CCG III:
W.modelle und Parsing

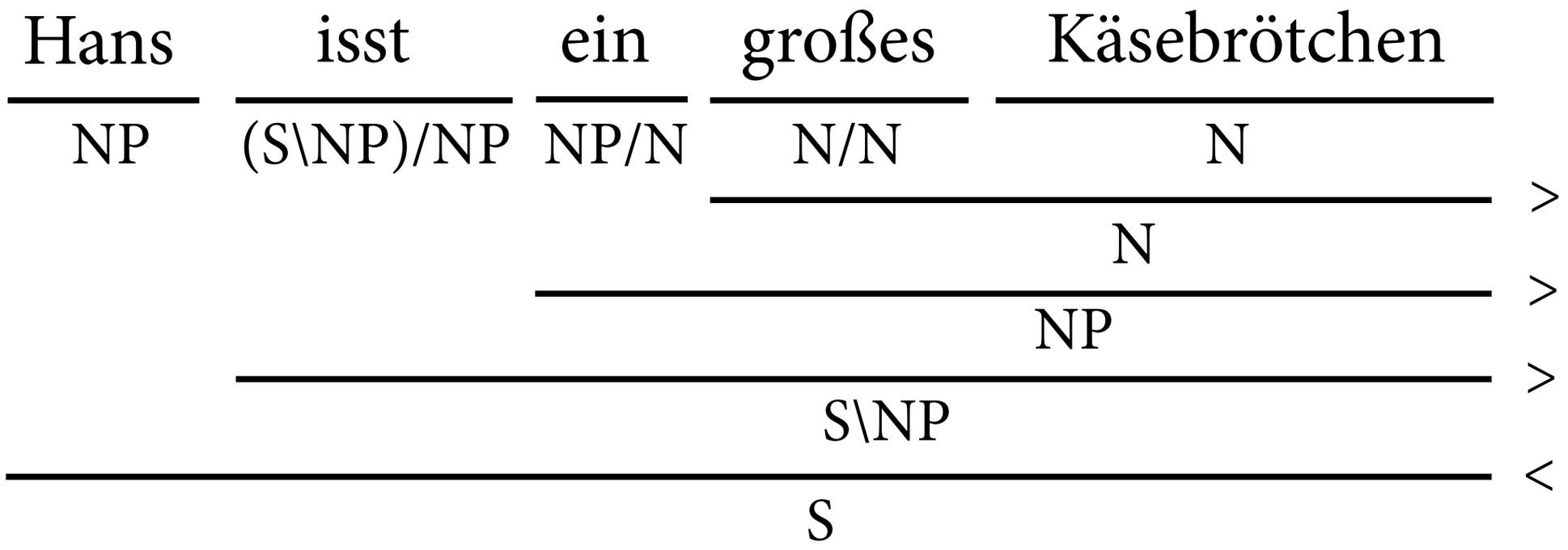
Vorlesung “Grammatikformalismen”
Alexander Koller

26. Mai 2017

CCG: Beispiel

Lexikon:

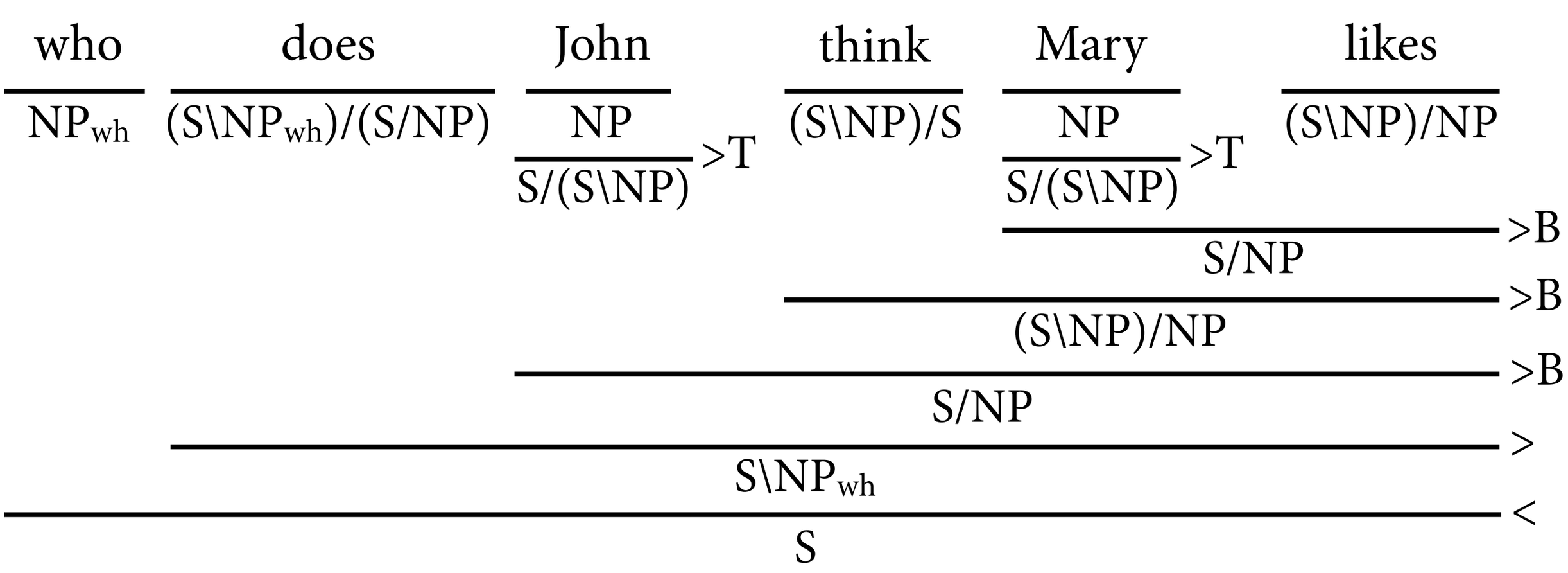
Hans: NP	ein: NP/N
isst: (S\NP)/NP	Käsebrötchen: N
großes: N/N	



CCG: Beispiel

Lexikon:

John: NP	think: (S\NP)/S
Mary: NP	likes: (S\NP)/NP
who: NP_{wh}	does: $(S\backslash NP_{wh})/(S/NP)$



