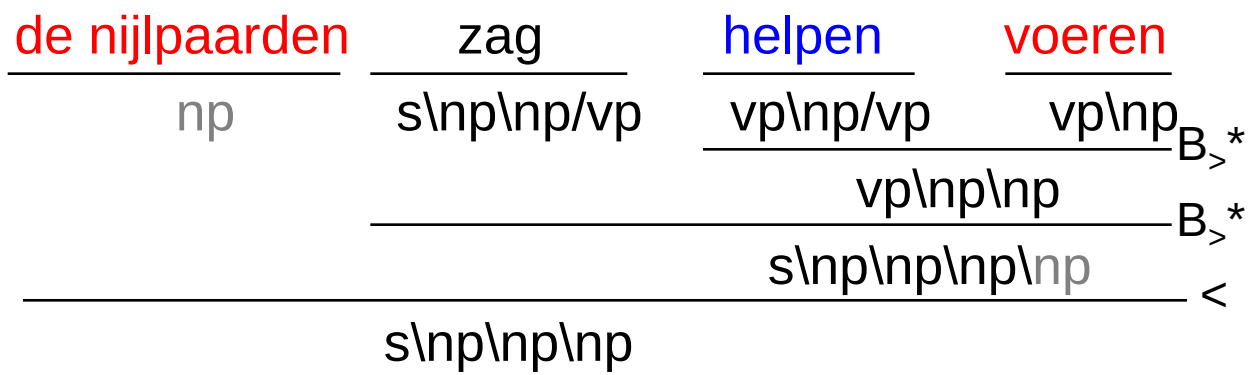

de nijlpaarden
zag
helpen
voeren
np
s\np\np/vp
vp\np/vp
vp\np
B>*
vp\np\np
B>*
s\np\np\np\np

CCG: Analyse



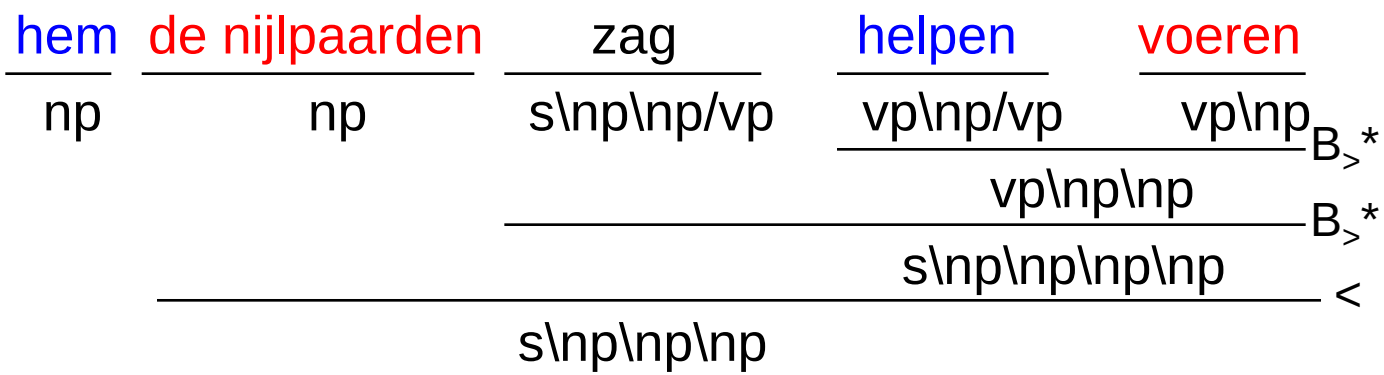
<: y x\y ⇒ x

CCG: Analyse

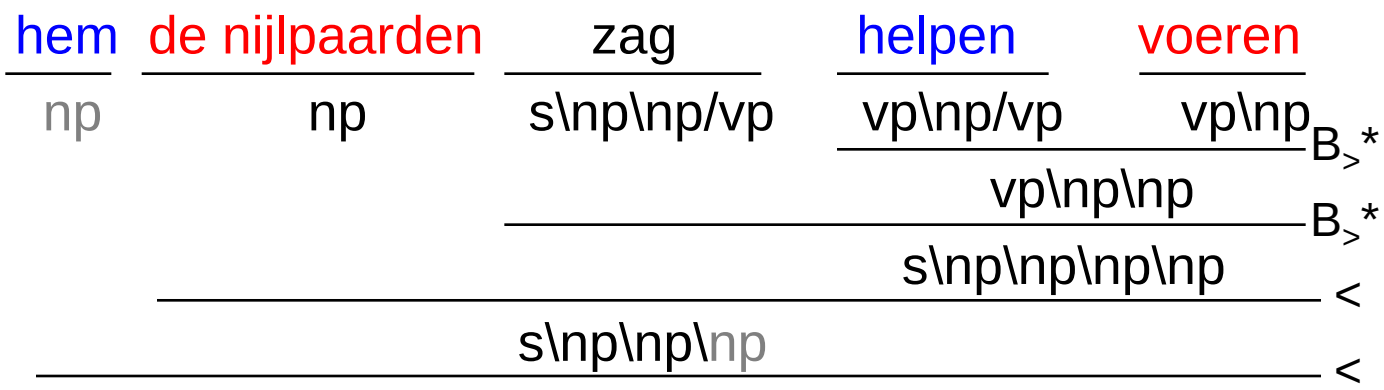


<: y x\y ⇒ x

CCG: Analyse

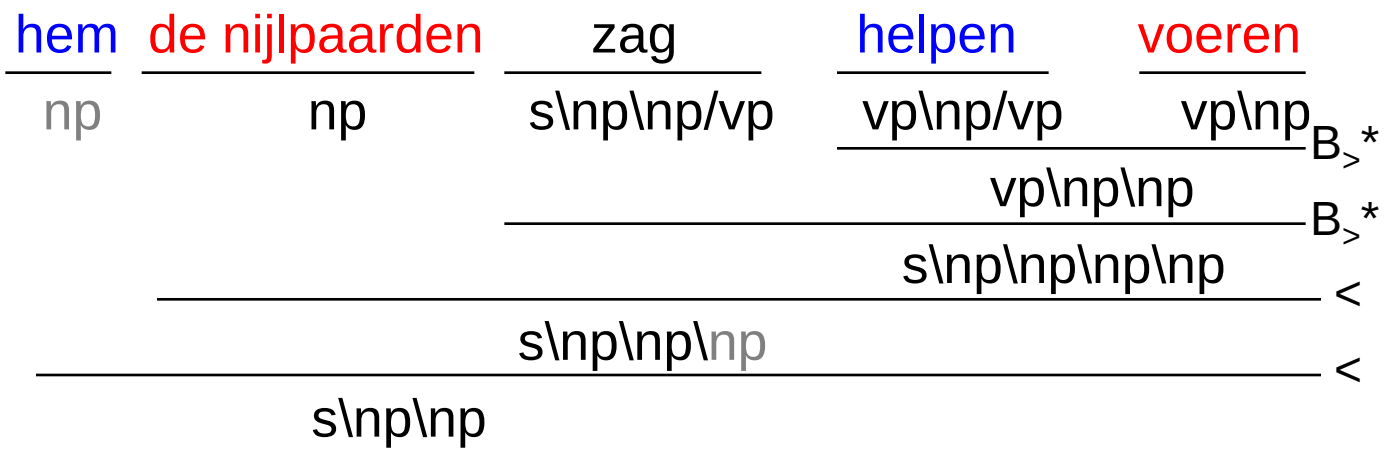

hem
de nijlpaarden
zag
helpen
voeren
np
np
s\np\np/vp
vp\np/vp
vp\np
B>*
vp\np\np
B>*
s\np\np\np\np
<
s\np\np\np

