

# **CCG III:** **W. modelle und Parsing**

Vorlesung “Grammatikformalismen”  
Alexander Koller

21. Mai 2019

# CCG: Beispiel

Lexikon:

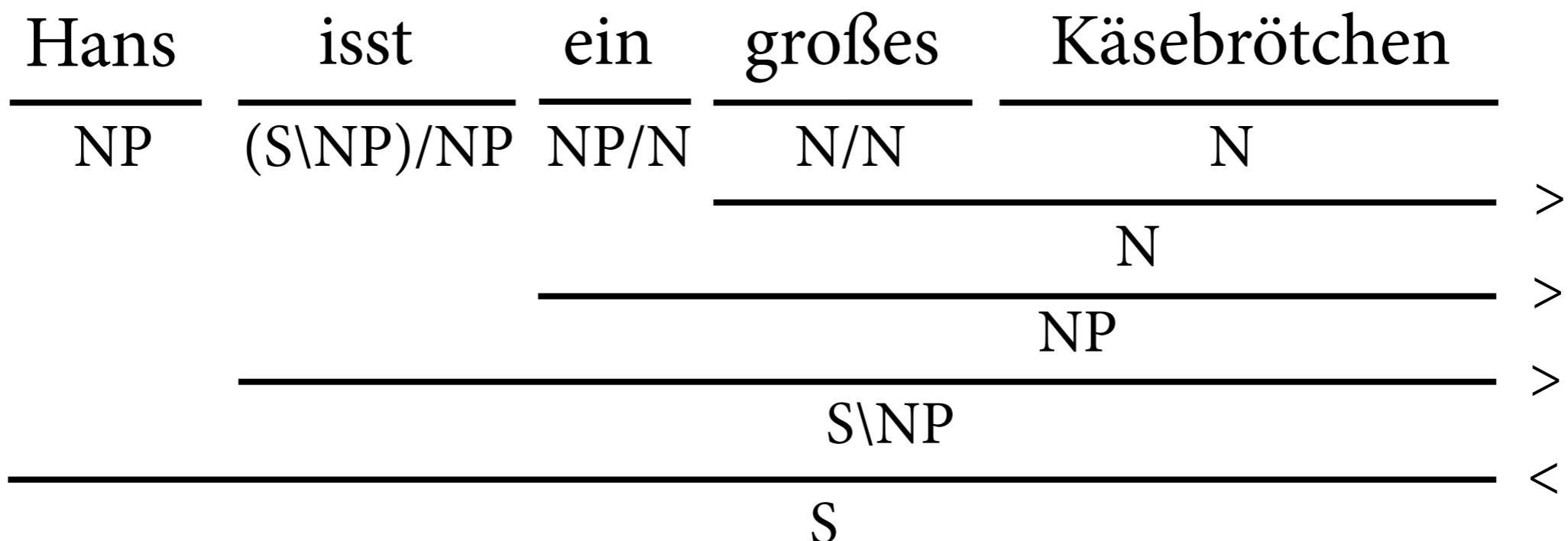
Hans: NP

ein: NP/N

isst: (S\NP)/NP

Käsebrötchen: N

großes: N/N



# CCG: Beispiel

Lexikon:

John: NP

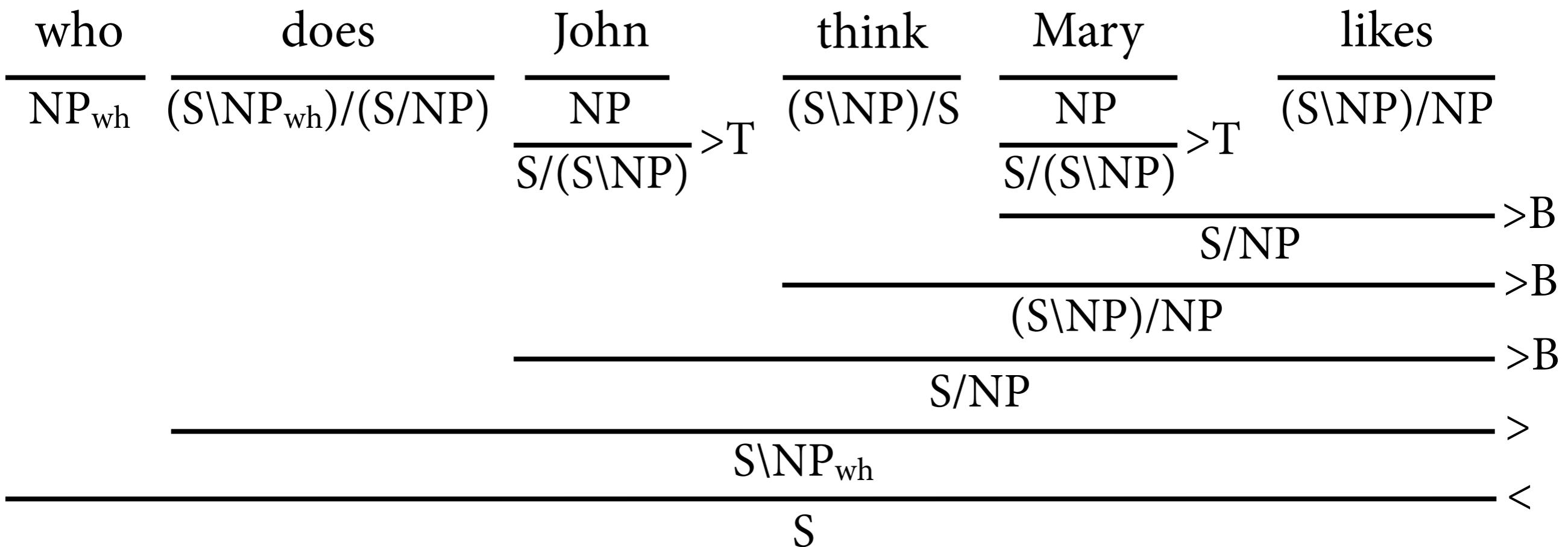
think: (S\NP)/S

Mary: NP

likes: (S\NP)/NP

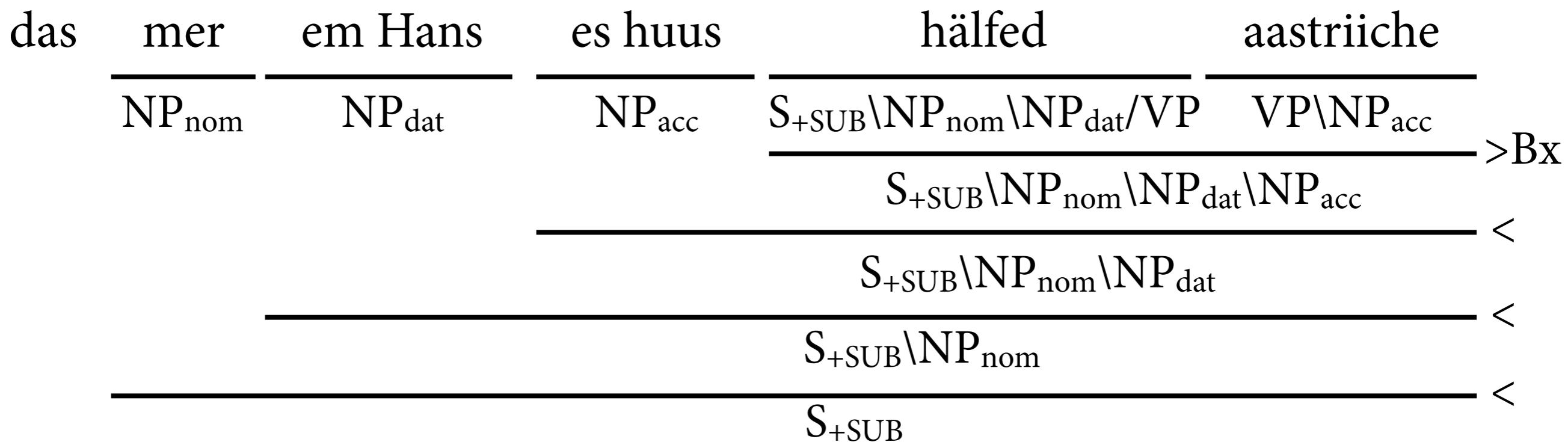
who: NP<sub>wh</sub>

does: (S\NP<sub>wh</sub>)/(S/NP)



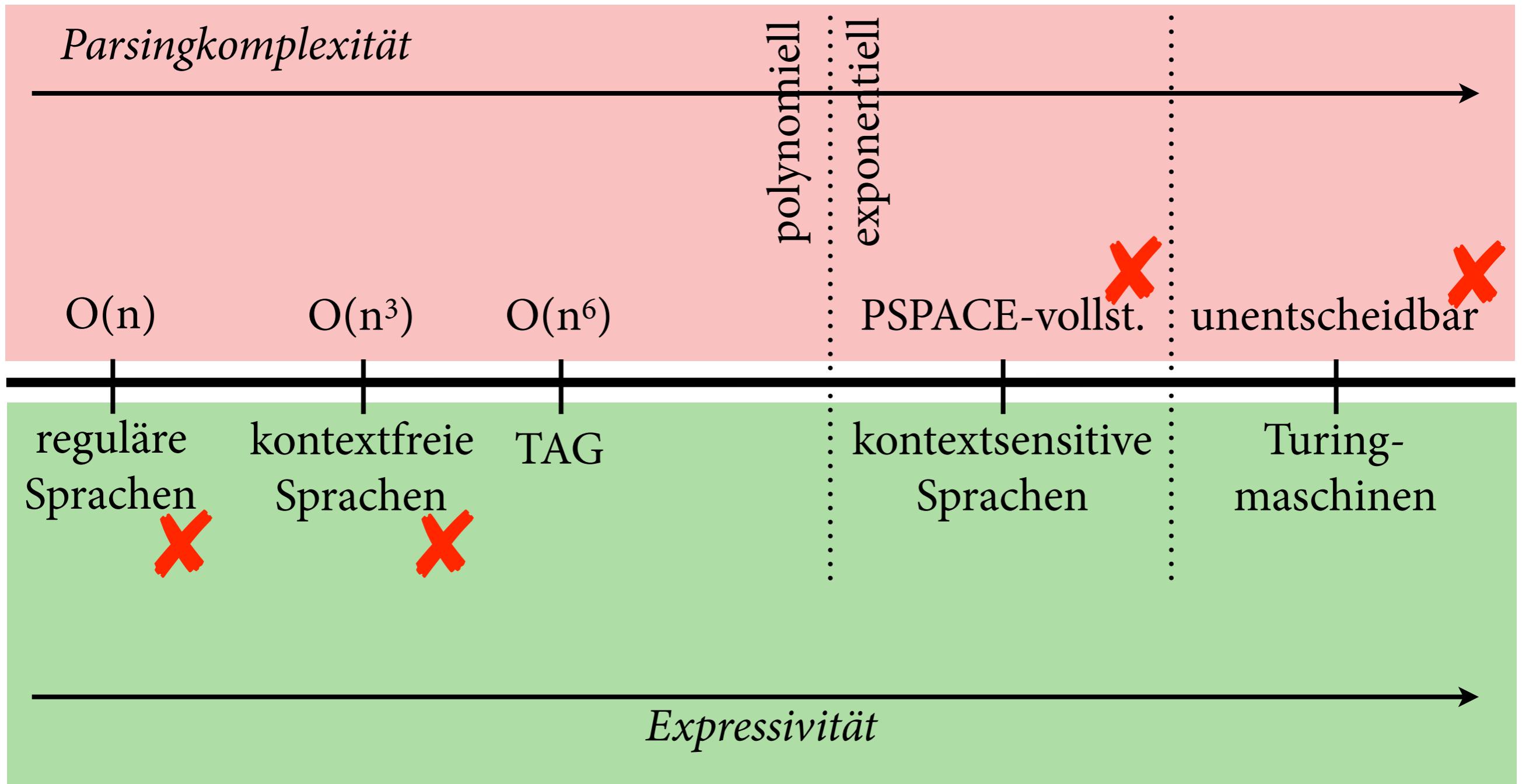
# Schweizerdeutsch

- Crossed composition erlaubt Modellierung von cross-serial dependencies:

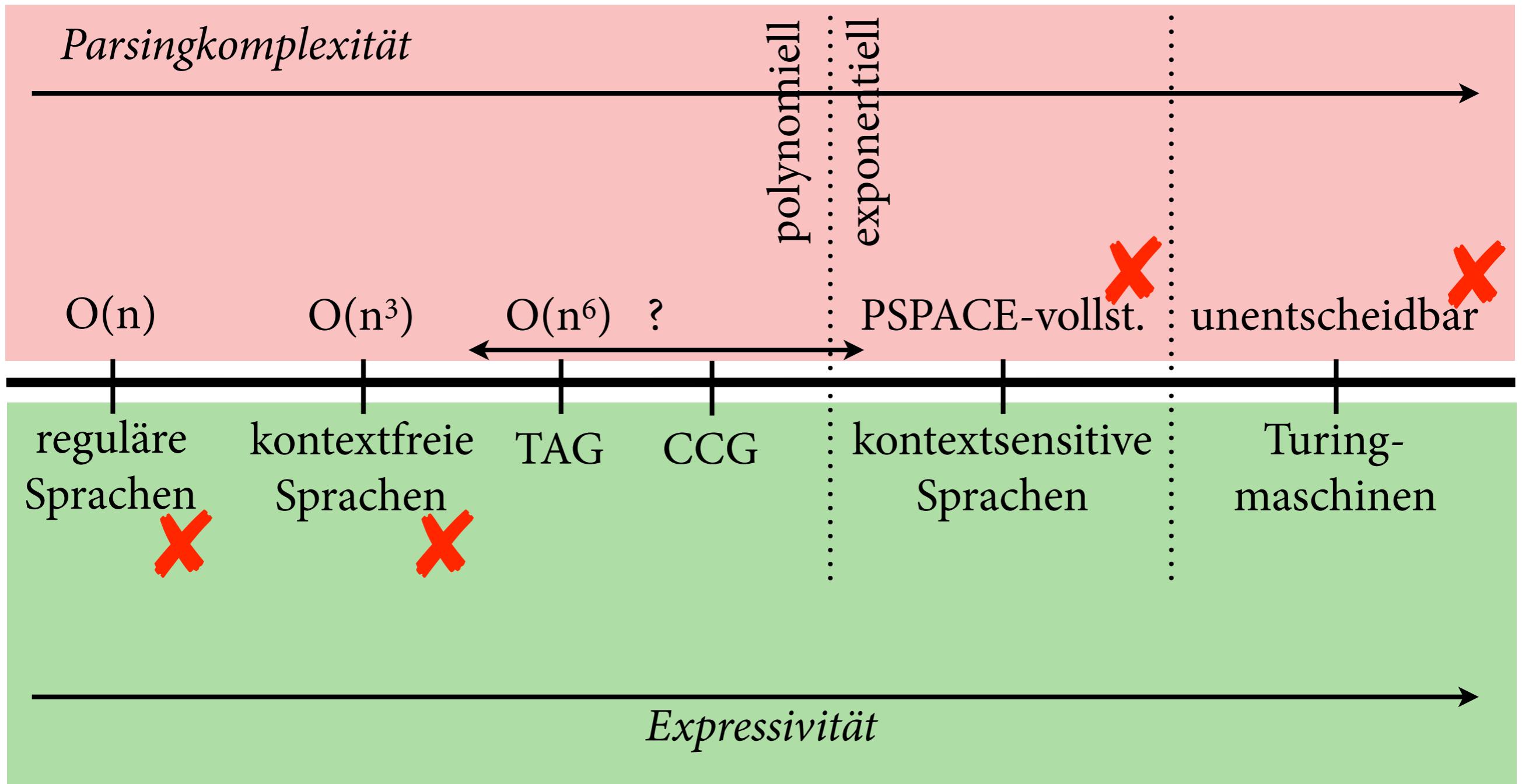


(“VP” = Abkürzung für S\NP<sub>nom</sub>)

# Grammatikformalismen



# Grammatikformalismen



# Heute

- Polynomieller CKY-Algorithmus für CCG.
- Log-lineares W.modell: Der C&C-Parser.
- Supertag-factored W.modell: Der EasyCCG-Parser.

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen  
NP            (S\NP)/NP    NP/N       N/N                    N

0					
1					
2					
3					
4					
5					
1					
2					
3					
4					
5					

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen  
NP            (S\NP)/NP    NP/N       N/N                    N

NP					
0					
1					
2					
3					
4					
5					
1					
2					
3					
4					
5					

# CKY-Parsing für CCG

Hans            isst            ein            großes            Käsebrötchen  
NP            (S\NP)/NP            NP/N            N/N            N

NP	S/(S\NP)				
0					
1					
2					
3					
4					
5					
1					
2					
3					
4					
5					

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N

---

N

NP S/(S\NP) S\(S/NP)				
1	2	3	4	5

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N

---

N

0	NP S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			
2					
3					
4					
1					

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			
2			NP/N		
3					
4					

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			
2			NP/N		
3				N/N	
4					

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			
2			NP/N		
3				N/N	
4					N

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			
2			NP/N		
3				N/N	N
4					N

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			
2			NP/N		NP
3				N/N	N
4					N

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP      S/(S\NP) S\/(S/NP)				
1		(S\NP)/NP			S\NP
2			NP/N		NP
3				N/N	N
4					N

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				S
1		(S\NP)/NP			S\NP
2			NP/N		NP
3				N/N	N
4					N
5	1	2	3	4	5

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				S
1		(S\NP)/NP			S\NP
2			NP/N	NP/N	NP
3				N/N	N
4					N

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)				S
1		(S\NP)/NP		(S\NP)/N	S\NP
2			NP/N	NP/N	NP
3				N/N	N
4					N
5	1	2	3	4	5

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP S/(S\NP) S\/(S/NP)	S/NP	S/N	S/N	S
1		(S\NP)/NP	(S\NP)/N	(S\NP)/N	S\NP
2			NP/N	NP/N	NP
3				N/N	N
4					N

# CKY-Parsing für CCG

Hans      isst      ein      großes      Käsebrötchen

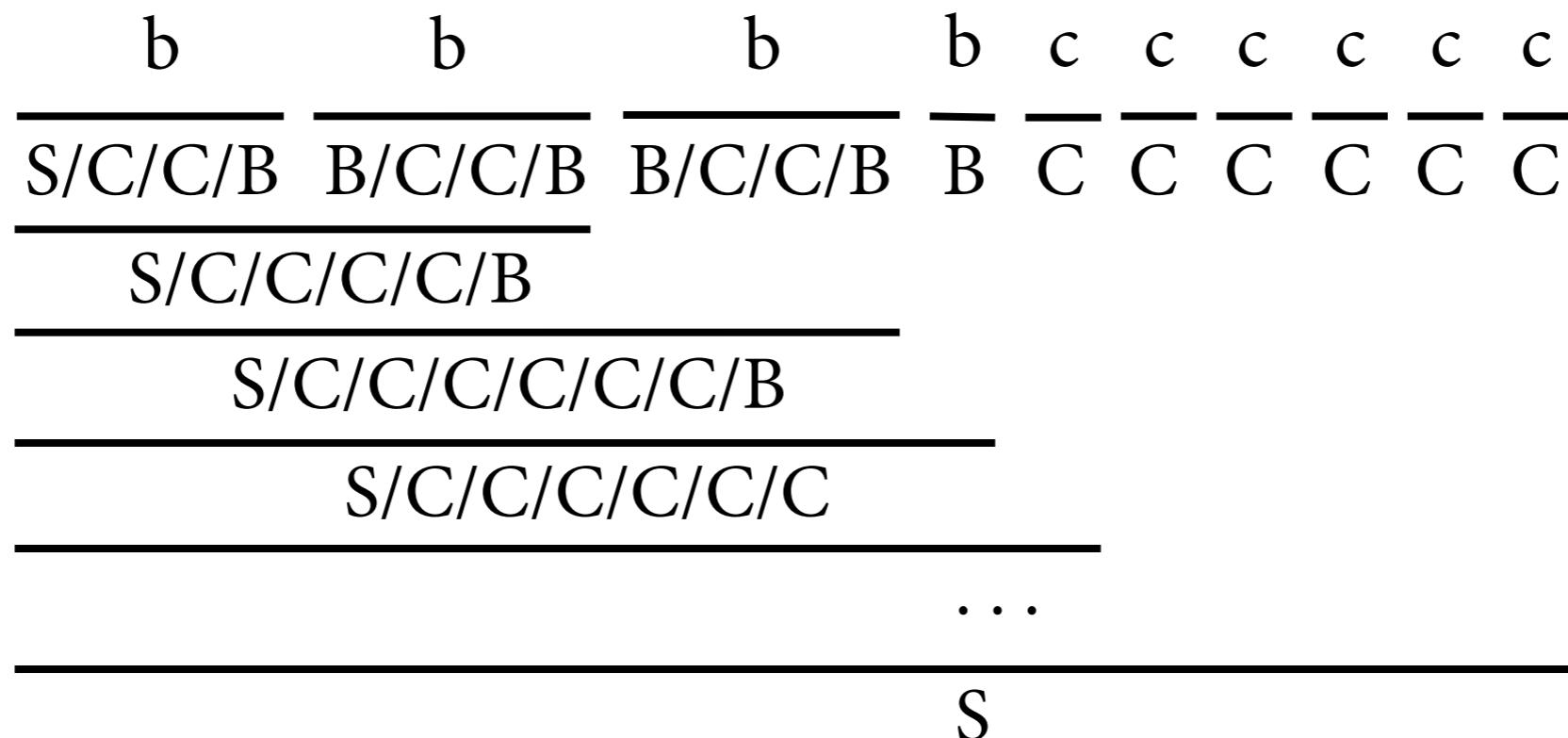
---

NP      (S\NP)/NP      NP/N      N/N      N

0	NP      S/(S\NP) S\/(S/NP)	S/NP	S/N	S/N	S
1		(S\NP)/NP	(S\NP)/N	(S\NP)/N	S\NP
2			NP/N	NP/N	NP
3	Wie viele Kategorien pro Chart-Zelle?			N/N	N
4					N