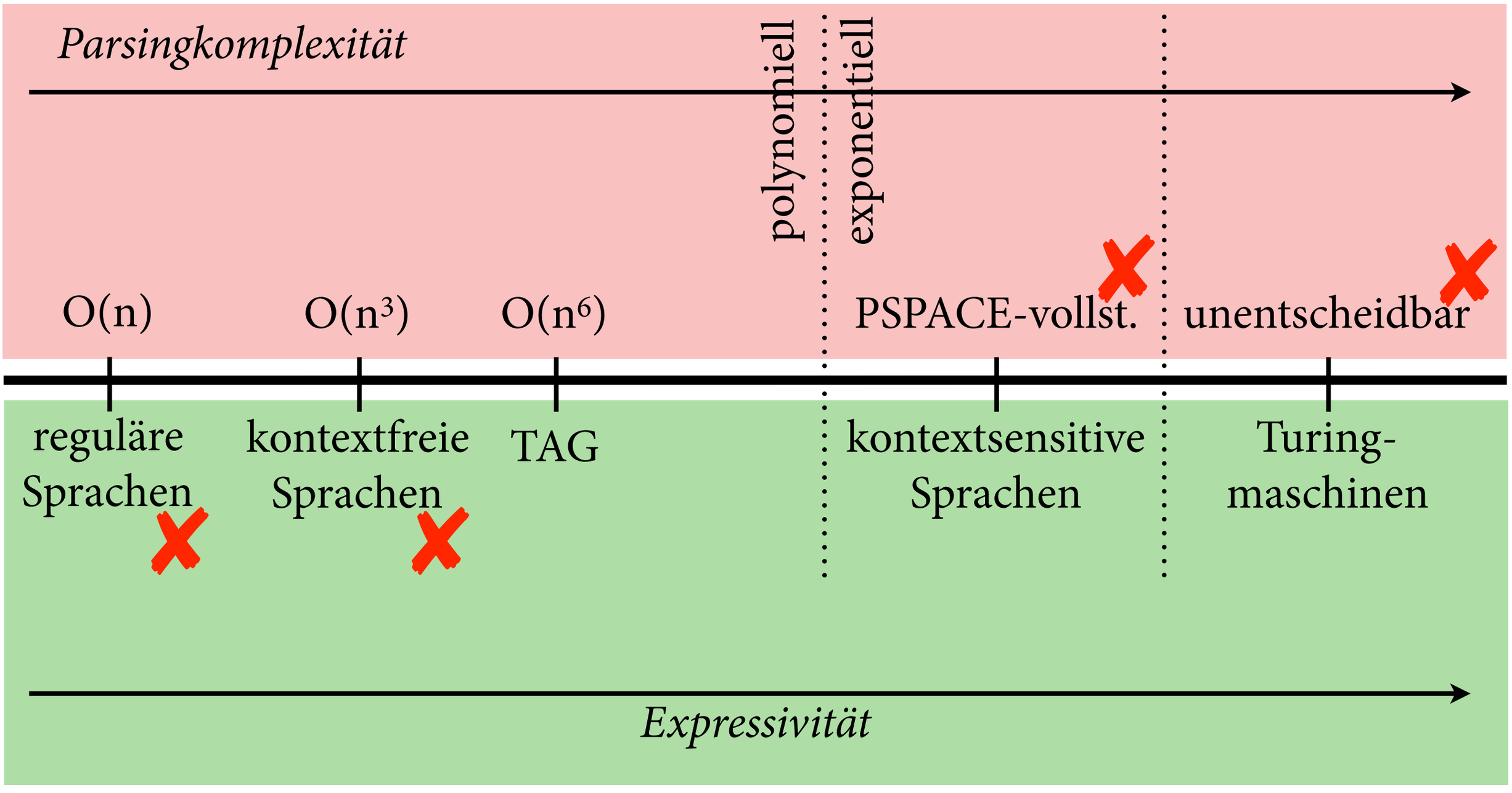
Schweizerdeutsch

- Crossed composition erlaubt Modellierung von cross-serial dependencies:

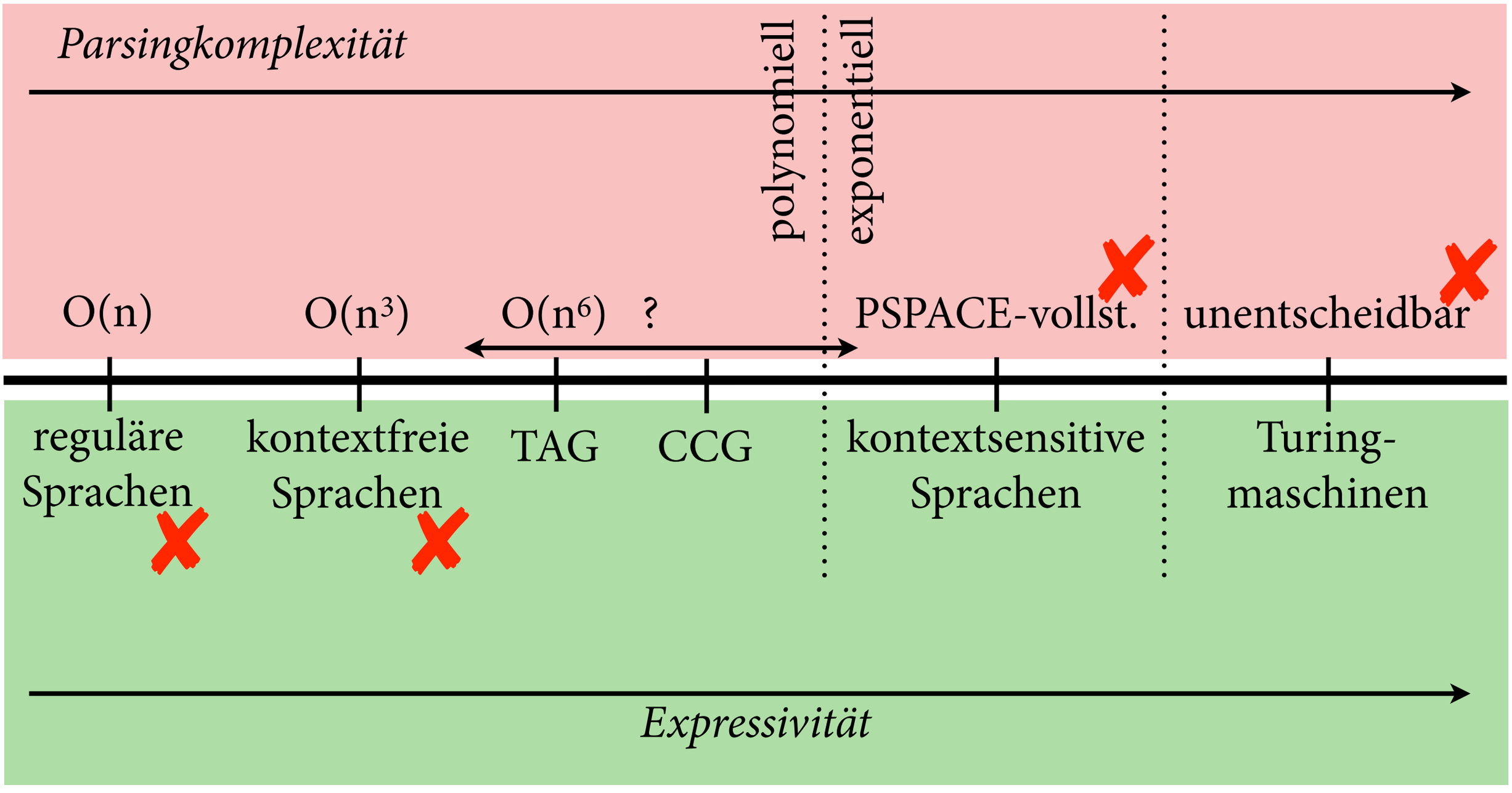
das	mer	em Hans	es huus	hälfed	aastriiche
	NP_{nom}	NP_{dat}	NP_{acc}	$S_{+SUB}\backslash NP_{nom}\backslash NP_{dat}/VP$	$VP\backslash NP_{acc}$
				$S_{+SUB}\backslash NP_{nom}\backslash NP_{dat}\backslash NP_{acc}$	>Bx
			$S_{+SUB}\backslash NP_{nom}\backslash NP_{dat}$		<
		$S_{+SUB}\backslash NP_{nom}$			<
	S_{+SUB}				<