CCG: Analyse



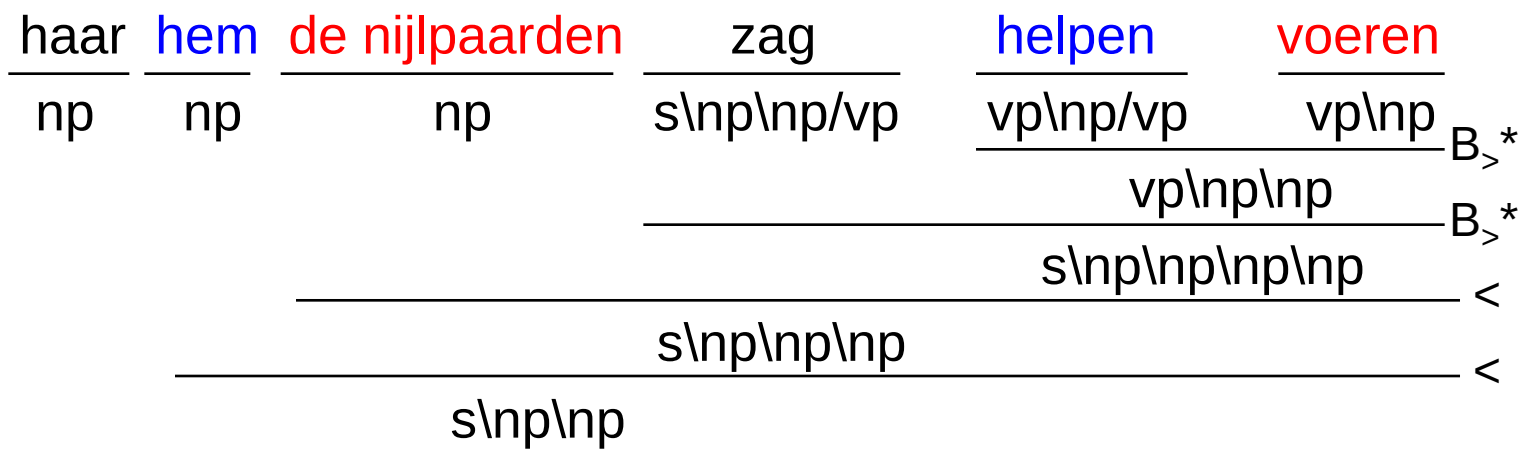
<: y x\y ⇒ x

CCG: Analyse

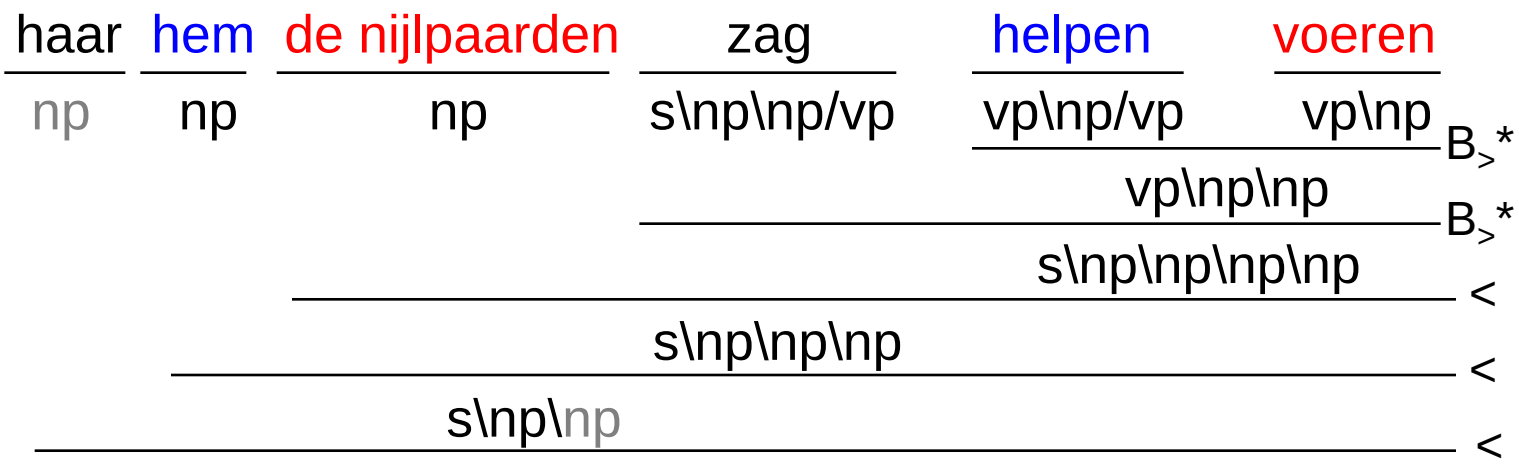


<: y x\y ⇒ x

CCG: Analyse

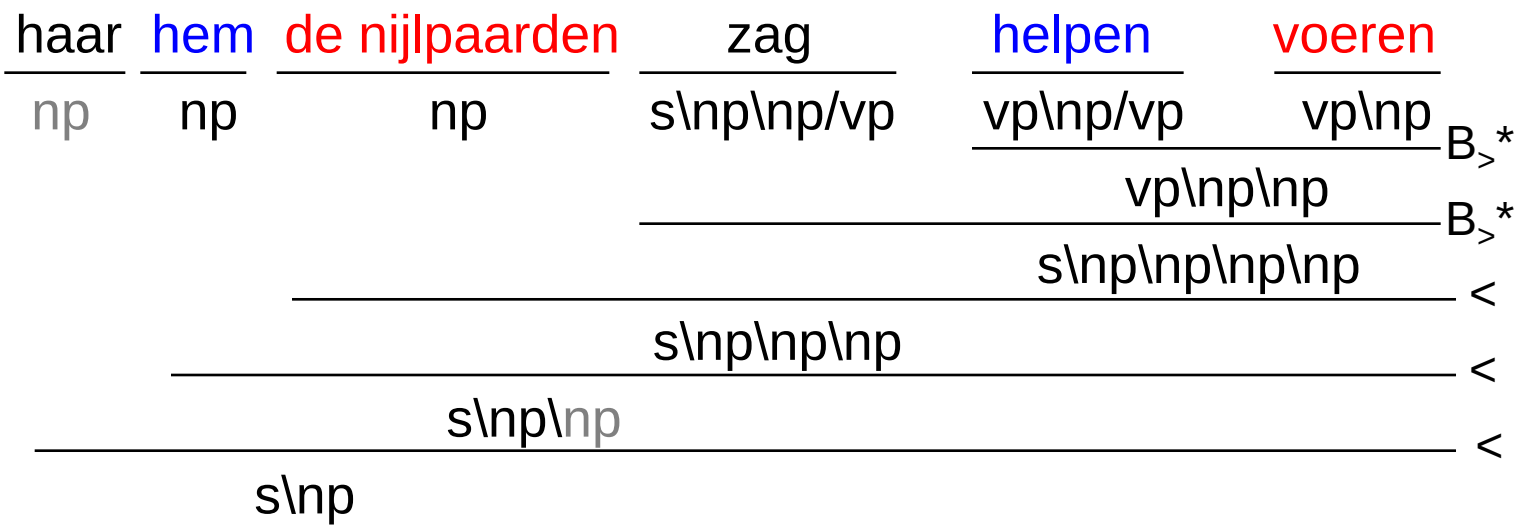

haar
hem
de nijlpaarden
zag
helpen
voeren
np
np
np
s\np\np/vp
vp\np/vp
vp\np
B>*
vp\np\np
B>*
s\np\np\np\np
<
s\np\np\np
<
s\np\np