# Das Problem

- Kategorie eines Ausdrucks der Länge n kann  $O(n)$  Argumente enthalten.
  - ▶ daher exponentiell (in n) viele Kategorien möglich
  - ▶ daher worst-case-Laufzeit exponentiell



(OpenCCG-Parser macht es in etwa so)

# Lösung (Grundidee)

$$\begin{array}{c} b \qquad b \\ \hline S/C/C/B \quad B/C/C/B \\ \hline S/C/C/C/C/B \end{array}$$

(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)

$$\begin{array}{c} b \qquad b \\ \hline S/C/C/B \quad B/C/C/B \\ \hline \hline S/C/C/C/C/B \quad B/C/C/B \end{array}$$

(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)

$$\begin{array}{c} b \qquad b \\ \hline S/C/C/B \quad B/C/C/B \\ \hline S/C/C/C/C/B \quad B/C/C/B \\ \hline \end{array}$$

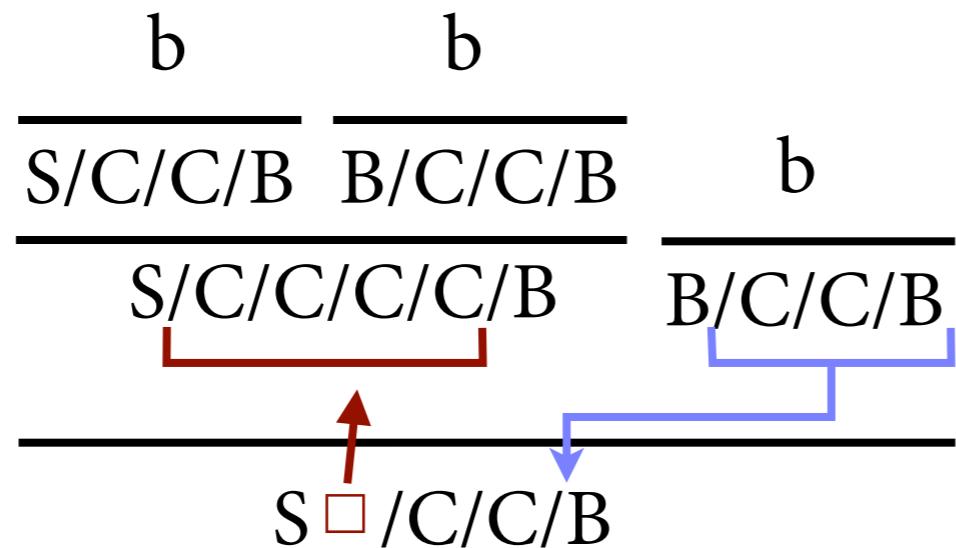
(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)

$$\begin{array}{c} b \qquad b \\ \hline S/C/C/B \quad B/C/C/B \\ \hline S/C/C/C/C/B \qquad b \\ \hline B/C/C/B \\ \hline S \square/C/C/B \end{array}$$

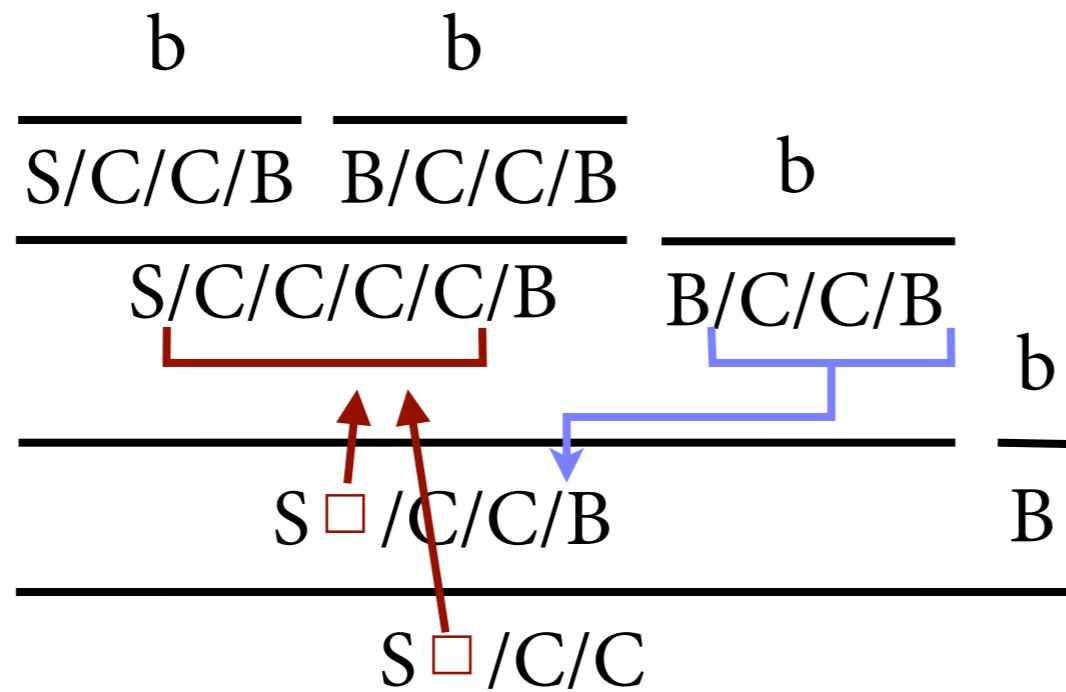
(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)



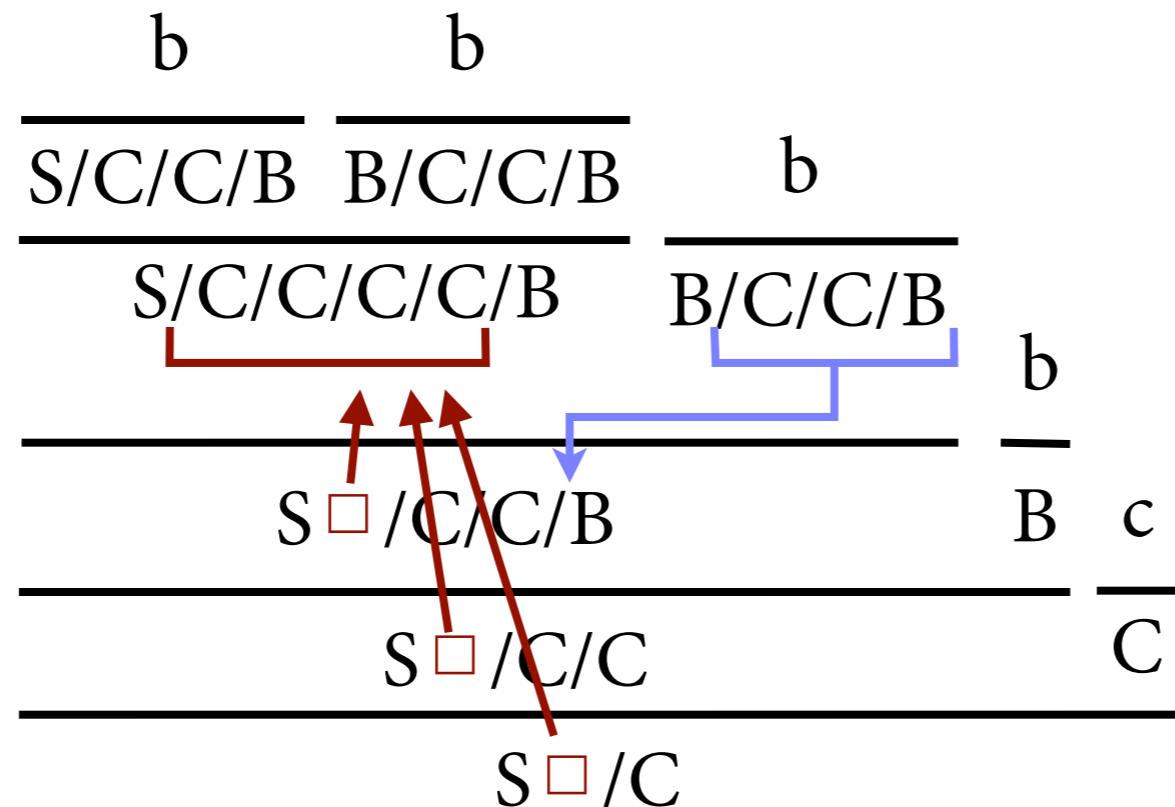
(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)



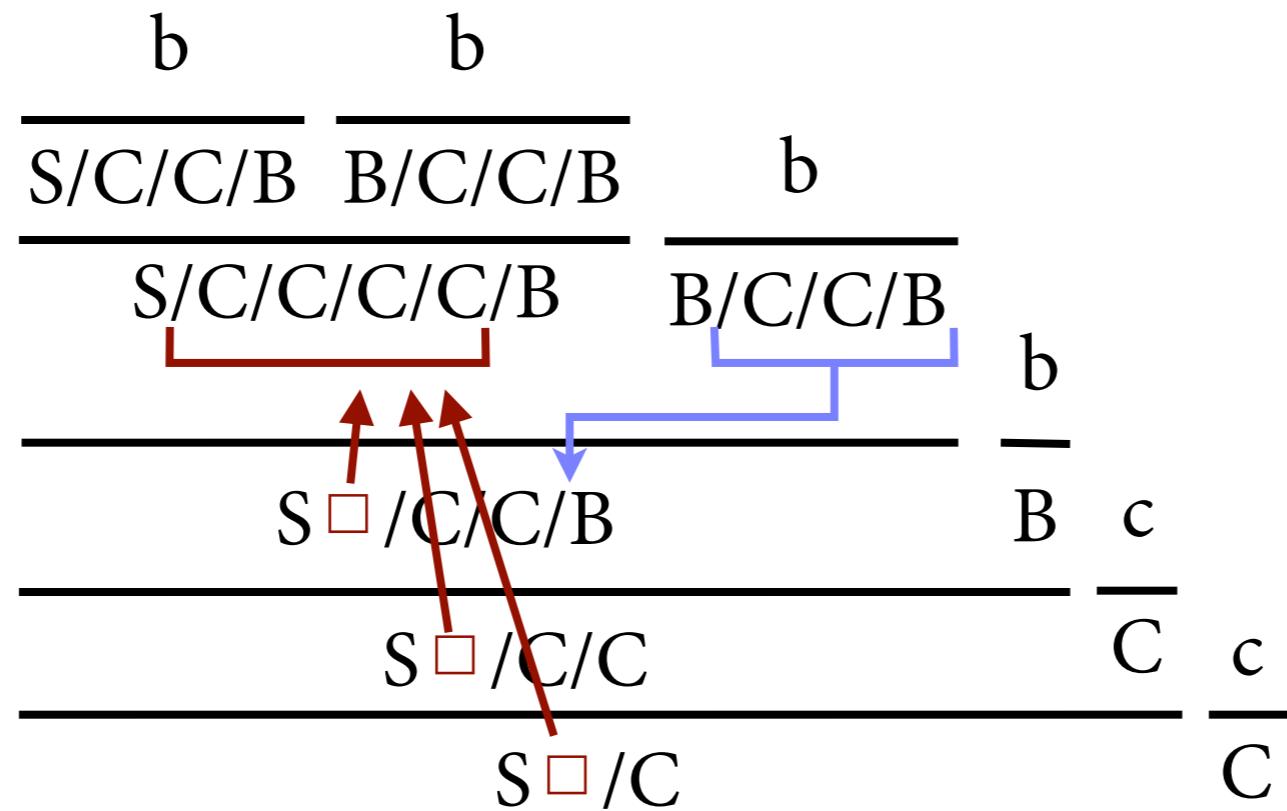
(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)