(“VP” = Abkürzung für $S\backslash NP_{nom}$)

Grammatikformalismen



Grammatikformalismen

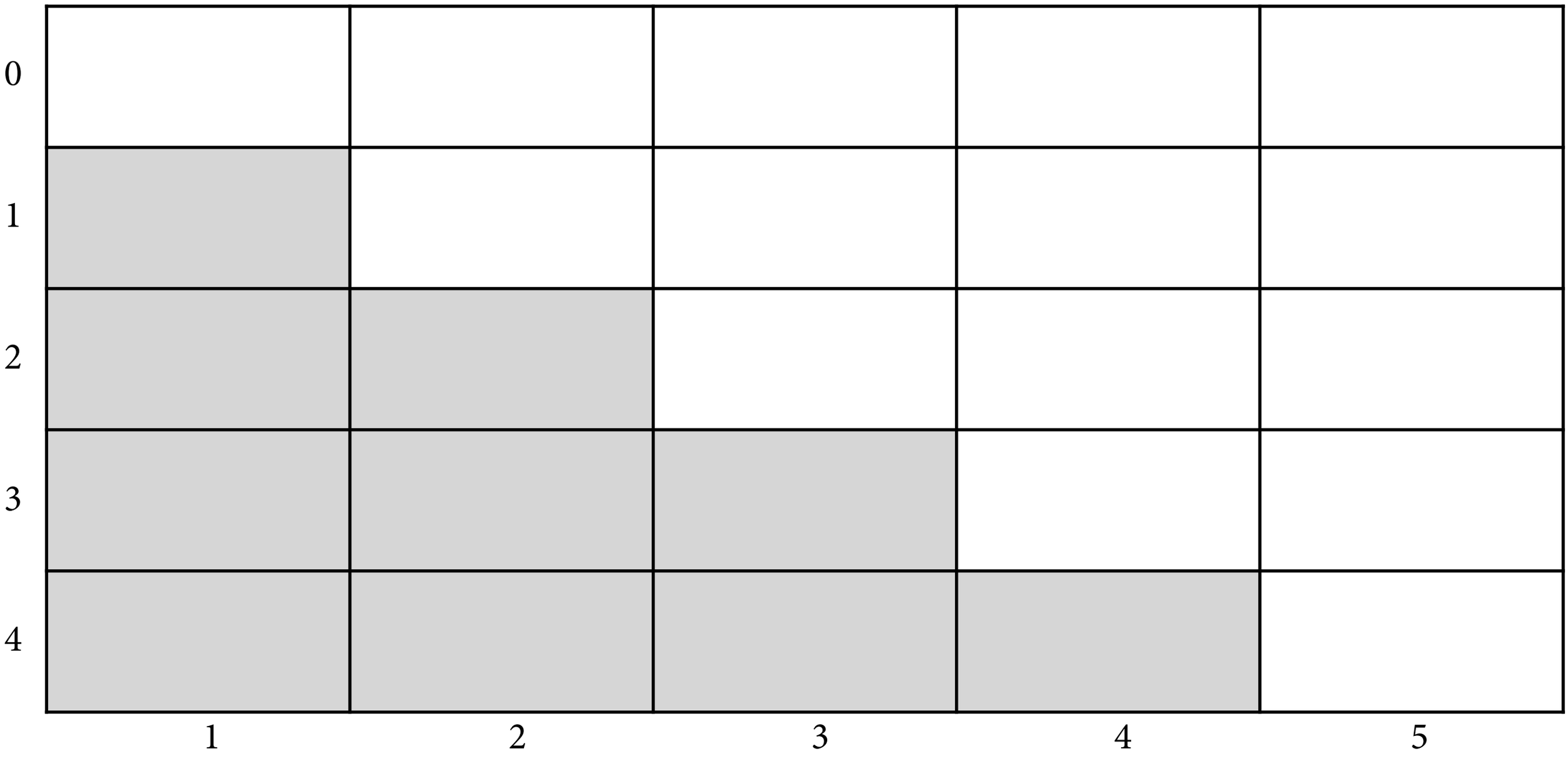


Heute

- 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



CKY-Parsing für CCG

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

	1	2	3	4	5
0	NP				
1					
2					
3					
4					

CKY-Parsing für CCG

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

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

CKY-Parsing für CCG

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

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

CKY-Parsing für CCG

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

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

CKY-Parsing für CCG

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

	1	2	3	4	5
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					
	1	2	3	4	5

CKY-Parsing für CCG

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

	1	2	3	4	5
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

	1	2	3	4	5
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

	1	2	3	4	5
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

	1	2	3	4	5