CCG: Analyse



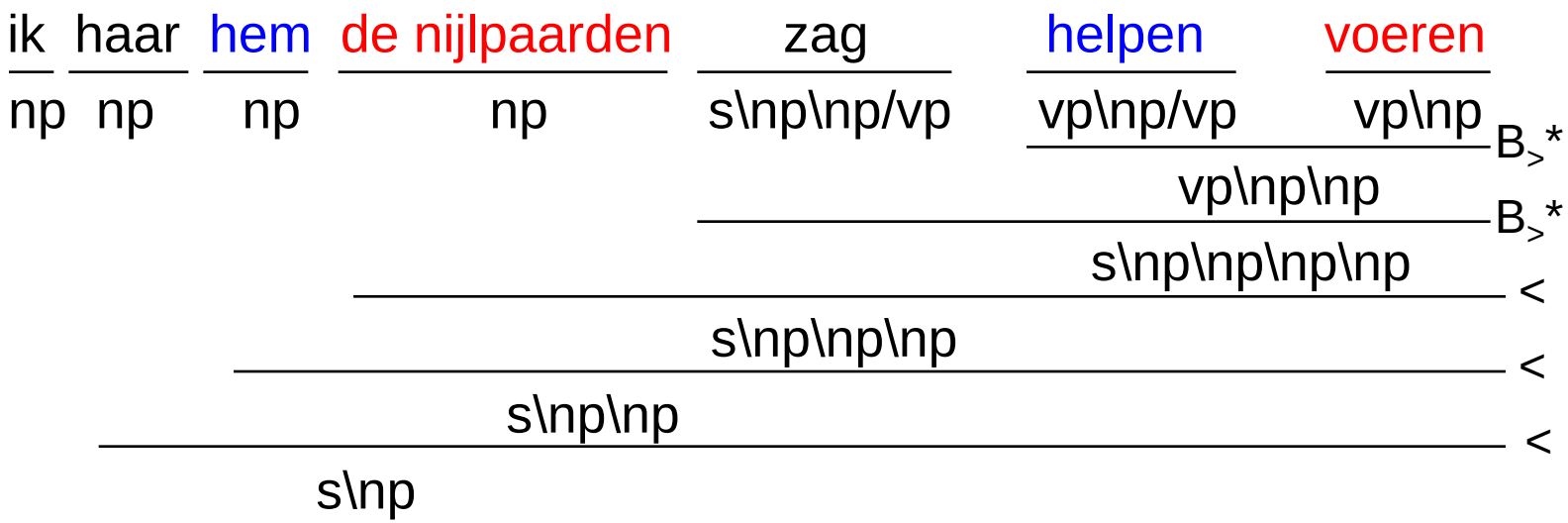
<: y x\y ⇒ x

CCG: Analyse

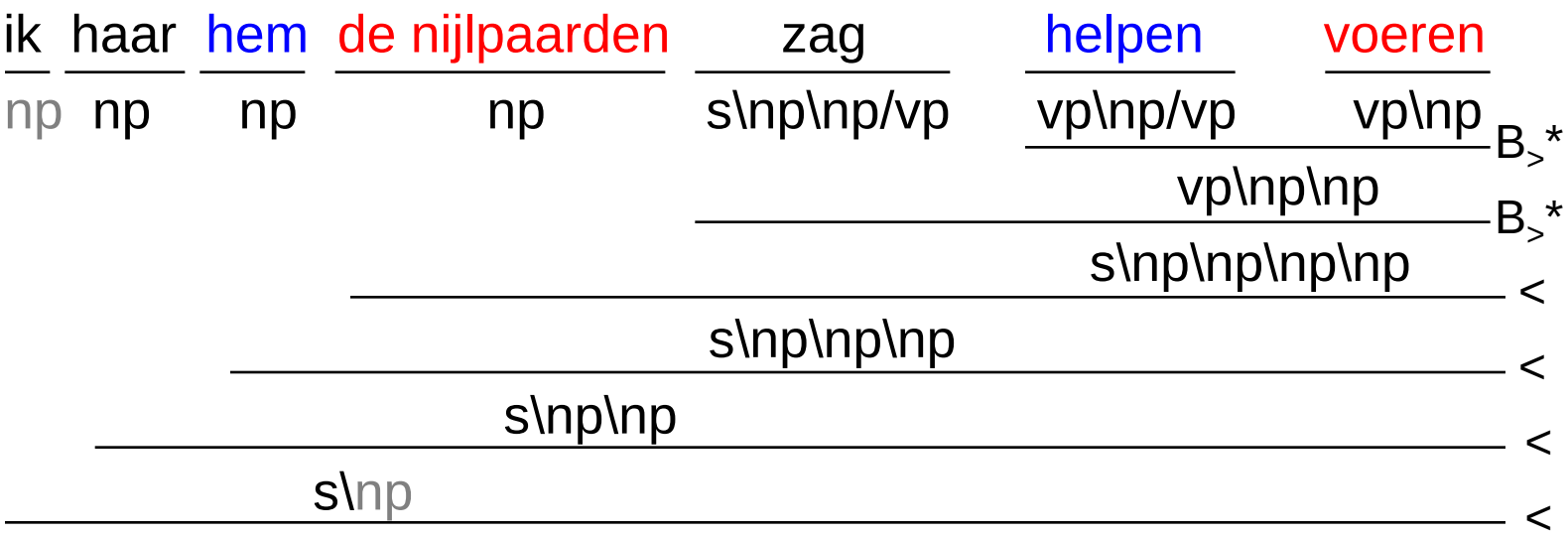


<: y x\y ⇒ x

CCG: Analyse

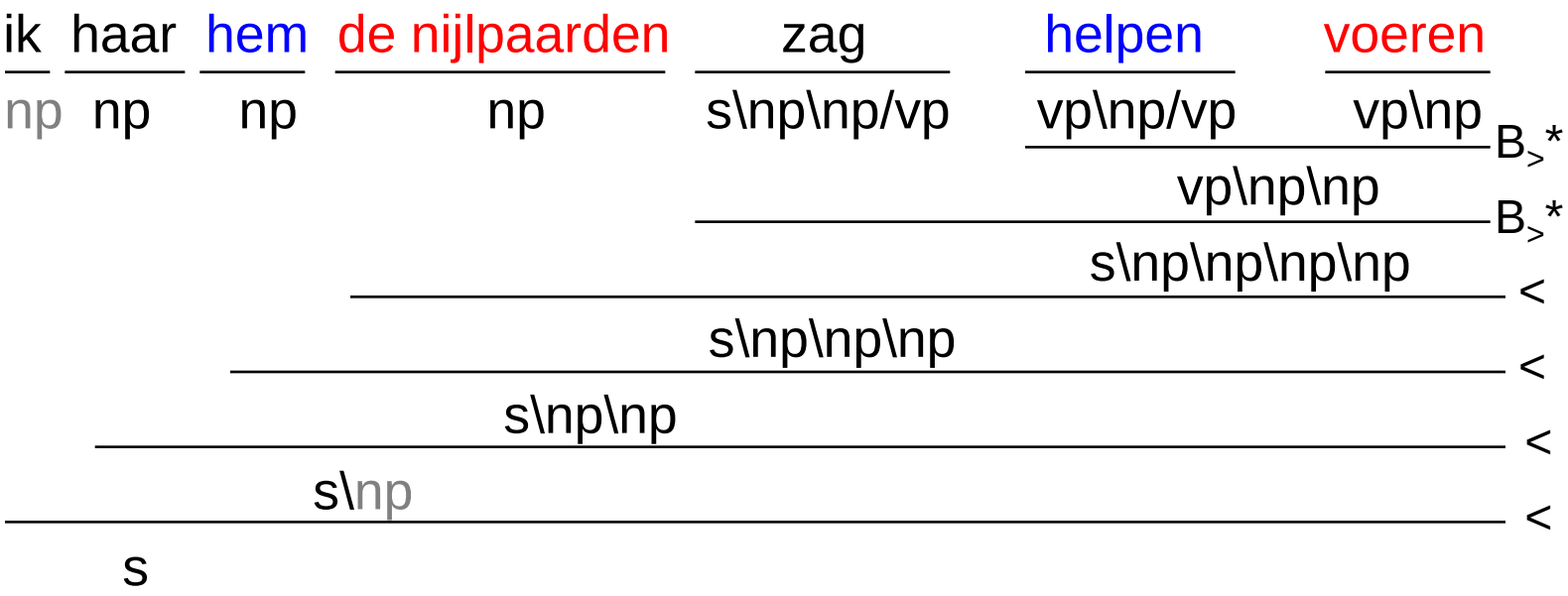

ik haar hem de nijlpaarden zag helpen voeren
np np np np s\np\np/vp vp\np/vp vp\np
B>*
vp\np\np
B>*
s\np\np\np\np
<
s\np\np\np
<
s\np\np
<
s\np

CCG: Analyse



<: y x\y ⇒ x

CCG: Analyse



<: y x\y ⇒ x

LIG: Beschreibung

- (Aho 1968) (Gazdar 1988)

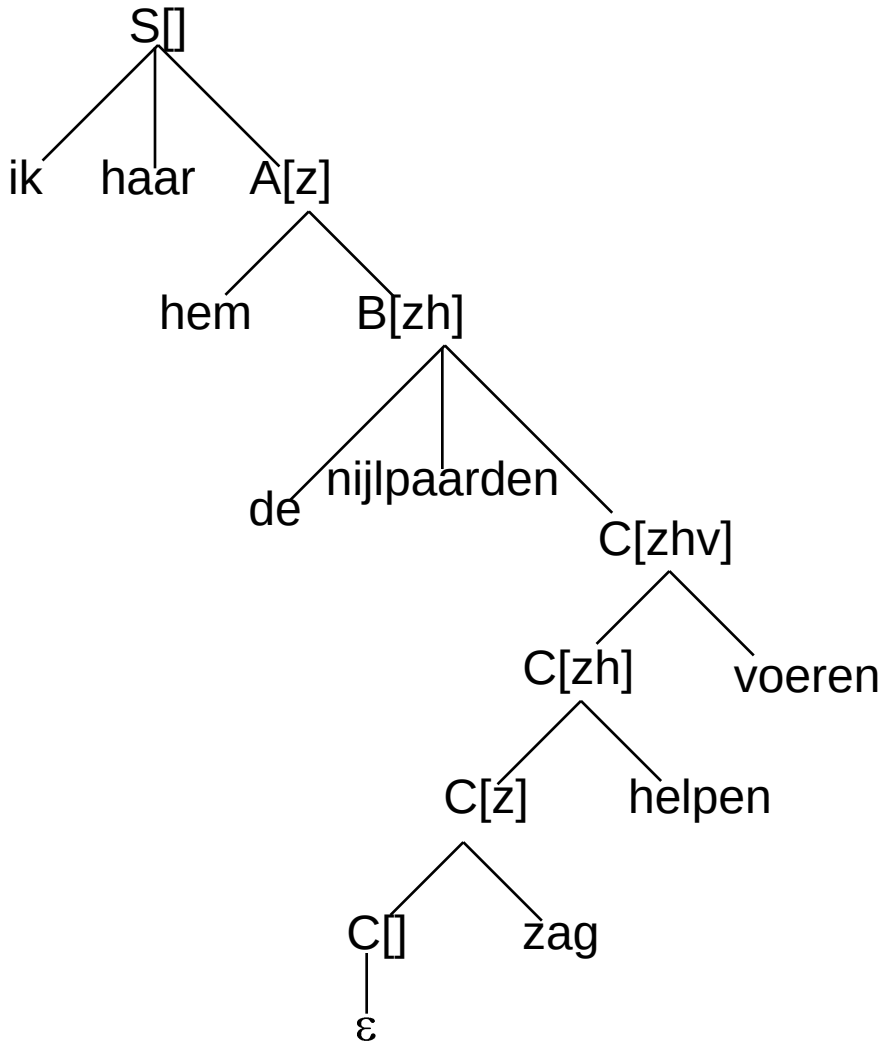
- wie kontextfreie Grammatik plus Indizes an Nonterminalen
  - Indizes beschreiben den Stack
  - im Unterschied zur IG nach Aho werden Indizes nur an ein Nonterminal weitergegeben