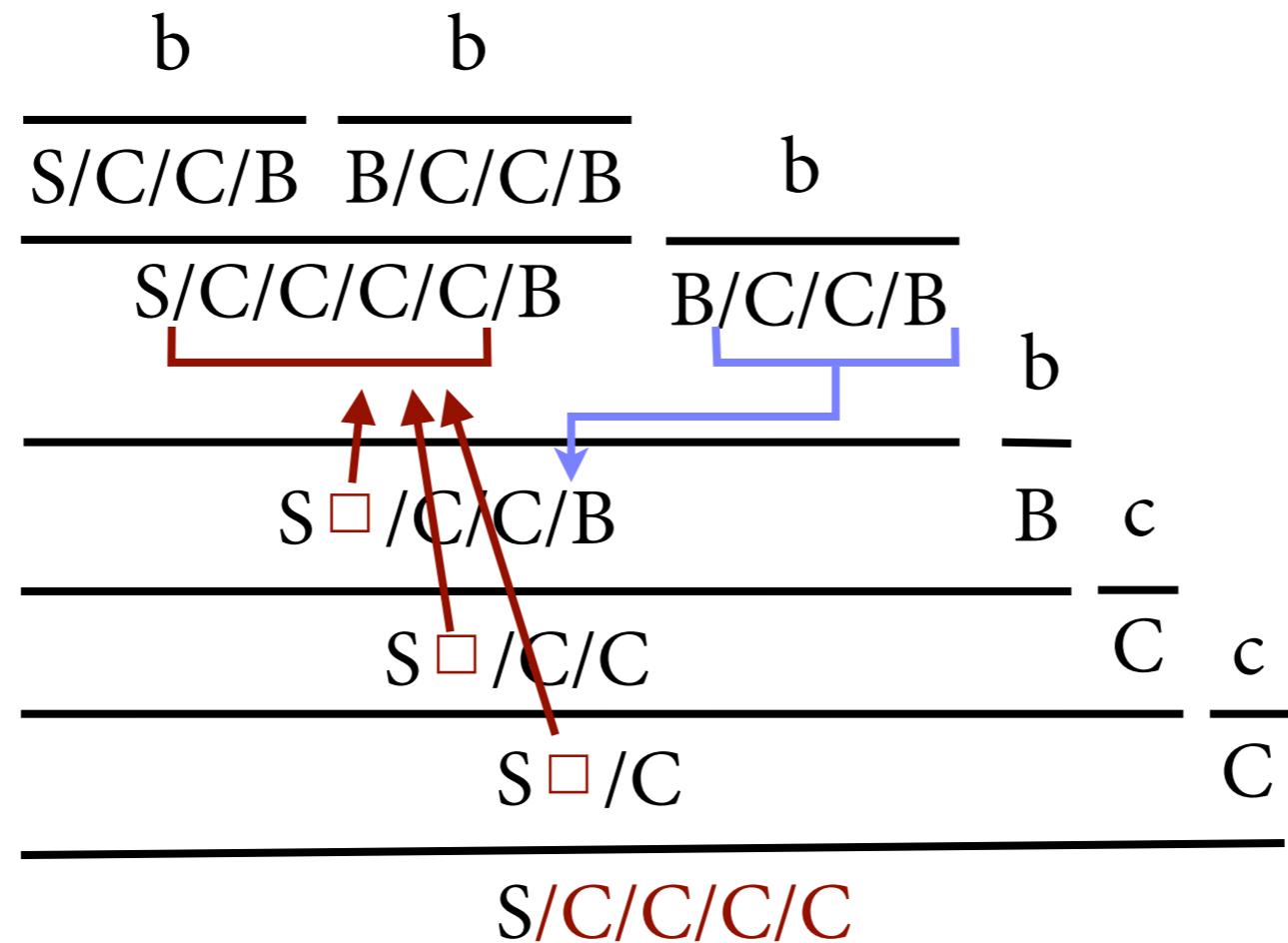
(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)



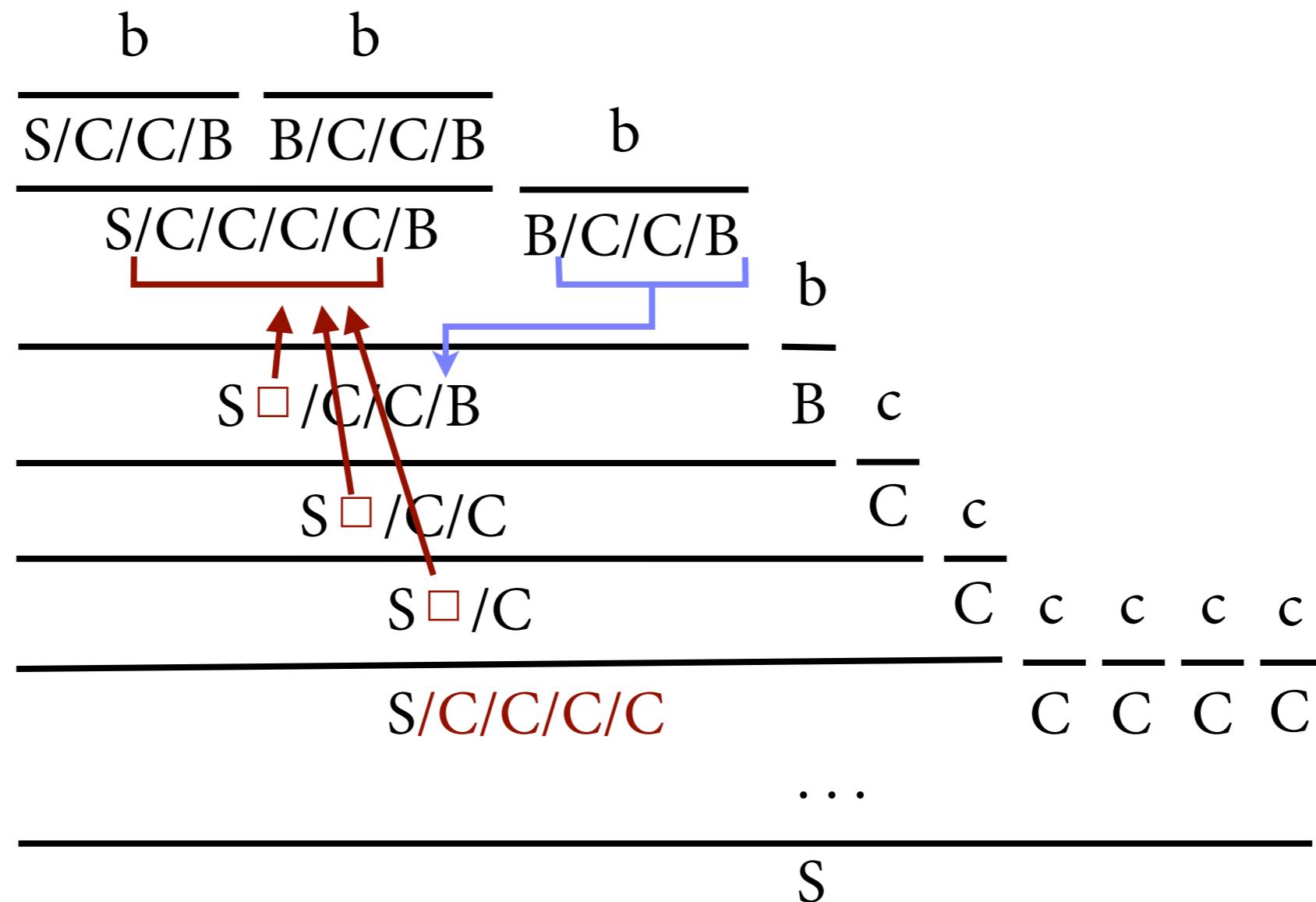
(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)



(Vijay-Shanker & Weir 1990)

# Lösung (Grundidee)



(Vijay-Shanker & Weir 1990)

# Parsingalgorithmus

- Idee: Länge von Kategorien in Items jetzt beschränkt.
  - ▶  $k_1$  = Grad der maximal erlaubten Kompositionsregel
  - ▶  $k_2$  = max. Anzahl der Argumente in lexikalischer Kat.
  - ▶ Längenbeschränkung:  $k_1 + k_2$  Argumente
- Teuerste Regel kombiniert drei Items:
  - ▶  $S \square/C + C + S/C/C/C/C/B \Rightarrow S/C/C/C/C$   

- Genaue Analyse dieser Regel ergibt Laufzeit  $O(n^6)$ .

# Statistisches Parsing

- Bahnbrechender statistischer Parser für CCG: C&C-Parser (Clark & Curran 2007).
- Idee:
  - ▶ Log-lineares W.modell auf Dependenzinterpretation von CCG-Parses.
  - ▶ Parsingalgorithmus berechnet besten Parse sehr effizient (mit Supertagging-Methoden).
  - ▶ Training: Syntaktisch annotiertes Korpus (z.B. PTB) in CCG-Annotationen übersetzen.