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

	1	2	3	4	5
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

CKY-Parsing für CCG

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

	1	2	3	4	5
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

	1	2	3	4	5
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

CKY-Parsing für CCG

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

	1	2	3	4	5
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

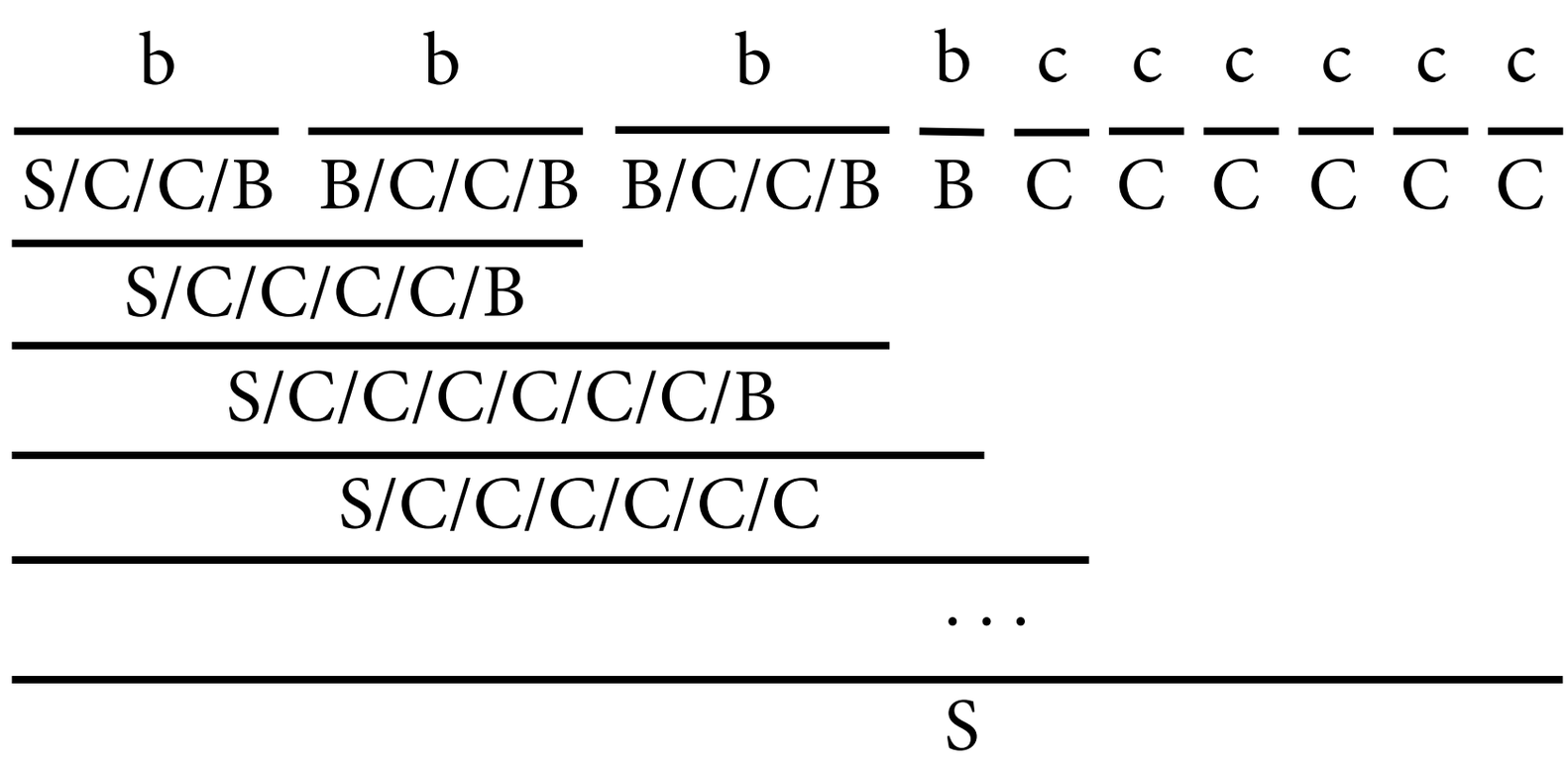
Hans	isst	ein	großes	Käsebrötchen
NP	(S\NP)/NP	NP/N	N/N	N

	1	2	3	4	5
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

Wie viele Kategorien pro Chart-Zelle?

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)

$$\frac{\dfrac{b}{S/C/C/B} \quad \dfrac{b}{B/C/C/B}}{S/C/C/C/C/B}$$

(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)

$$\dfrac{\dfrac{\dfrac{b}{S/C/C/B}\quad\dfrac{b}{B/C/C/B}}{S/C/C/C/C/B}\quad\dfrac{b}{B/C/C/B}}{}$$

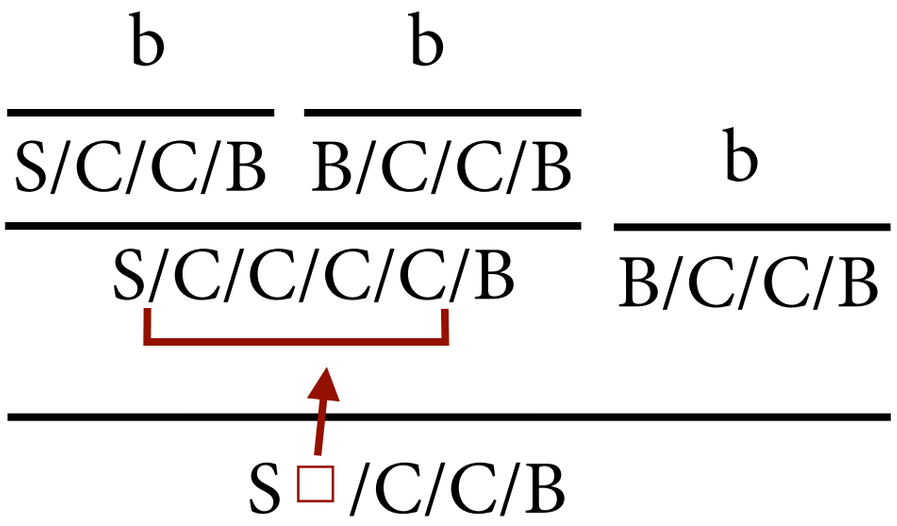
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)

$$\dfrac{\dfrac{\dfrac{b}{S/C/C/B}\quad\dfrac{b}{B/C/C/B}}{S/C/C/C/C/B}\quad\dfrac{b}{B/C/C/B}}{\qquad\qquad}$$