LIG: Beispielgrammatik

- Regeln:
  - S[..] → ik haar A[..z]
  - A[..] → hem B[..h]
  - B[..] → de nijlpaarden C[..v]

  - C[..v] → C[..] voeren
  - C[..h] → C[..] helpen
  - C[..z] → C[ ] zag
  - C[ ] → ε

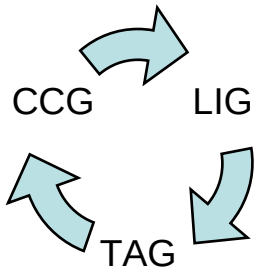
LIG: Analyse


S[]
ik
haar
A[z]
hem
B[zh]
de
nijlpaarden
C[zhv]
C[zh]
voeren
C[z]
helpen
C[]
zag
ε

Äquivalenzbeweis CCG, LIG, TAG

zeige die Äquivalenz in Bezug auf die Stringsprachen der Grammatikformalismen CCG, LIG und TAG durch zyklische Inklusion (Vijay-Shanker und Weir 1994):

- zeige CCG $\subseteq$ LIG
  zu einer CCG gibt es eine LIG, die die gleiche Sprache erzeugt
- zeige LIG $\subseteq$ TAG
  zu einer LIG gibt es eine TAG, die die gleiche Sprache erzeugt
- zeige TAG $\subseteq$ CCG
  zu einer TAG gibt es eine CCG, die die gleiche Sprache erzeugt



Beweis: CCG → LIG

Ideen zur Überführung einer CCG in eine LIG

- CCG-Kategorien können interpretiert werden als Nonterminal +Stack (z.B. S $\Rightarrow$ S[], S/a $\Rightarrow$ S[/a])
- Applikation entspricht push
- Komposition entspricht pop

Beweis: CCG → LIG

Konstruktion der LIG zur bestehenden CCG

- alle Kategorien des Lexikons werden als entsprechende LIG-Produktion angelegt
- zum Beispiel zag := s\np\np/vp
  - Produktionen aus den Instanzen der Rückwärtsapplikation

    np x\np → np S[..] → np S[..\np]

  - Produktionen aus den Instanzen der Vorwärtskreuzkomposition

    x/vp vp\(np/vp) → x\(np/vp) S[..\np/vp] → S[../vp] vp[\np/vp]

Beweis: LIG → TAG

Idee für den Beweis:

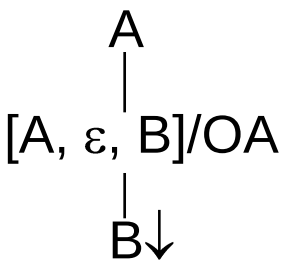
- zunächst LIG normalisieren
  - jede Regel pusht und poppt maximal ein Item auf den/vom Stack
  - alle Stacks sind zu Anfang und am Ende einer Ableitung leer

- Konstruktion der TAG
  - TAG-Nichtterminale kodieren die Stackoperationen
  - Adjunktionsknoten werden mit elementaren Stackoperationen markiert
    - Eingabe-Nonterminal
    - Transitionstyp (keine Transition, pushen oder poppen)
    - Ausgabe-Nonterminal

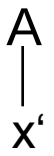
Beweis: LIG → TAG

Konstruktion der TAG

- Initialbäume

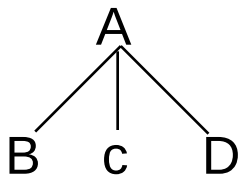


für alle Nonterminale A, B



für Regeln der Form A[ ] → x

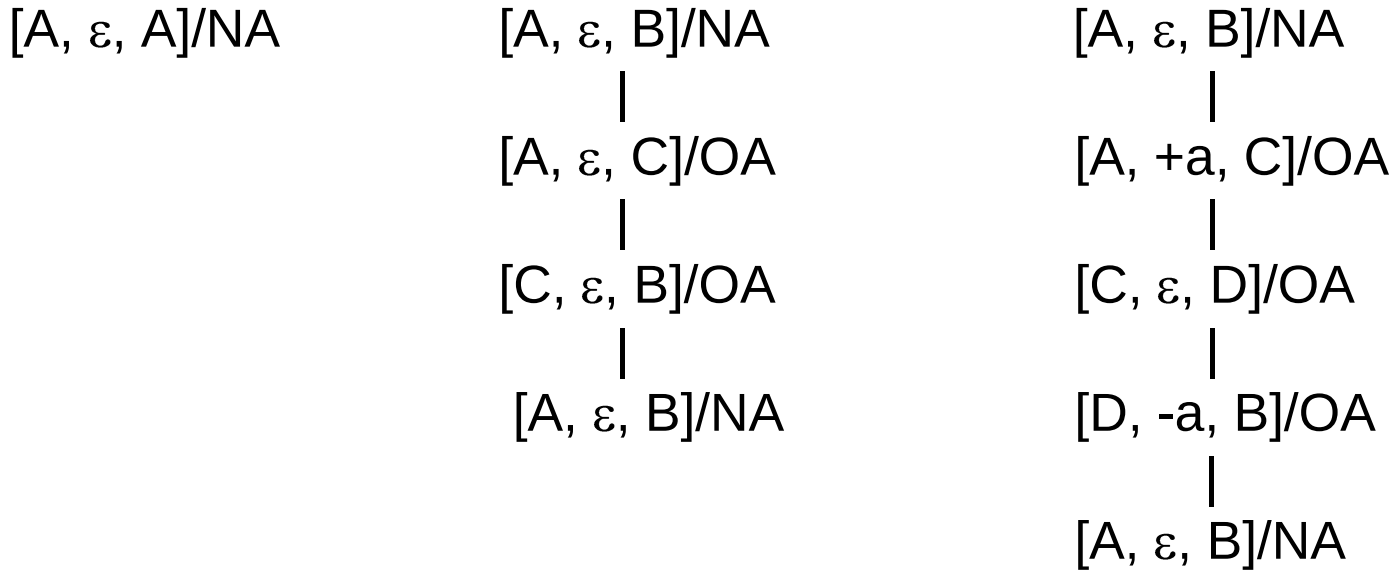
Beispiel für A[ ] → B [ ] c D[ ]



Beweis: LIG → TAG

Konstruktion der TAG

- Auxiliarbäume: für alle Nonterminale A, B, C, D und alle Stacksymbole a:

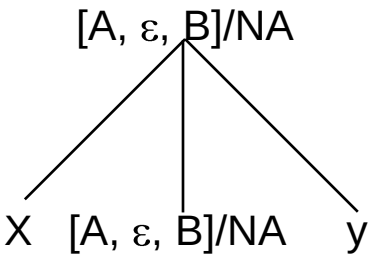


Beweis: LIG → TAG

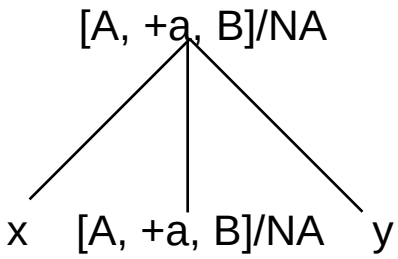
Konstruktion der TAG

- weitere Auxiliarbäume für LIG-Regeln

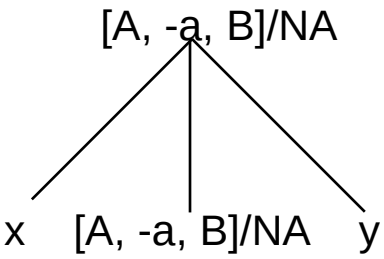
keine Stack-OP:
A[..] → x B[..] y



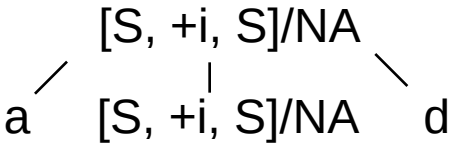
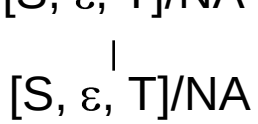
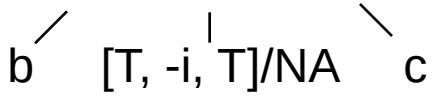
push:
A[..] → x B[..a] y



pop:
A[..a] → x B[..] y

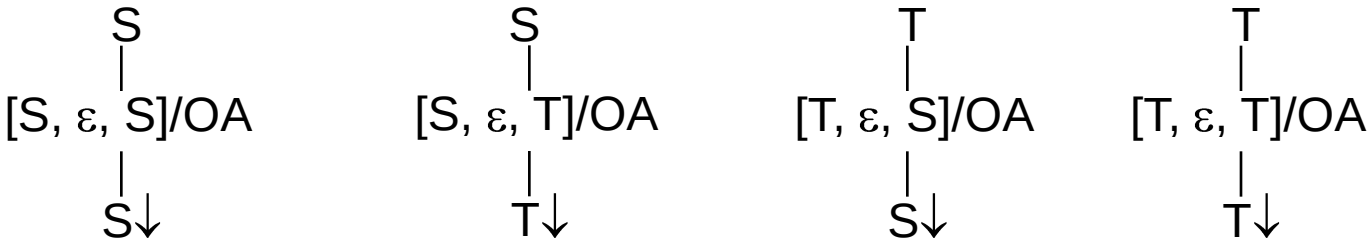


Beweis: LIG → TAG Beispiel

LIG-Regel ($a^n b^n c^n d^n$)		TAG-Regel
S[..] → a S[..i] d	⇒	
S[..] → T[..]	⇒	
T[..i] → b T[..] c	⇒	
T[ ] → ε	⇒	