# Dependenz-Interpretation von CCG

- Extrahiere aus CCG-Parse Dependenz-Tupel:  
 $\langle \text{isst}_2, (\text{S}\backslash\text{NP}_1)/\text{NP}_2, 2, \text{käsebrot}_5 \rangle$ 
  - ▶ “Käsebrot” an Position 5 füllt zweites Argument von “isst” an Position 2.
- Vorgehen:
  - ▶ Argumentpositionen lexikalisch spezifizieren:  
isst:  $(\text{S}\backslash\text{NP}_1)/\text{NP}_2$
  - ▶ für jede Konstituente *Kopfwort* mitführen
  - ▶ Regelanwendung etabliert Dependenz zwischen Köpfen

# Beispiel

Lexikon:

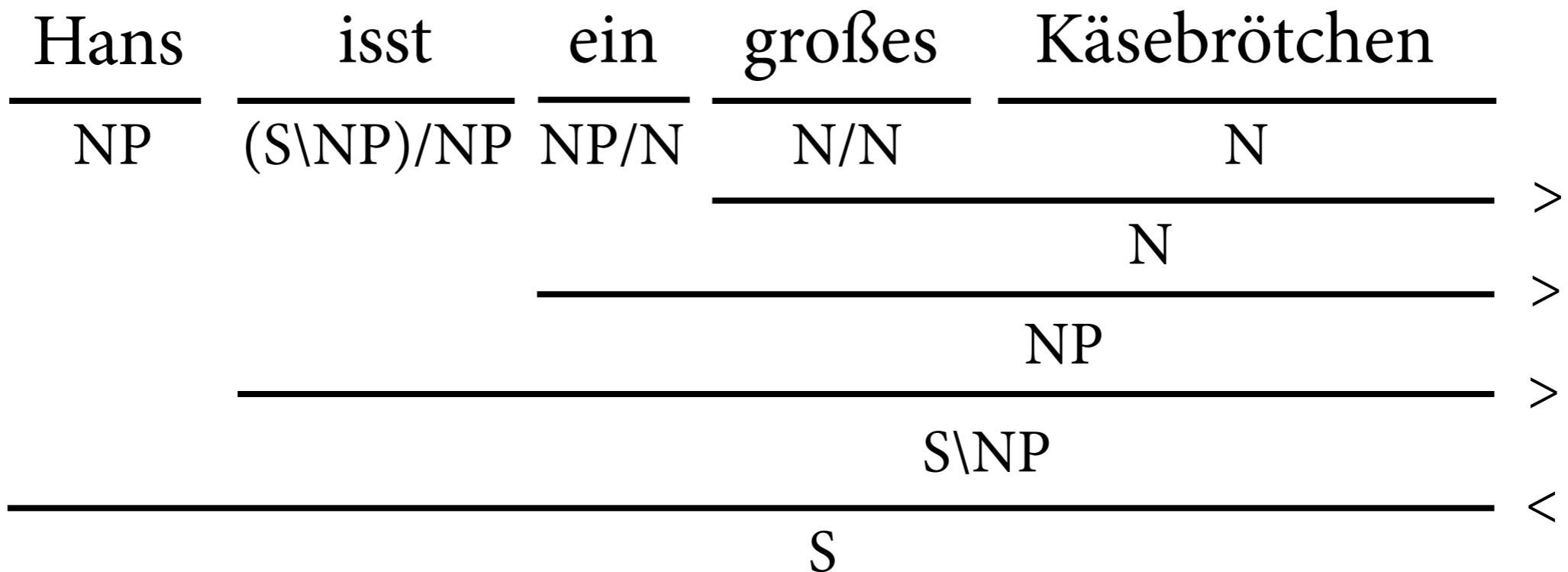
Hans: NP

ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>



# Beispiel

Lexikon:

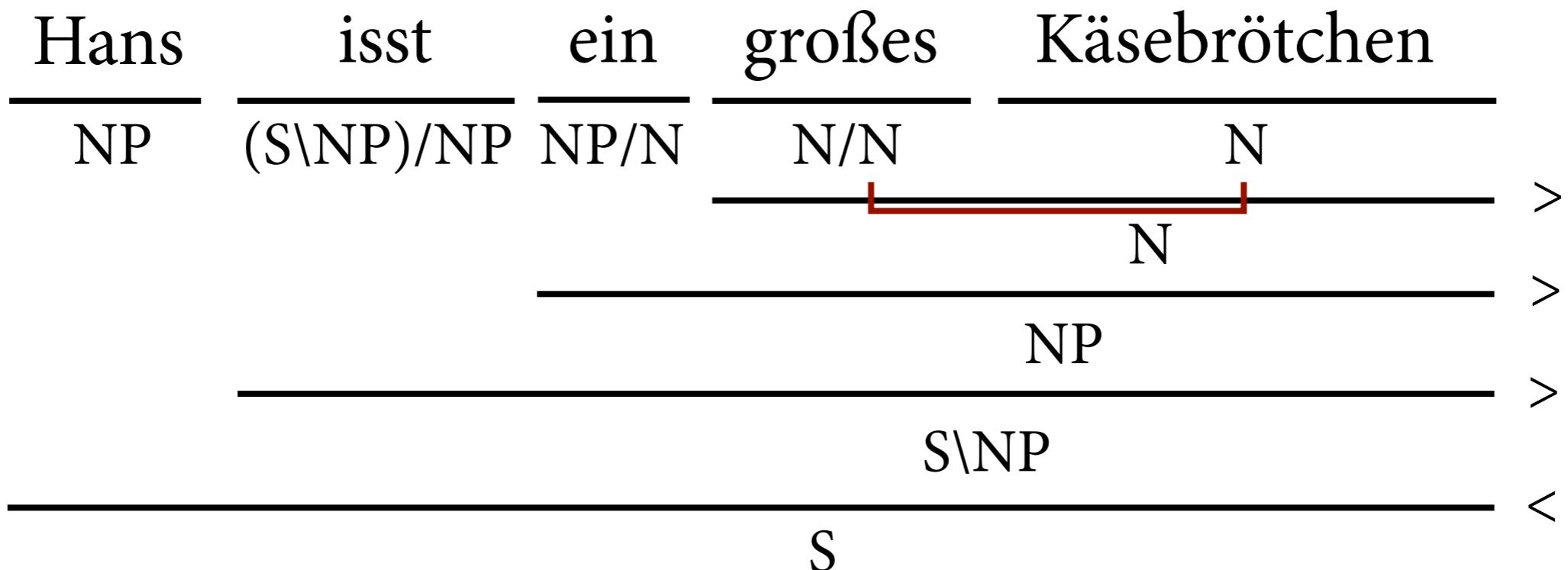
Hans: NP

ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>



# Beispiel

Lexikon:

Hans: NP

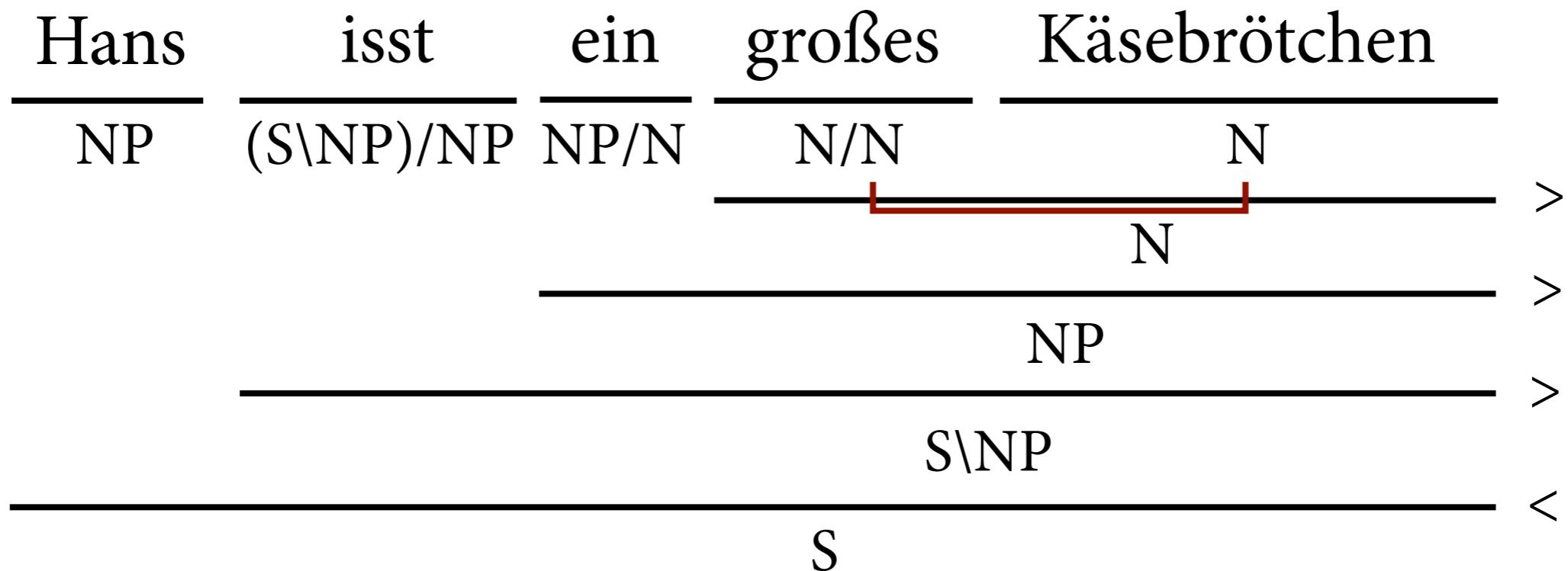
ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

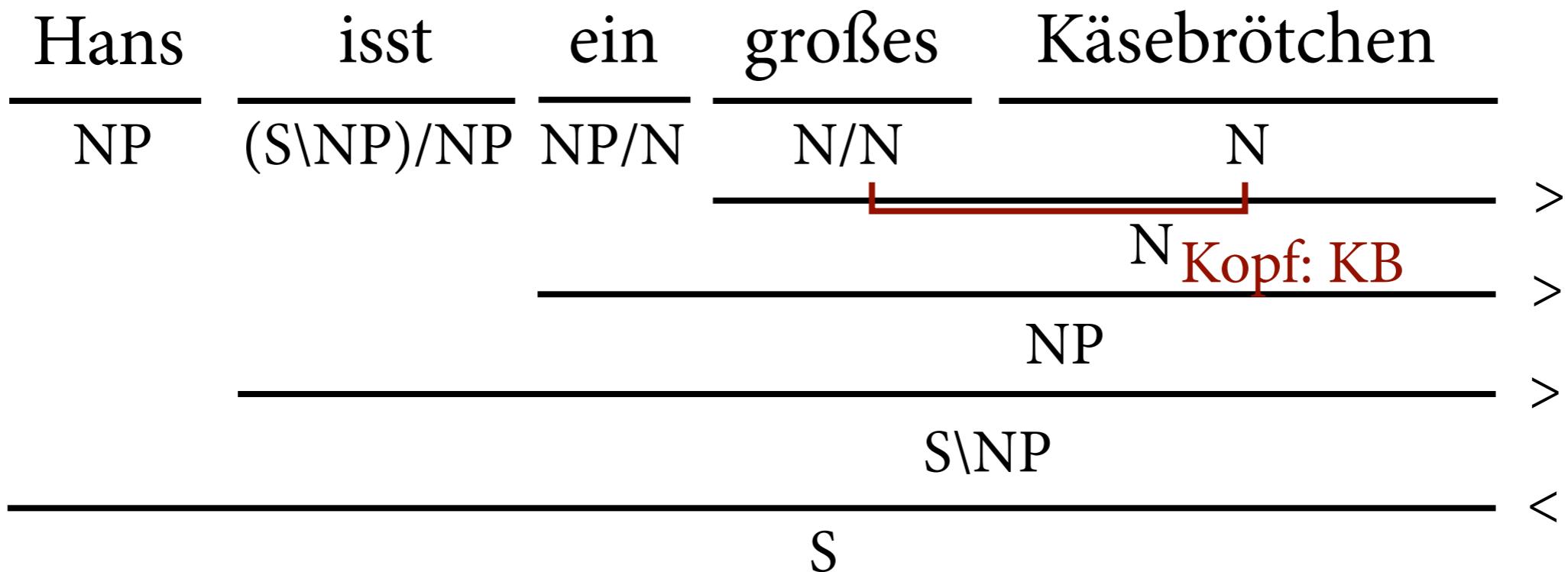
ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