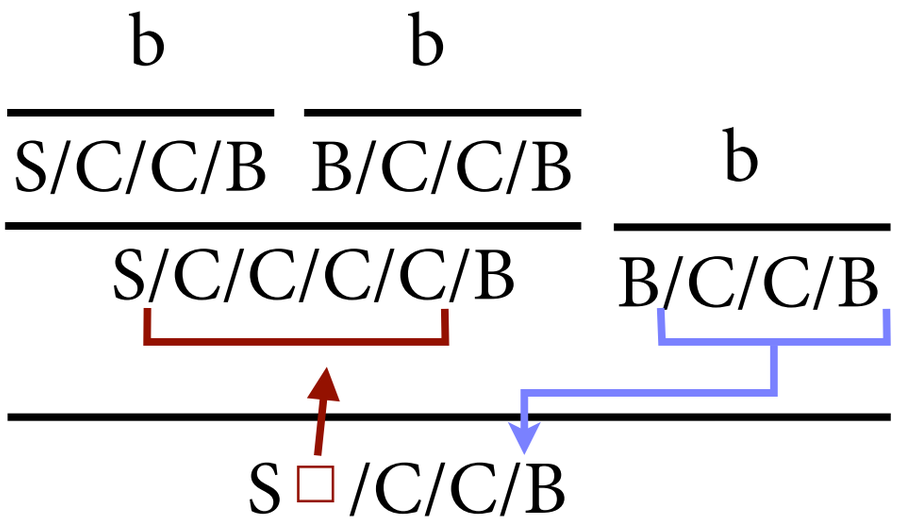
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)



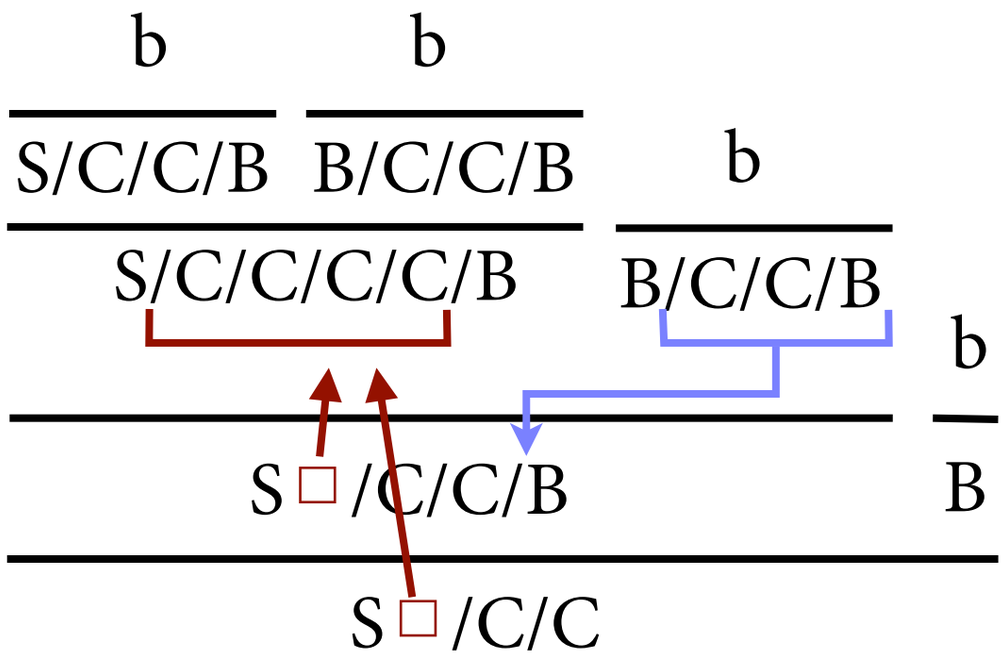
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)



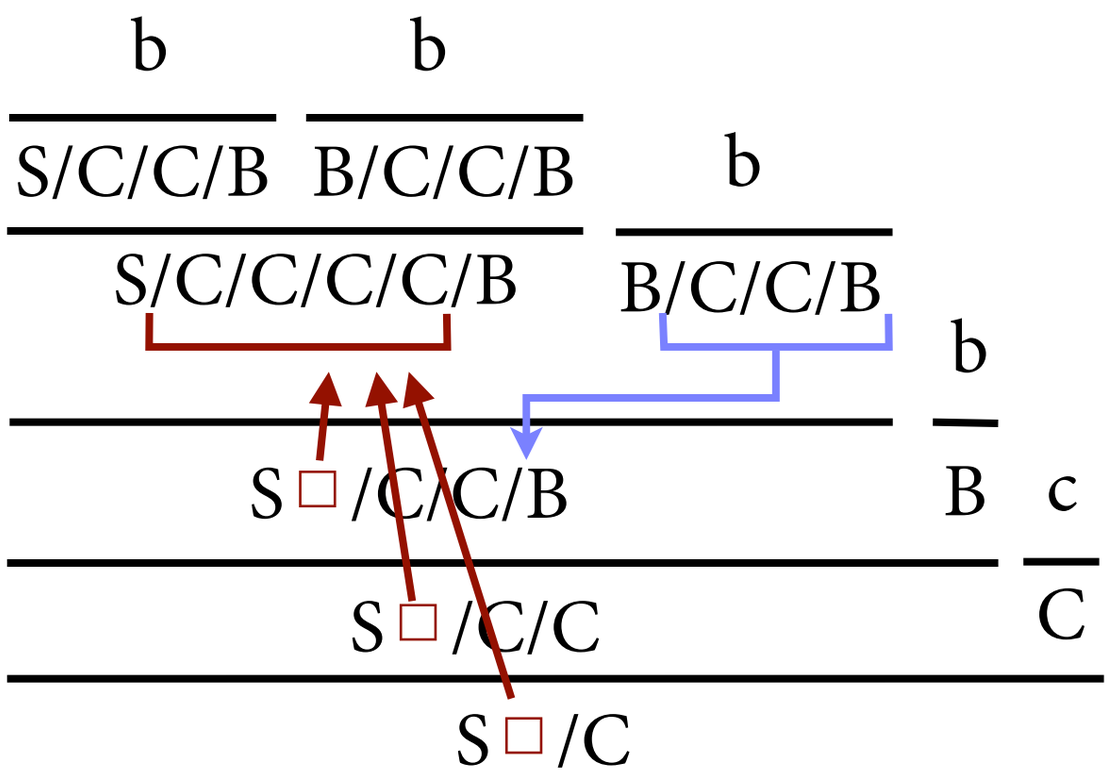
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)