Beweis: LIG → TAG Beispiel

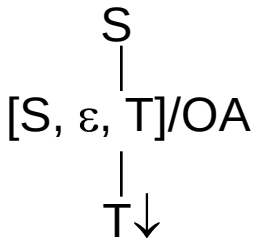
Initialbäume für Nonterminale S und T

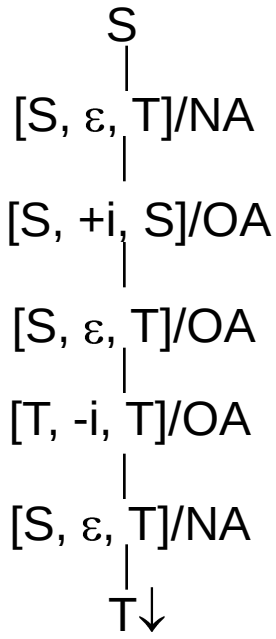


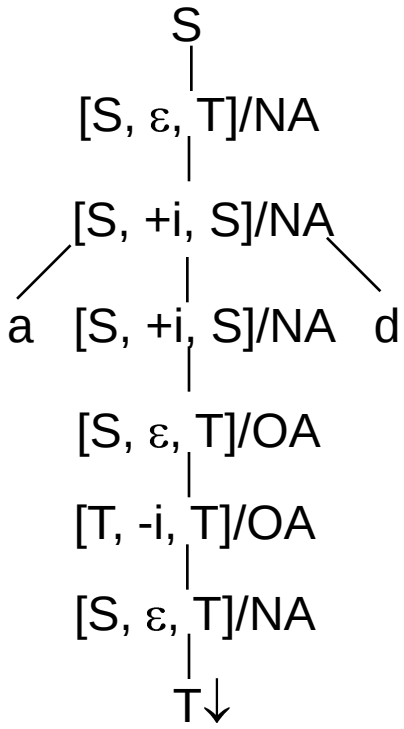
Die Auxiliarbäume werden nicht angegeben, da durch die Konstruktion zu viele entstehen. Für zwei Nonterminale und ein Stacksymbol werden bereits $2^4 = 16$ Stack-Auxiliarbäume erzeugt.