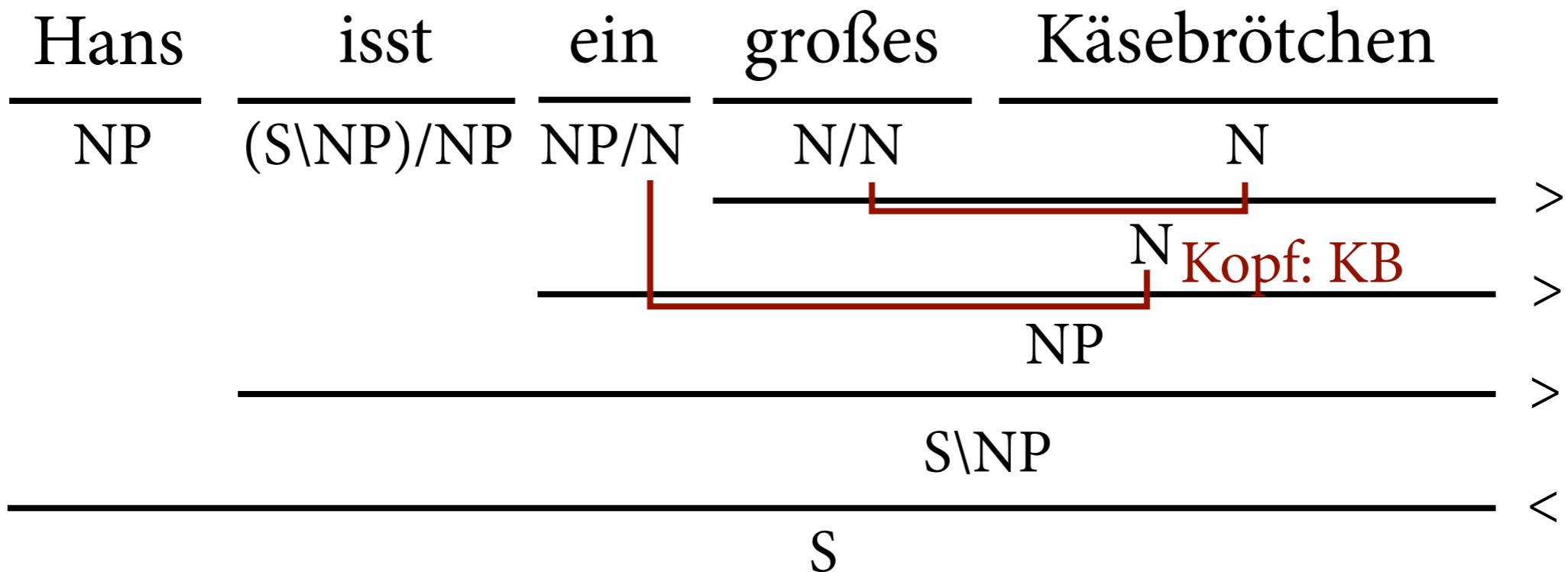
ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

ein: NP/N<sub>1</sub>

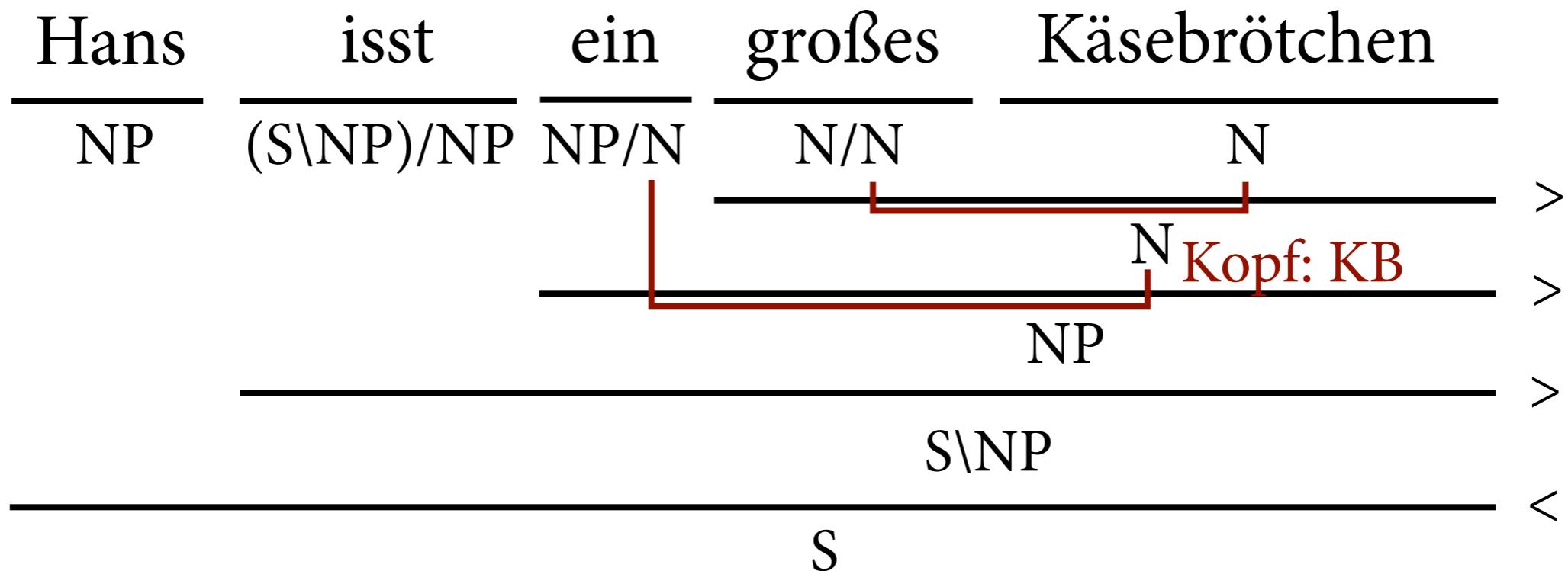
isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩

⟨ein<sub>3</sub>, NP/N<sub>1</sub>, 1, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

ein: NP/N<sub>1</sub>

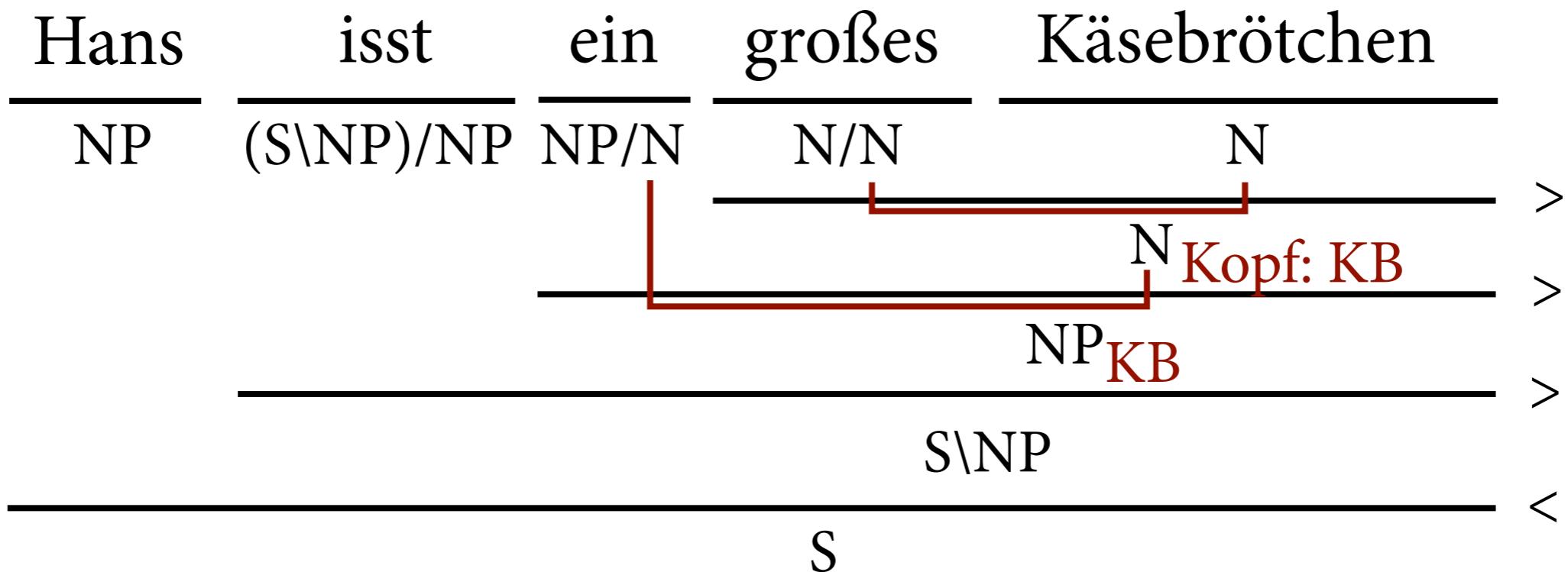
isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩

⟨ein<sub>3</sub>, NP/N<sub>1</sub>, 1, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