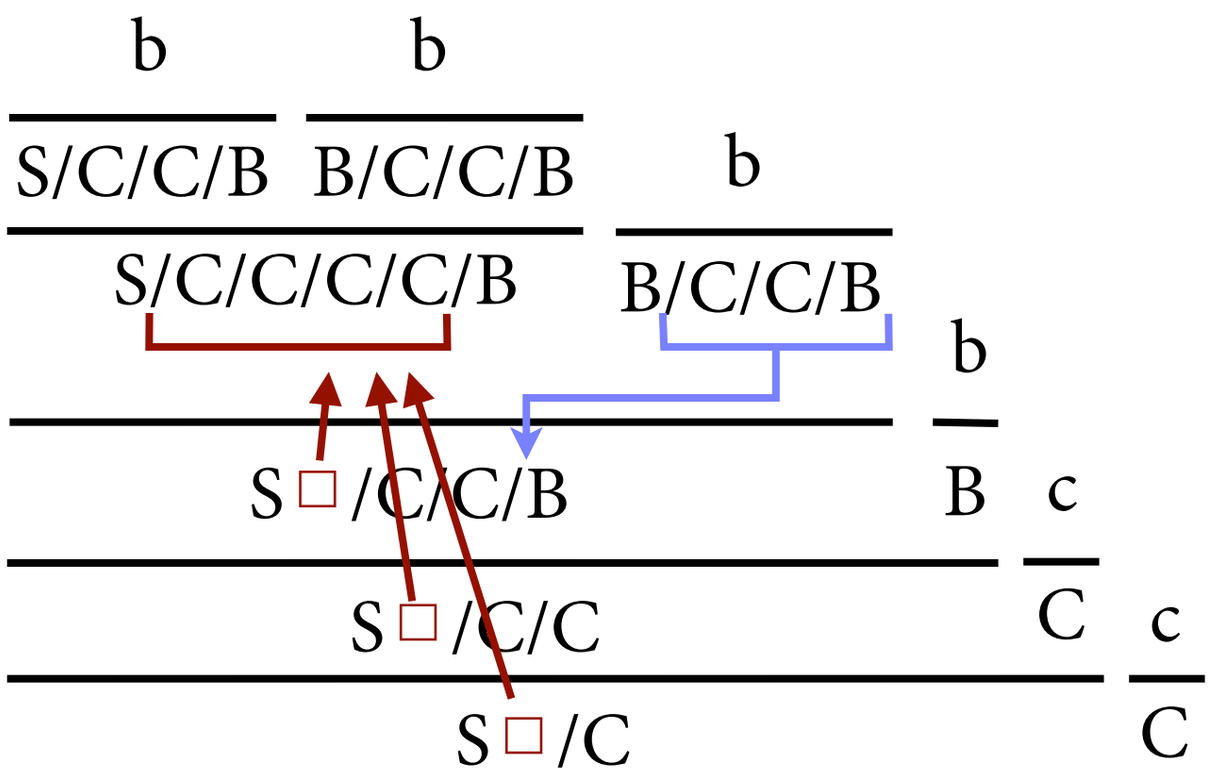
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)



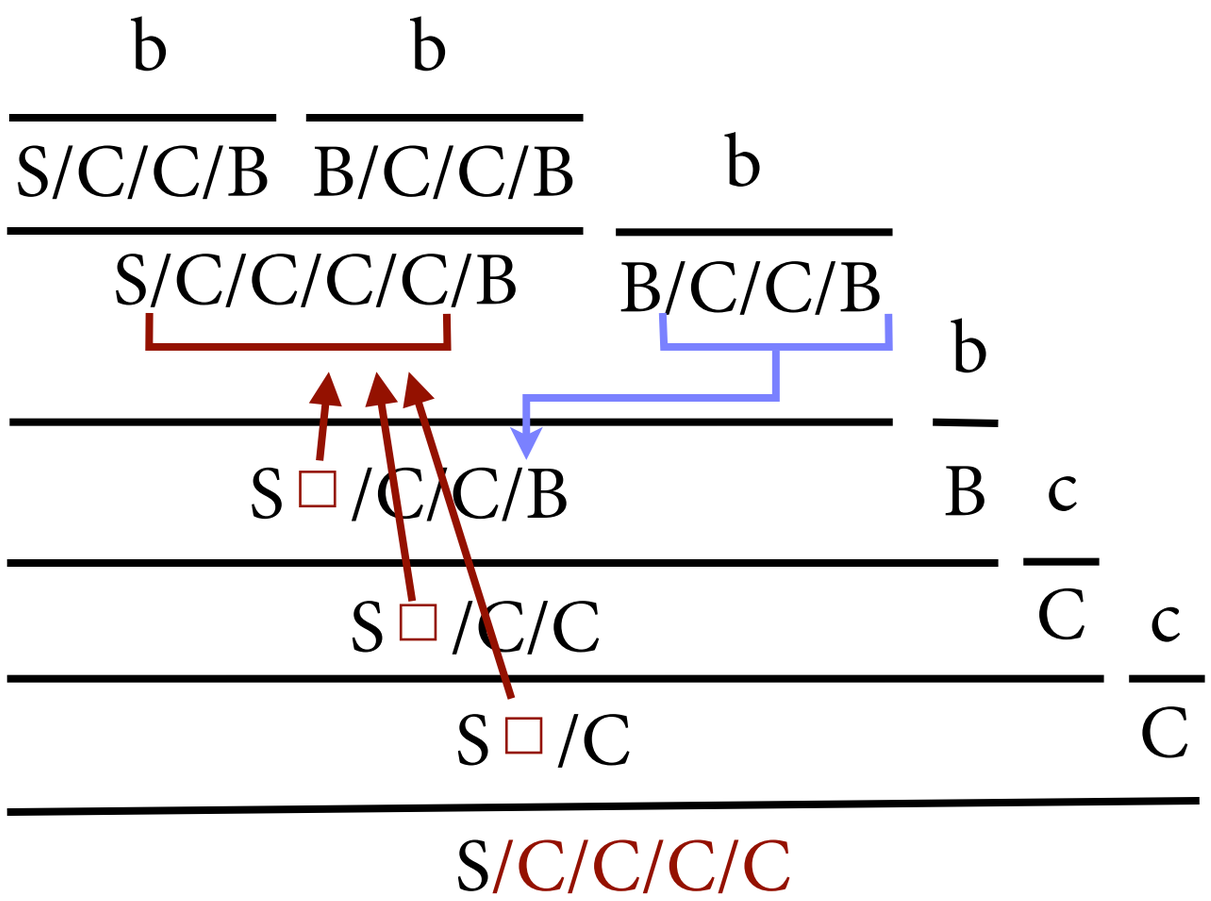
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)