Beweis: LIG → TAG Beispiel

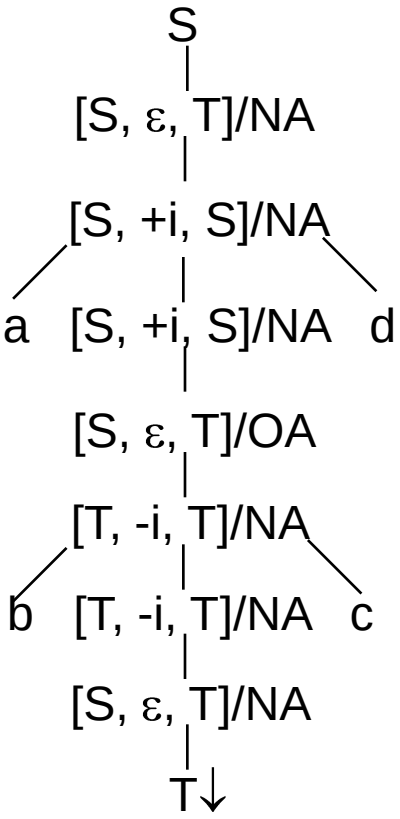
Ableitung für das Wort aabbccdd

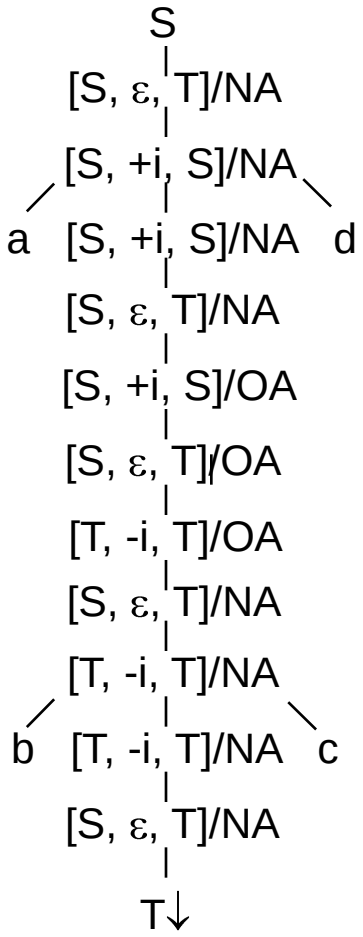




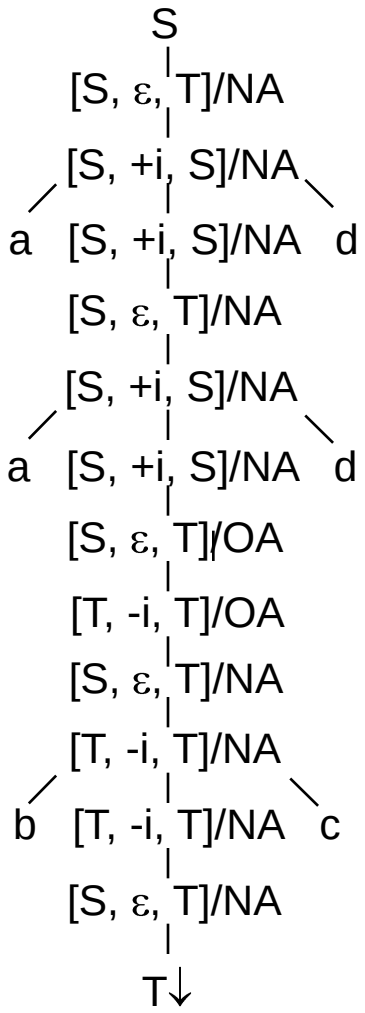


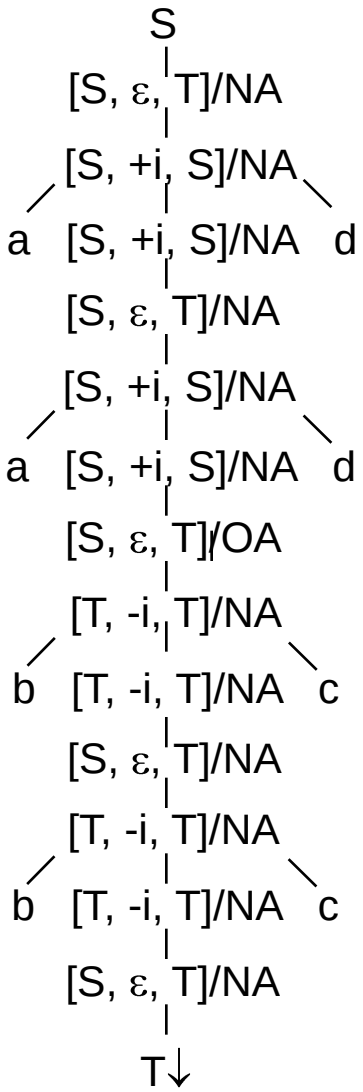
Beweis: LIG → TAG Beispiel



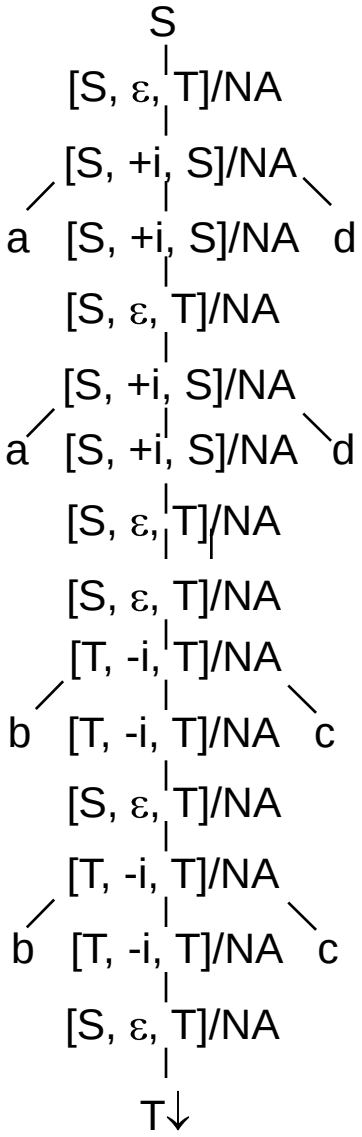


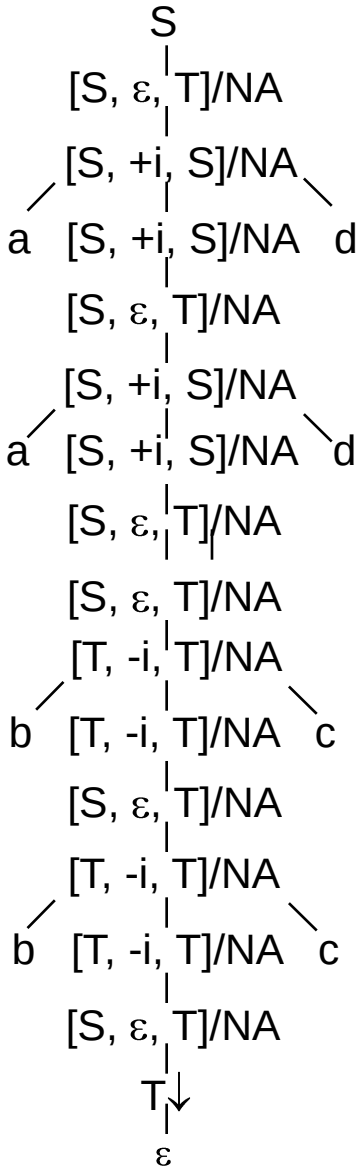
Beweis: LIG → TAG Beispiel





Beweis: LIG → TAG Beispiel

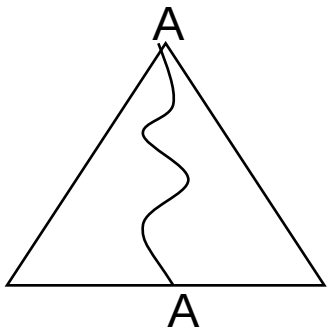




Beweis: TAG → CCG

Idee zur Überführung:

- Adjunktion entspricht Komposition
- TAG muss in Normalform vorliegen, d.h.
  - höchstens binär verzweigend
  - alle inneren Knoten sind entweder NA oder OA
  - alle OA-Knoten liegen auf dem spine oder sind Schwesterknoten vom spine
  - alle Schwesterknoten vom Spine haben eine einzige Tochter: das leere Wort



- alle TAGs können in Normalform überführt werden

Beweis: TAG → CCG

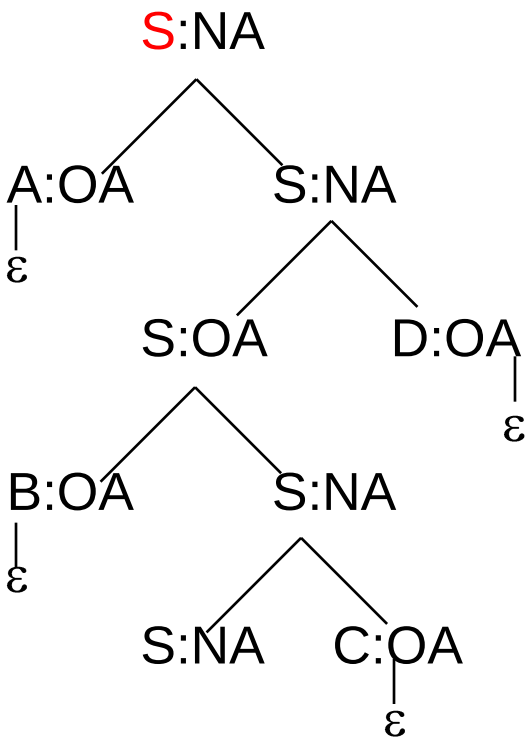
Algorithmus zur Überführung TAG nach CCG (überführt nur einzelne Auxiliarbäume nach CCG)

- Initialisierung:
  - pos = root (t) = A
  - c = A oder c = Â
- bis das Blatt des Baumes t erreicht ist:
  - wenn die Non-spine-Tochter von pos linke Tochter mit Marke B/OA ist, ergänze c um /B
  - wenn die Non-spine-Tochter von pos rechte Tochter mit Marke B/OA ist, ergänze c um \B
  - wenn die spine-Tochter von pos Marke C/OA hat, ergänze c um Ĉ
  - pos = spine-Tochter von pos