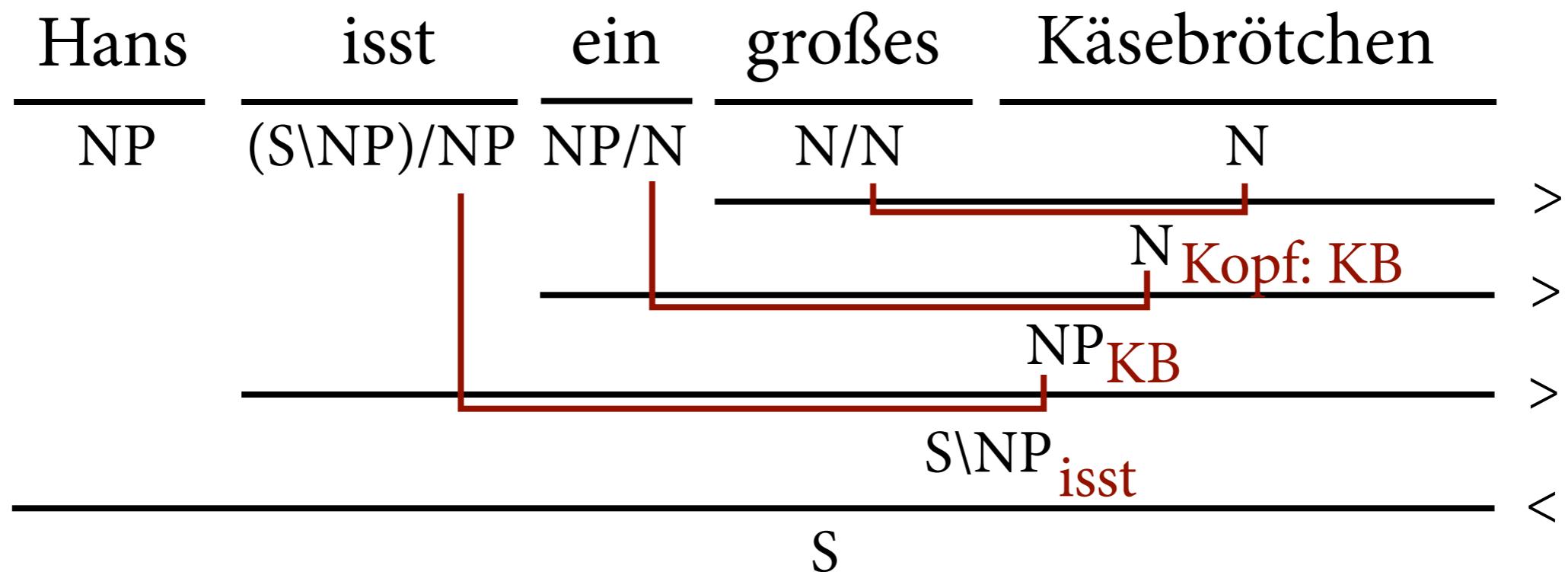
Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩

⟨ein<sub>3</sub>, NP/N<sub>1</sub>, 1, KB<sub>5</sub>⟩

⟨isst<sub>2</sub>, (S\NP<sub>1</sub>)/NP<sub>2</sub>, 2, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

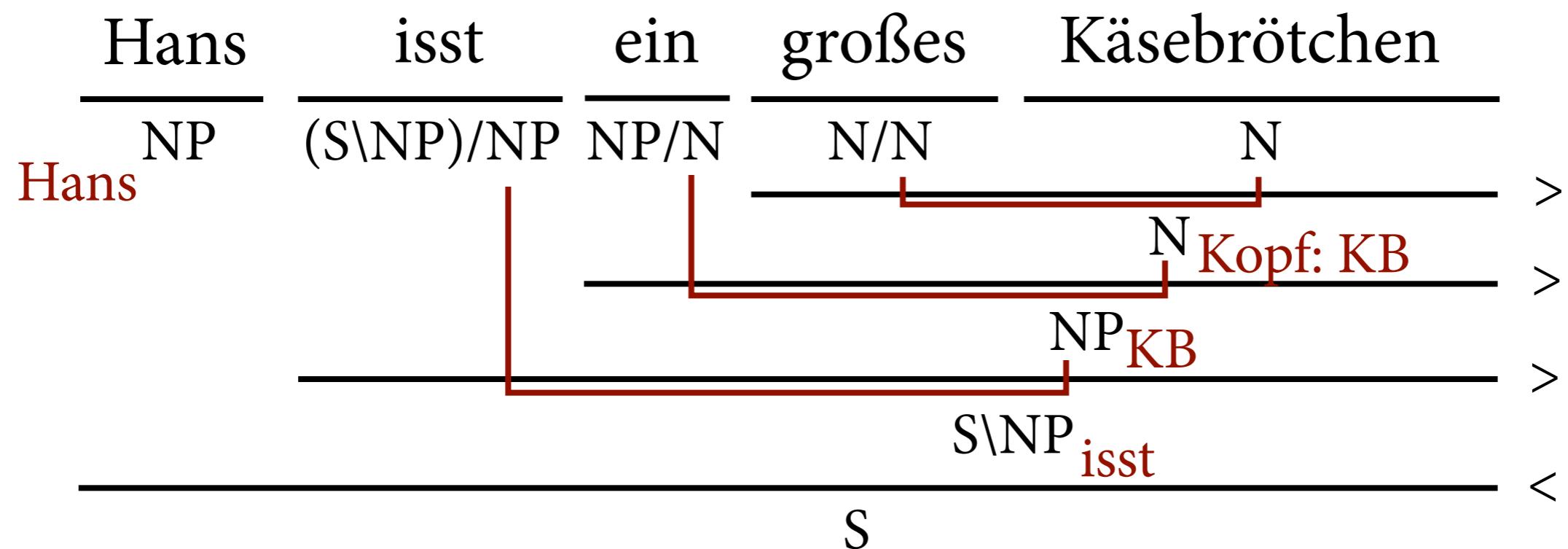
Käsebrötchen: N

großes: N/N<sub>1</sub>

⟨großes<sub>4</sub>, N/N<sub>1</sub>, 1, KB<sub>5</sub>⟩

⟨ein<sub>3</sub>, NP/N<sub>1</sub>, 1, KB<sub>5</sub>⟩

⟨isst<sub>2</sub>, (S\NP<sub>1</sub>)/NP<sub>2</sub>, 2, KB<sub>5</sub>⟩



# Beispiel

Lexikon:

Hans: NP

ein: NP/N<sub>1</sub>

isst: (S\NP<sub>1</sub>)/NP<sub>2</sub>

Käsebrötchen: N

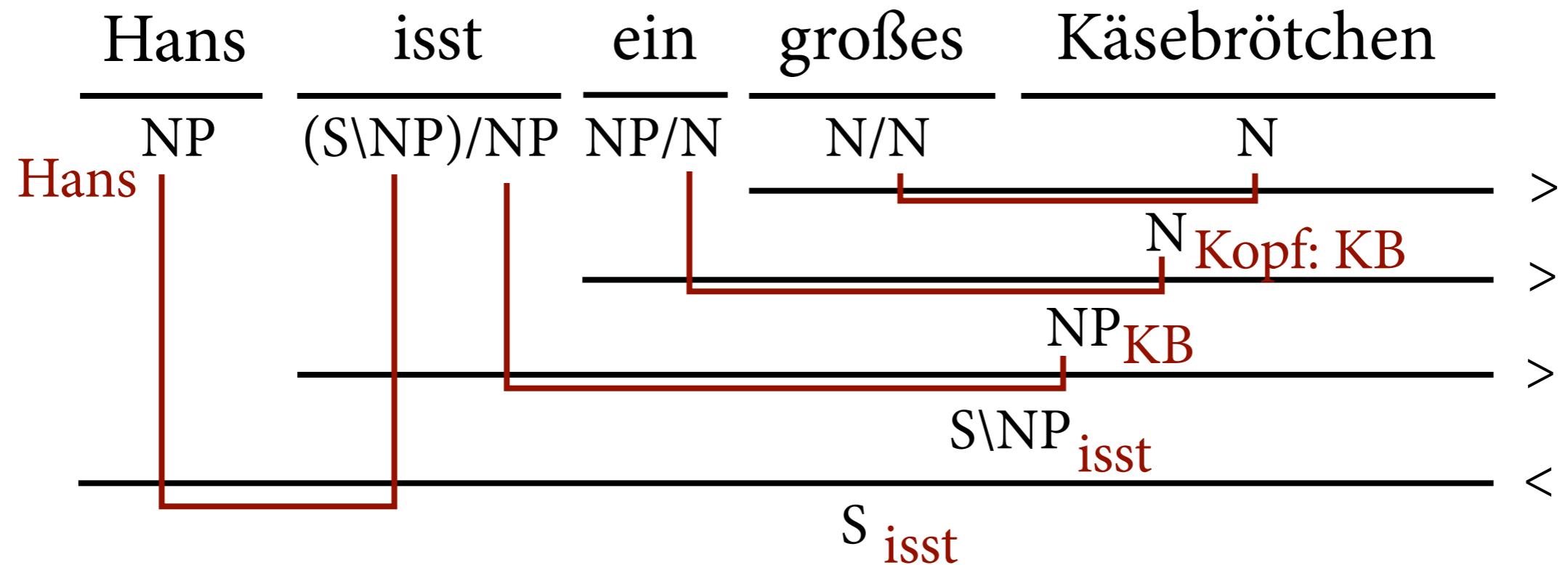
großes: N/N<sub>1</sub>

$\langle \text{großes}_4, \text{N}/\text{N}_1, 1, \text{KB}_5 \rangle$

$\langle \text{ein}_3, \text{NP}/\text{N}_1, 1, \text{KB}_5 \rangle$

$\langle \text{isst}_2, (\text{S}\backslash\text{NP}_1)/\text{NP}_2, 2, \text{KB}_5 \rangle$

$\langle \text{isst}_2, (\text{S}\backslash\text{NP}_1)/\text{NP}_2, 1, \text{Hans}_1 \rangle$



# Log-lineare W.modelle

- Wichtige & nützliche Klasse von W.modellen:  
*log-lineare* (oder *Maximum-Entropy*-) Modelle
- Definiert W.verteilung  $P(y|x)$  aufgrund von
  - ▶ *Featurefunktionen*  $f_i(x, y) \in \mathbb{R}$ : vom Nutzer festgelegt
  - ▶ *Gewichten*  $w_i \in \mathbb{R}$  für die Features: beim Training gelernt

$$P(y \mid x) = \frac{e^{w \cdot f(x, y)}}{\sum_{y'} e^{w \cdot f(x, y')}}$$

$$w \cdot f(x, y) = \sum_{i=1}^n w_i \cdot f_i(x, y)$$

# Training von log-lin Modellen

- Die Gewichte von log-linearen Modellen werden automatisch aus *annotierten Daten* gelernt.
  - ▶ Trainingsinstanzen sind Paare von (x,y)
- Berechne log-likelihood der Daten C aufgrund der aktuellen Schätzung von w, d.h.

$$LL(C) = \sum_{i=1}^N \log P(y_i \mid x_i; w)$$

- Suche nach w für maximale log-likelihood durch Gradient-Ascent-Algorithmus.