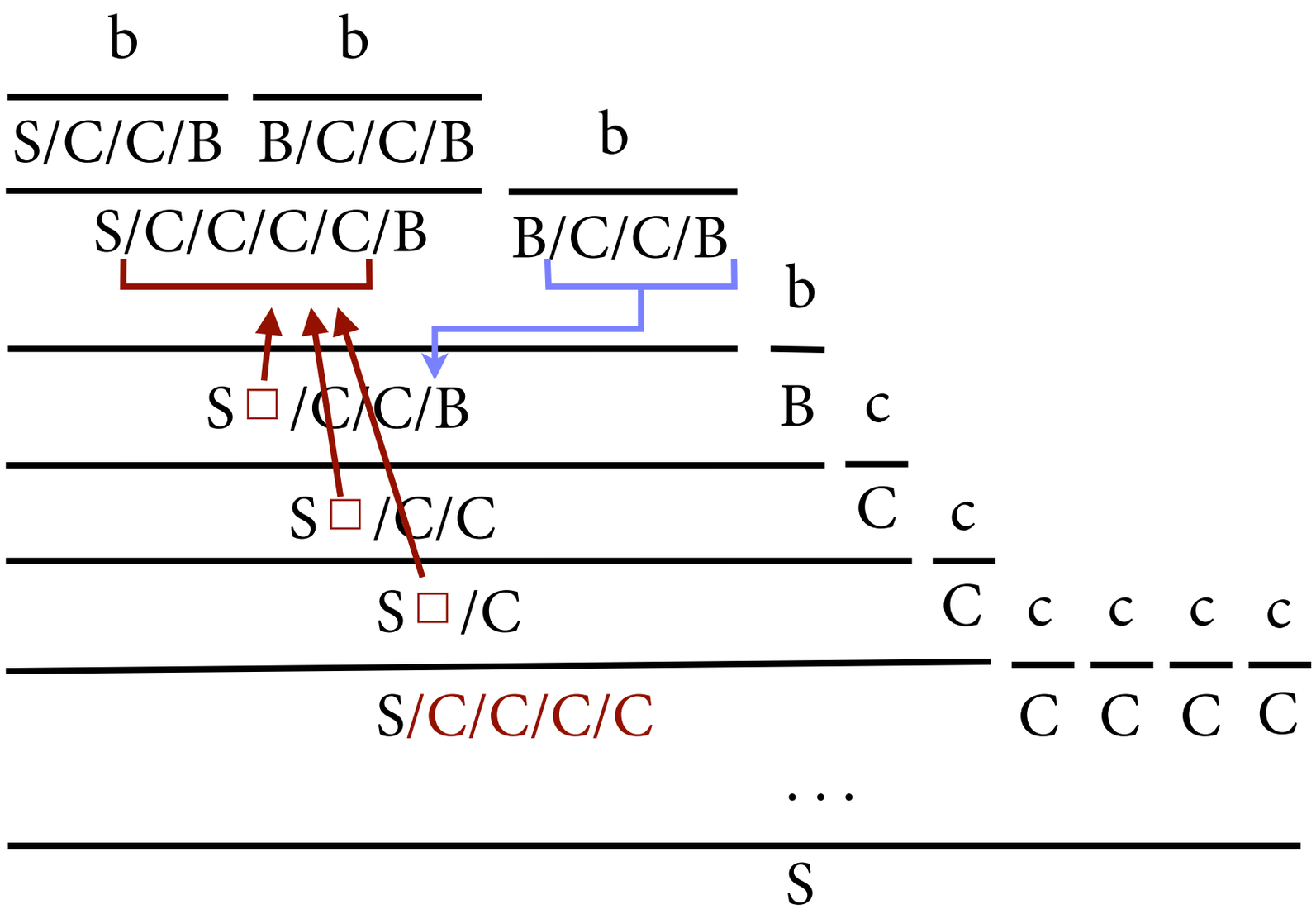
(Vijay-Shanker & Weir 1990)

Lösung (Grundidee)



(Vijay-Shanker & Weir 1990)

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□/C + C + S/C/C/C/C/B ⇒ S/C/C/C/C

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

(Vijay-Shanker & Weir 1990)

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

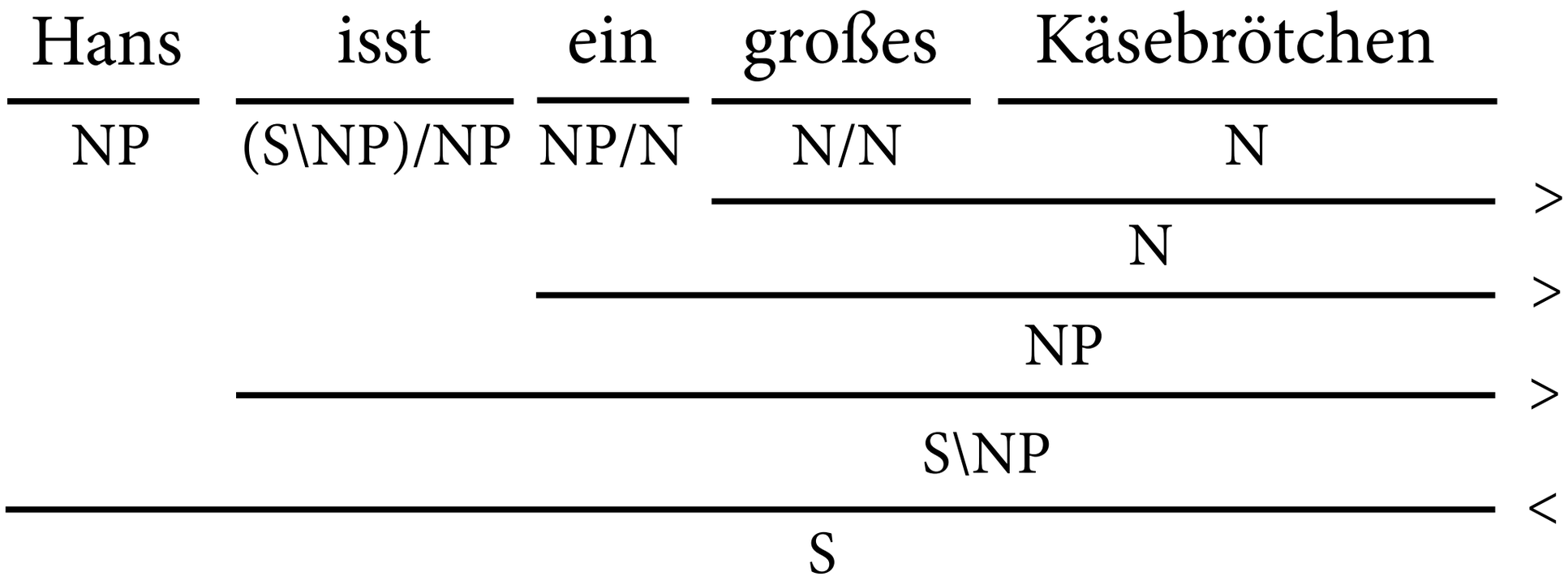
- Extrahiere aus CCG-Parse Dependenz-Tupel:
  $\langle isst_2, (S\backslash NP_1)/NP_2, 2, käsebrot_5 \rangle$
  - "Käsebrot" an Position 5 füllt zweites Argument von "isst" an Position 2.