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



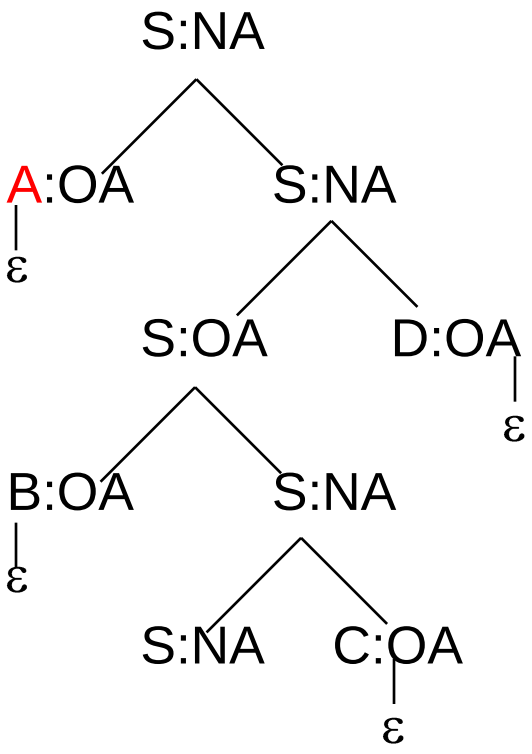
S

$\hat{S}$

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



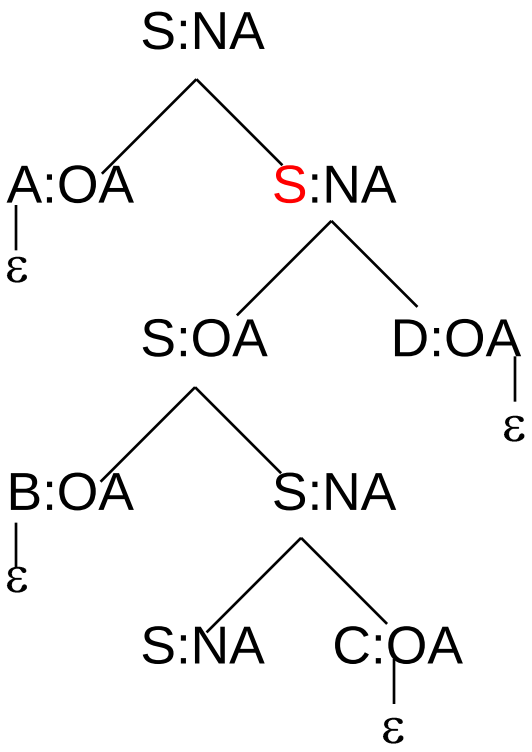
S\A

$\hat{S}\backslash A$

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



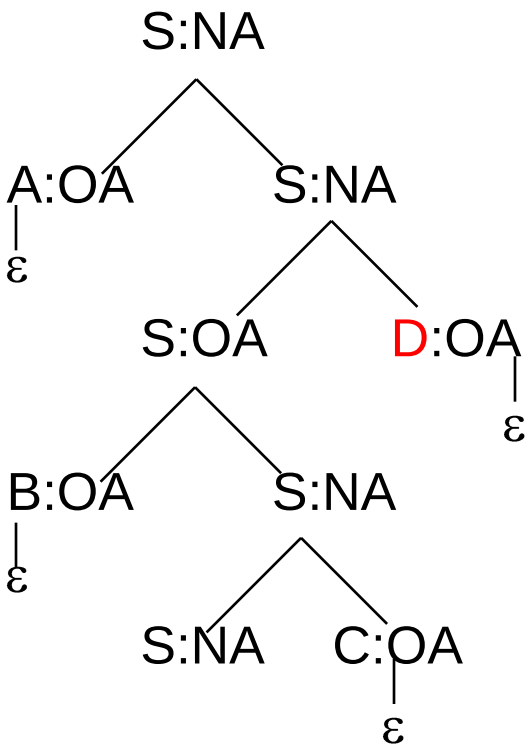
S\A

$\hat{S}\backslash A$

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



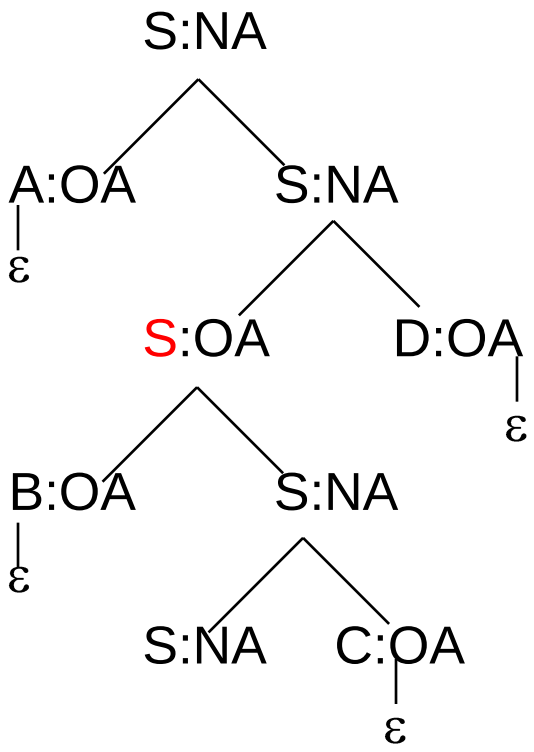
S\A/D

$\hat{S}$\A/D

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



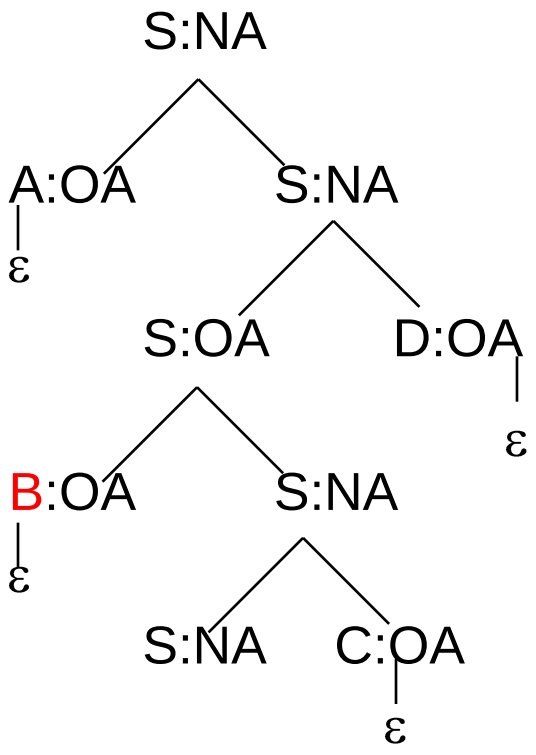
S\A/D/Ŝ

Ŝ\A/D/Ŝ

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



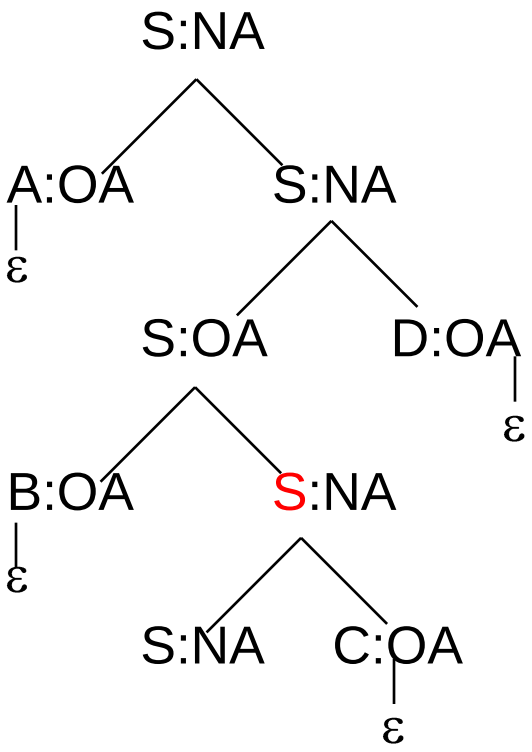
S\A/D/$\hat{S}$\B

$\hat{S}$\A/D/$\hat{S}$\B

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



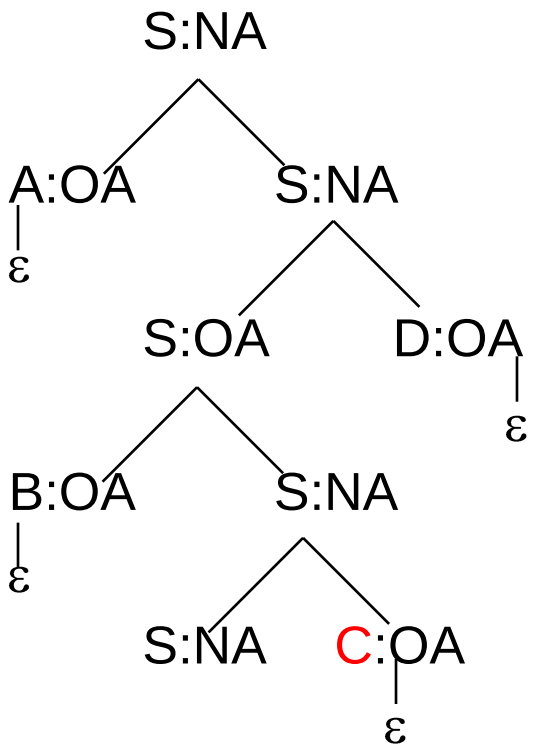
S\A/D/$\hat{S}$\B

$\hat{S}$\A/D/$\hat{S}$\B

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



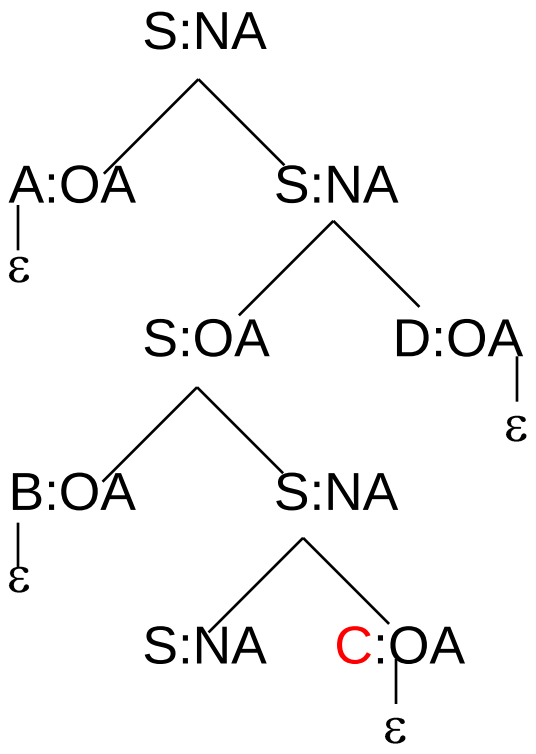
S\A/D/$\hat{S}$\B/C

$\hat{S}$\A/D/$\hat{S}$\B/C

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



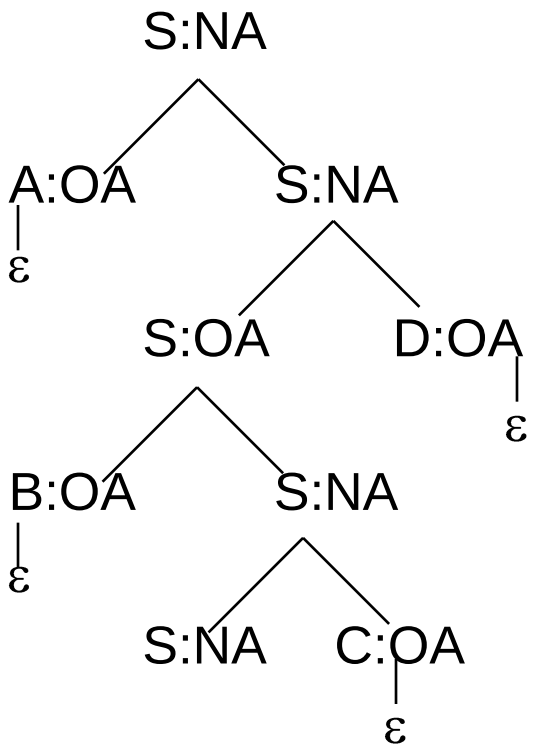
S\A/D/$\hat{S}$\B/C

$\hat{S}$\A/D/$\hat{S}$\B/C

Beweis: TAG → CCG

Algorithmus zur Überführung TAG nach CCG

- Beispiel:



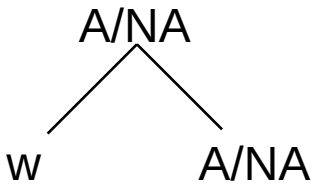
S\A/D/$\hat{S}$\B/C

$\hat{S}$\A/D/$\hat{S}$\B/C

Beweis: TAG → CCG

Konstruktion einer CCG aus einer TAG in Normalform:

- Lexikon:
  - für jeden lexikalischen Auxiliarbaum der Form



    konstruiere einen Lexikoneintrag der Form w := A

  - für jede aus einem nicht-lexikalischen Auxiliarbaum gewonnene Kategorie c (Algorithmus) konstruiere ein Lexikoneintrag ε := c

Beweis: TAG → CCG

Konstruktion einer CCG aus einer TAG in Normalform

- Regeln
  - Vorwärts- und Rückwärtsapplikation modellieren Adjunktion von Auxiliarbäumen in Spine-Schwestern:

    $x/A\ A \rightarrow x$

    $A\ x\backslash A \rightarrow x$

    für jedes Nonterminal A

  - Komposition modelliert die Adjunktion von nicht-lexikalischen Bäumen in das Spine:

    für jedes Nonterminal A und $i \leq n$, wobei n die Länge des längsten Spine eines Auxiliarbaums der TAG ist:

    $x/\hat{A}\ \ \hat{A}|_1 z_1|_2 \ldots .|_i z_i \rightarrow\ x|_1 z_1|_2 \ldots |_i z_i \qquad (| \in \{/, \backslash\})$

  - Dachkategorien stellen sicher, dass nur nicht-lexikalische Auxiliarbäume in den Spine von anderen nicht-lexikalischen Auxiliarbäumen adjungiert werden dürfen

Schluss

- schwach kontextsensitive Grammatikformalismen sind eine leichte Erweiterung der kontextfreien Grammatiken mit dem Ziel, nicht kontextfreie Phänomene (z.B. kreuzende Abhängigkeiten im Holländischen) beschreiben zu können, aber dennoch effizient sind
- vorgestellt wurden TAG, CCG und LIG
- die drei Grammatikformalismen beruhen auf völlig unterschiedlichen Paradigmen:
  - TAG: Baumersetzungssystem
  - CCG: Kategorialgrammatik
  - LIG: CFG mit Stack
- der Beweis von Vijay-Shanker und Weir (1994) zeigt, dass alle drei äquivalent bzgl. Ihrer Stringsprachen sind
- hier nicht gezeigt: Alle drei können in n^6 geparst werden

  (Vijay-Shanker und Weir 1993)

Literatur

- Gerhard Jäger & Jens Michaelis (Sommerschule ESSLLI 2004 Nancy)

  An Intoduction to mildly context sensitive grammar formalisms
- Klaus von Heusinger: Kategoriale Unifikationsgrammatik 2005
- Volker Kaatz: Linear indizierte Grammatiken und Sprachen 2005
- K. Vijay-Shanker, David J. Weir 1994: The Equivalence of Four Extensions of Context-free Grammars. Mathematical Systems Theory 27(6): 511-546

weitere Literaturreferenzen

- Alfred V. Aho, 1968. Indexd Grammars and extension of context-free grammars. Journal of the Accociation for Computing Machinery 15, 647-671.
- Bar-Hillel, Yehoshua 1953: A Quasi-Artithmetical Notation for Syntactic Description Language, 29, 4758
- Gerald Gazdar, 1988. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, Natural Language Parsing and Linguistic Theories, pages 69-94. Doredrecht: Reidel.
- Aravind K. Joshi, Leon S. Levy, Masako Takahashi, 1975: Tree Adjunct Grammars. Journal of Computer and System Science. 10(1): 136-163
- Aravind K. Joshi, K. Vijay-Shanker, 1985: Some Computional Properties of Tree Adjoining Grammars. ACL 1985: 82-93
- Mark Steedman 2000: The Syntactic Process. Cambridge, MA, MIT Press.
- K. Vijay-Shanker, David J. Weir 1993: Parsing Some Constrained Grammar Formalisms. Computional Linguistics 19(4): 591-636

Anhang I: Definition TAG I

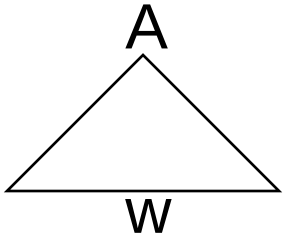
$G = <V_N, V_T, T_{ini}, T_{aux}, S>$ mit

- V_N eine endl. Menge von Nonterminalen
- V_T eine endl. Menge von Terminalen
- T_{ini} eine endl. Menge von Initialbäumen
- T_{aux} eine endl. Menge von Auxiliarbäumen
- $S \in V_N$ ein unterscheidbares Nonterminalsymbol (Start-Symbol)

Anhang II: Definition TAG II

- Initialbäume:

  t ist ein endl. markierter Baum $< N_t, \lhd^*_t, \prec_t, \text{Marke}_t >$ mit

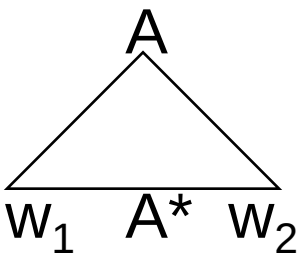


$A \in V_N$

$w \in \text{Strings}\,(V_N \{\downarrow\} \cup V_T)$

- Auxiliarbäume:

  t ist ein endl. markierter Baum $< N_t, \lhd^*_t, \prec_t, \text{Marke}_t >$ mit



$A \in V_N$

$w_1, w_2 \in \text{Strings}\,(V_N \{\downarrow\} \cup V_T)$

Anhang III:Definition CCG I

CCG G = $<V_T, V_N, S, f, R>$

- V_T endl. Menge von Terminalen (lexikalische Items)
- V_N endl. Menge von Nonterminalen (atomare Kategorien)
- V_N und V_T disjunkt
- $S \in V_N$
- f Funktion, die Elemente aus $V_T \cup \{\varepsilon\}$ auf endl. Submengen von $C(V_N)$ mapped mit
  - $C(V_N)$: Menge der Kategorien mit $V_N \subseteq C(V_N)$
  - $c_1, c_2 \in C(V_N) \rightarrow (c_1/c_2) \in C(V_N)$ und $(c_1\backslash c_2) \in C(V_N)$
- R endl. Menge von Kombinationsregeln

Anhang IV:Definition CCG II

vier Typen von Kombinationsregeln

X, y, z_1, … Variablen über $C(V_N)$, $|_i \in \{/,\backslash\}$

- Vorwärtsapplikation
  $(x/y)y \rightarrow x$
- Rückwärtsapplikation
  $y(x\backslash y) \rightarrow x$
- generalisierte Vorwärtskomposition für $n \geq 1$:
  $(x/y)\ (\ldots(y\ |_1\ z_1)_2\ \ldots|_n\ z_n) \rightarrow (\ldots(x_1\ |_1\ z_1)\ |_2\ \ldots\ |_n\ z_n)$
- generalisierte Rückwärtskomposition für $n \geq 1$:
  $(\ldots(y\ |_1\ z_1)_2\ \ldots|_n\ z_n)\ (x\backslash y) \rightarrow (\ldots(x_1\ |_1\ z_1)\ |_2\ \ldots\ |_n\ z_n)$

Anhang V: Definition LIG

IG $G = (V_N, V_T, V_S, S, P)$

- V_N endl. Menge von Nonterminalen
- V_T endl. Menge von Terminalen
- V_S endl. Menge von Stacksymbolen
- V_N, V_T, V_S disjunkt
- $S \in V_N$ Startsymbol
- P endl. Menge von Produktionen der Form $A[..x] \rightarrow \alpha_1 \ldots \alpha_n$
  - $X \in V_S^*$
  - für alle $1 \leq i \leq n$, $\alpha_i = A[..y]$, $\alpha_i = A[z]$ oder $\alpha_i = \omega$ mit $A \in V_N$, $\omega \in V_T^*$ und $y,z \in V_S^*$

Nur ein Nonterminal auf der rechten Seite darf den Stack von der rechten Seite betreten

Anhang VI: Definition MCFG

$G = < V_N, V_O, F, R, S>$ mit

- G ist generalisierte CFG
- $V_O = \bigcup_{i \geq 1}^{m}$ Strings $(V_T)^i$, V_T Menge von Terminalen
- Für jedes $f \in F$ gibt es Zahlen n und $d_1,\ldots,d_n$, r mit
  f: Strings $(V_T)^{d_1}$ x … x Strings $(V_T)^{d_n} \rightarrow$ Strings $(V_T)^r$
  f ist total und regulär

Anhang VII: Definition LCFRS

$G = < V_N, V_O, F, R, S >$ mit

- V_N Menge von Nonterminalen
- V_O Menge von (linguistischen Objekten)
- $F \subseteq \bigcup_{n \geq 0} F_n$, Menge von Funktionen mit
  - F_n Menge nichtleerer part. Funktionen von V_O^n nach V_O
  - F_0 Menge aller Konstanten in V_O
- $R \subseteq \bigcup_{n \geq 0} (F \cap F_n) \times V_N^{n+1}$, endl. Menge von (beschreibenden) Regeln
- $S \in V_N$ ein unterscheidbares Nonterminal (Startsymbol)