# C&C-Parser: log-lin. Modell

- Wir wollen diskriminatives W.modell  
 $P(t \mid x)$  für Parsebaum t gegeben String x.
- Verwende dafür log-lineares W.modell mit Features, die spezifisch für CCG-Parsing gemacht sind.
- Training auf annotiertem Korpus, das in CCG-Ableitungen (plus Dependenzen) konvertiert wurde.
  - ▶ vgl. Konvertierung von Penn Treebank nach TAG
  - ▶ Fokus auf korrekte lexikalische Kategorien

# Einige Features

Feature type	Example
LexCat + Word	$(S/S)/NP + \text{Before}$
LexCat + POS	$(S/S)/NP + \text{IN}$
RootCat	$S[dcl]$
RootCat + Word	$S[dcl] + \text{was}$
RootCat + POS	$S[dcl] + \text{VBD}$
Rule	$S[dcl] \rightarrow NP S[dcl] \setminus NP$
Rule + Word	$S[dcl] \rightarrow NP S[dcl] \setminus NP + \text{bought}$
Rule + POS	$S[dcl] \rightarrow NP S[dcl] \setminus NP + \text{VBD}$

Feature type	Example
Word–Word	$\langle \text{bought}, (S \setminus NP_1)/NP_2, 2, \text{stake}, (NP \setminus NP)/(S[dcl]/NP) \rangle$
Word–POS	$\langle \text{bought}, (S \setminus NP_1)/NP_2, 2, \text{NN}, (NP \setminus NP)/(S[dcl]/NP) \rangle$
POS–Word	$\langle \text{VBD}, (S \setminus NP_1)/NP_2, 2, \text{stake}, (NP \setminus NP)/(S[dcl]/NP) \rangle$
POS–POS	$\langle \text{VBD}, (S \setminus NP_1)/NP_2, 2, \text{NN}, (NP \setminus NP)/(S[dcl]/NP) \rangle$
Word + Distance(words)	$\langle \text{bought}, (S \setminus NP_1)/NP_2, 2, (NP \setminus NP)/(S[dcl]/NP) \rangle + 2$
Word + Distance(punct)	$\langle \text{bought}, (S \setminus NP_1)/NP_2, 2, (NP \setminus NP)/(S[dcl]/NP) \rangle + 0$
Word + Distance(verbs)	$\langle \text{bought}, (S \setminus NP_1)/NP_2, 2, (NP \setminus NP)/(S[dcl]/NP) \rangle + 0$
POS + Distance(words)	$\langle \text{VBD}, (S \setminus NP_1)/NP_2, 2, (NP \setminus NP)/(S[dcl]/NP) \rangle + 2$
POS + Distance(punct)	$\langle \text{VBD}, (S \setminus NP_1)/NP_2, 2, (NP \setminus NP)/(S[dcl]/NP) \rangle + 0$
POS + Distance(verbs)	$\langle \text{VBD}, (S \setminus NP_1)/NP_2, 2, (NP \setminus NP)/(S[dcl]/NP) \rangle + 0$

# Effizientes Parsing

- Features sind so definiert, dass sie schon auf Parse-Items ausgewertet werden können, nicht erst auf fertigem Parsebaum.
- Damit effizientes Chartparsing möglich.
- C&C-Parser ist worst-case exponentiell, aber reduziert Suchraum des Chartparsers, indem vorher Supertagging durchgeführt wird.

# A\*-Parsing für CCG

- Parsing-Schema für CCG-Regeln:

$$\frac{[X/Y, i, j] \quad [Y, j, k]}{[X, i, k]} \qquad \frac{[Y\backslash Z, i, j] \quad [X\backslash Y, j, k]}{[X\backslash Z, i, k]}$$

- Erzeuge damit neue Parse-Items aus alten.
  - Neue Parse-Items kommen auf *Agenda*, die nach *Score* des Items sortiert ist.
  - In jedem Schritt: hole erstes (= bestes) Item von der Agenda.
- Beende Parser, sobald erstes komplettes Item (für den ganzen String) gefunden wurde.

# Scores für Items

- Scores für Items beeinflussen die Effizienz und Akkuratheit des Parsers.
  - ▶ erstes gefundenes Ziel-Item = bester Parse?
  - ▶ wie viele unnötige Items erzeugt, bevor erstes Ziel-Item von der Agenda genommen wird?
- Idee aus A<sup>\*</sup>-Suche:
  - ▶ Score = W. für Substring + Schätzung der zukünftigen
  - ▶ garantiert, dass erstes Ziel-Item das beste ist
  - ▶ gute Schätzung = wenig unnötige Items erzeugt

# Supertag-Factored W.modell

- Definiere W. einer CCG-Ableitung t für Satz w über die Supertags (= lex. Kategorien) der einzelnen Wörter:

$$P(t \mid w) = \prod_{i=1}^n P(s_i \mid w)$$

- Score für A<sup>\*</sup>-Suche:
  - ▶ Bisherige W. für String mit Supertags s<sub>i</sub>, ..., s<sub>k</sub>:  
 $P(s_i \mid w) * \dots * P(s_k \mid w)$
  - ▶ Schätzung für restliche Kosten:  
 $(\max P(s_1 \mid w)) * \dots * (\max P(s_{i-1} \mid w))$   
 $* (\max P(s_{k+1} \mid w)) * \dots * (\max P(s_n \mid w))$

# Parsing-Algorithmus

## 1. Supertagging:

Berechne für jedes Wort die W. für alle lex. Kategorien.

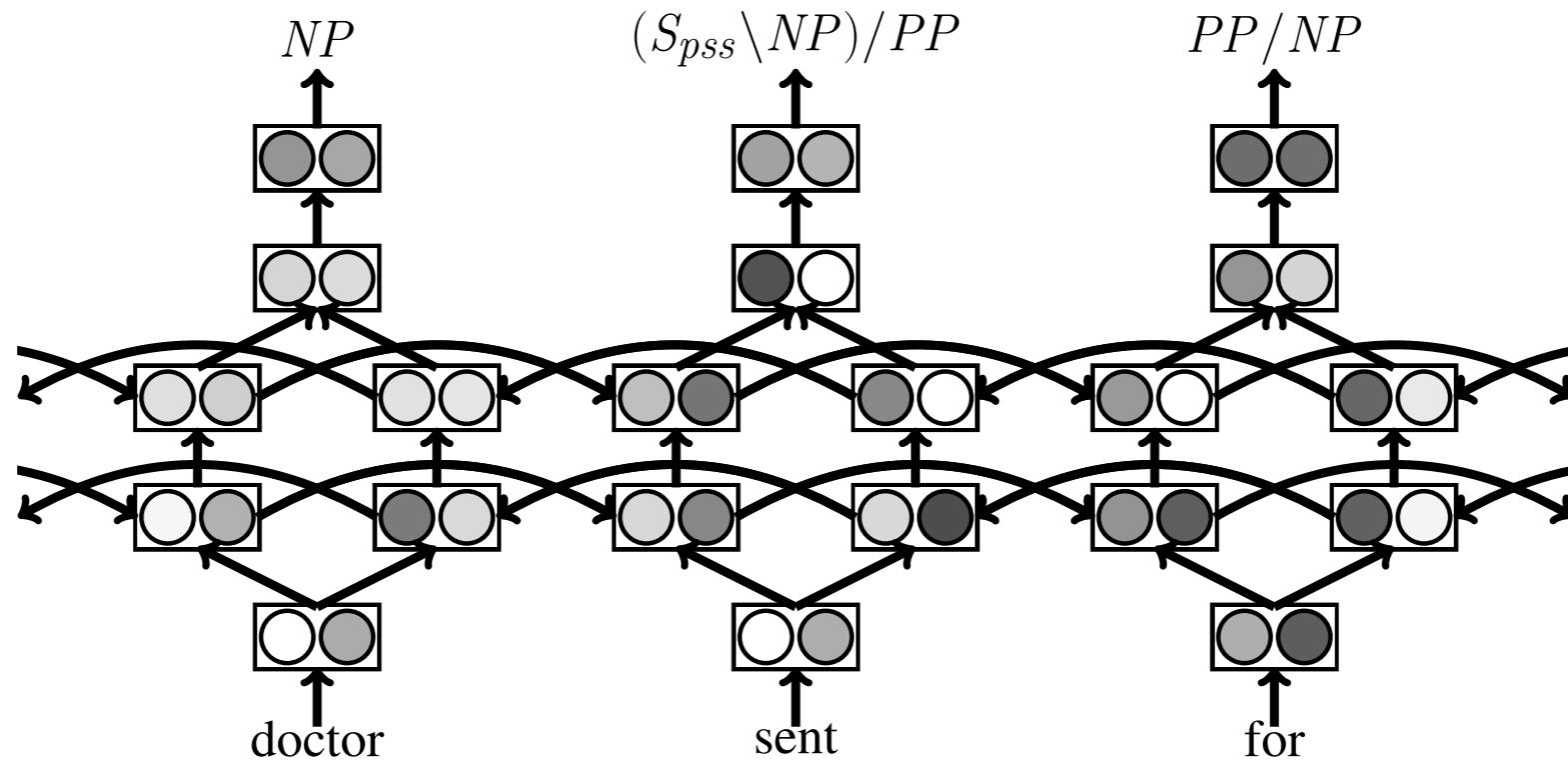
## 2. A<sup>\*</sup>-Parsing:

So lange Items von der Agenda nehmen und neue Items in Chart + Agenda einsortieren, bis Ziel-Item gefunden wurde.

## 3. CCG-Ableitung für erstes gefundenes Ziel-Item zurückgeben.

# Supertagging für CCGs

- Mit neuronalen Netzwerken (bidirectional LSTM):



# Evaluation: Supertagging

Model	Dev	Test
C&C tagger	91.5	92.0
NN	91.3	91.6
RNN	93.1	93.0
LSTM	94.1	94.3
LSTM + Tri-training	<b>94.9</b>	<b>94.7</b>

Table 1: Supertagging accuracy on CCGbank.

“Tri-Training”: Verwende zwei existierende Parser A und B, um großes Korpus zu parsen. Nimm alle Sätze, auf denen A und B gleiche CCG-Ableitung zurückgeben, zu Trainingsdaten hinzu.  
→ 43 Mio. Tokens zusätzliche Trainingsdaten.

# Evaluation: Parsing-Accuracy

Model	P	R	F1
C&C	86.2	84.2	85.2
C&C + RNN	87.7	86.4	87.0
EASYCCG	83.7	83.0	83.3
Dependencies	86.5	85.8	86.1
LSTM	87.7	86.7	87.2
LSTM + Dependencies	88.2	87.3	87.8
LSTM + Tri-training	<b>88.6</b>	<b>87.5</b>	<b>88.1</b>
LSTM + Tri-training + Dependencies	88.2	87.3	87.8

Table 2: Labelled F1 for CCGbank dependencies on the CCGbank test set (Section 23).

# Evaluation: Geschwindigkeit

Parser	Sentences per second
SpaCy* <sup>4</sup>	778
Berkeley GPU* (Hall et al., 2014)	687
Chen and Manning (2014)*	391
C&C	66
EASYCCG	606
LSTM	214
LSTM + Dependencies	58
LSTM GPU	2670

“LSTM GPU”: Supertagger läuft auf GPU und bekommt gleichzeitig Tausende von Sätzen als Input.

# Zusammenfassung

- Naive Parsingalgorithmen für CCG: exponentielle Laufzeit.
- Geschicktes Management von Kategorien: kann Laufzeit auf  $O(n^6)$  bringen.
- Wichtige statistische CCG-Parser:
  - ▶ C&C Parser
  - ▶ EasyCCG-Parser
  - ▶ Worst-case exponentiell, aber effektive Nutzung von Supertagging erlaubt extrem hohe Effizienz.