- Vorgehen:
  - Argumentpositionen lexikalisch spezifizieren:
    isst: $(S\backslash NP_1)/NP_2$
  - für jede Konstituente *Kopfwort* mitführen
  - Regelanwendung etabliert Dependenz zwischen Köpfen

Beispiel

Lexikon:

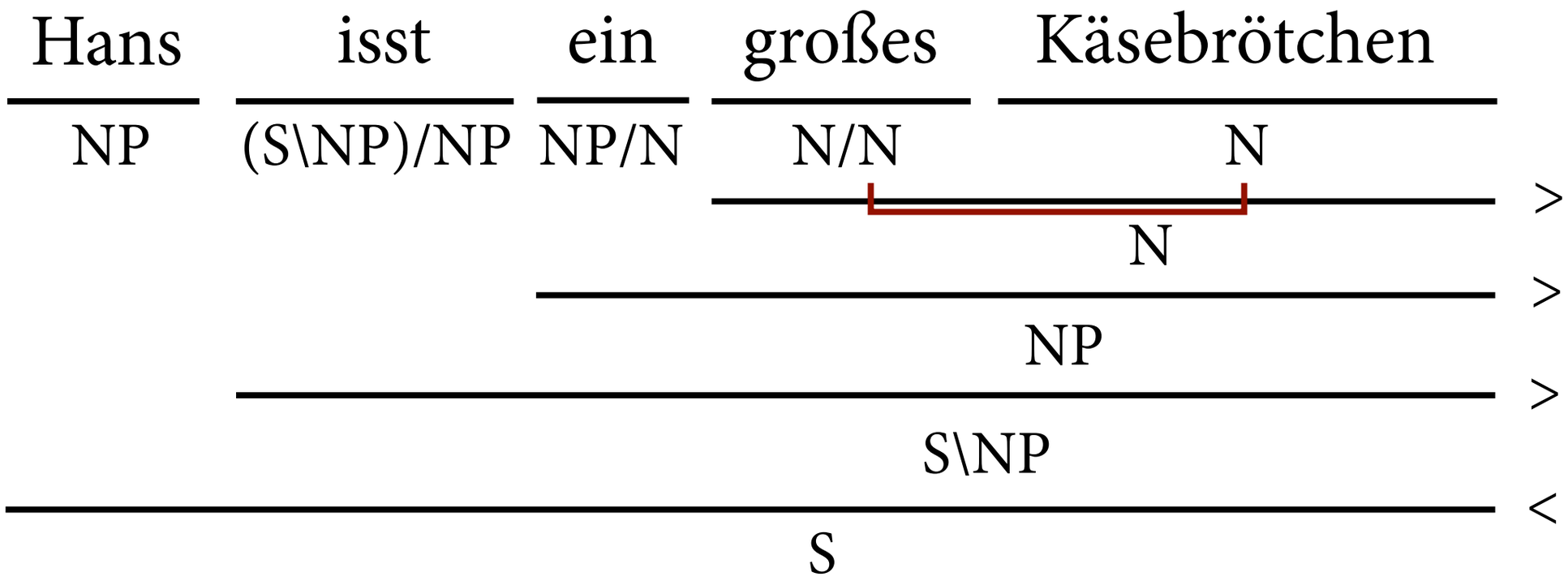
Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	



Beispiel

Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

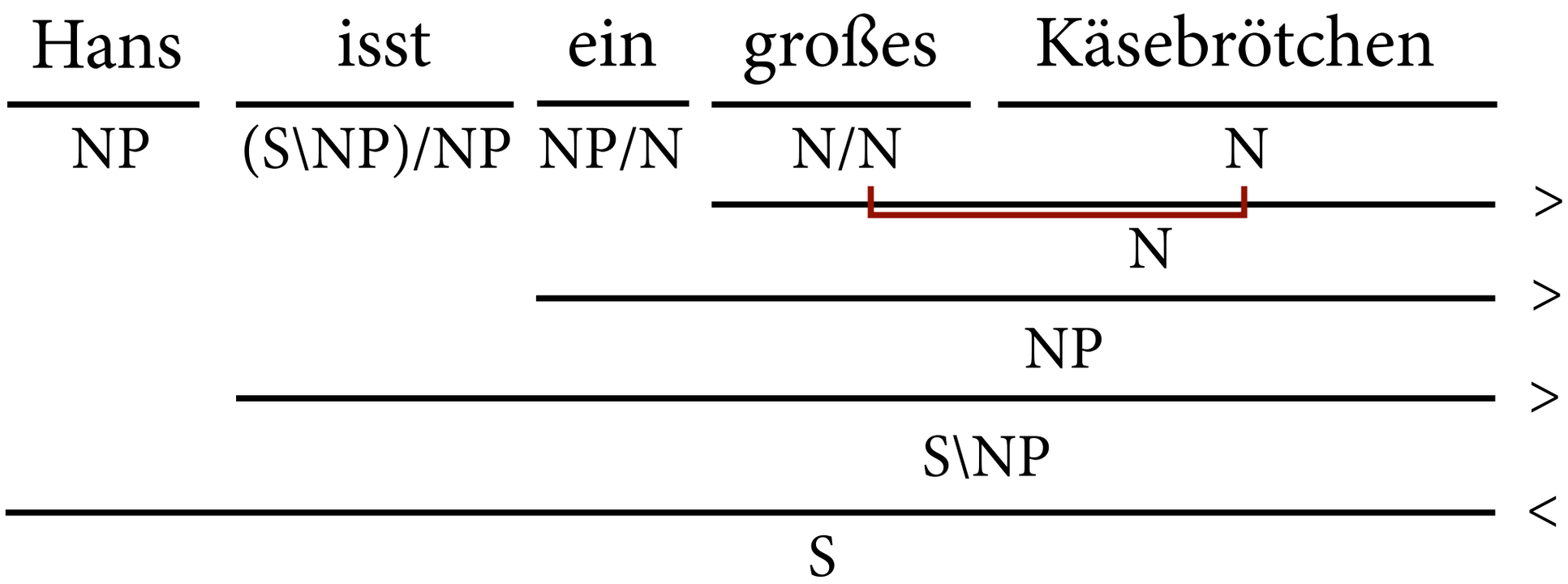


Beispiel

Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

$\langle großes_4, N/N_1, 1, KB_5 \rangle$

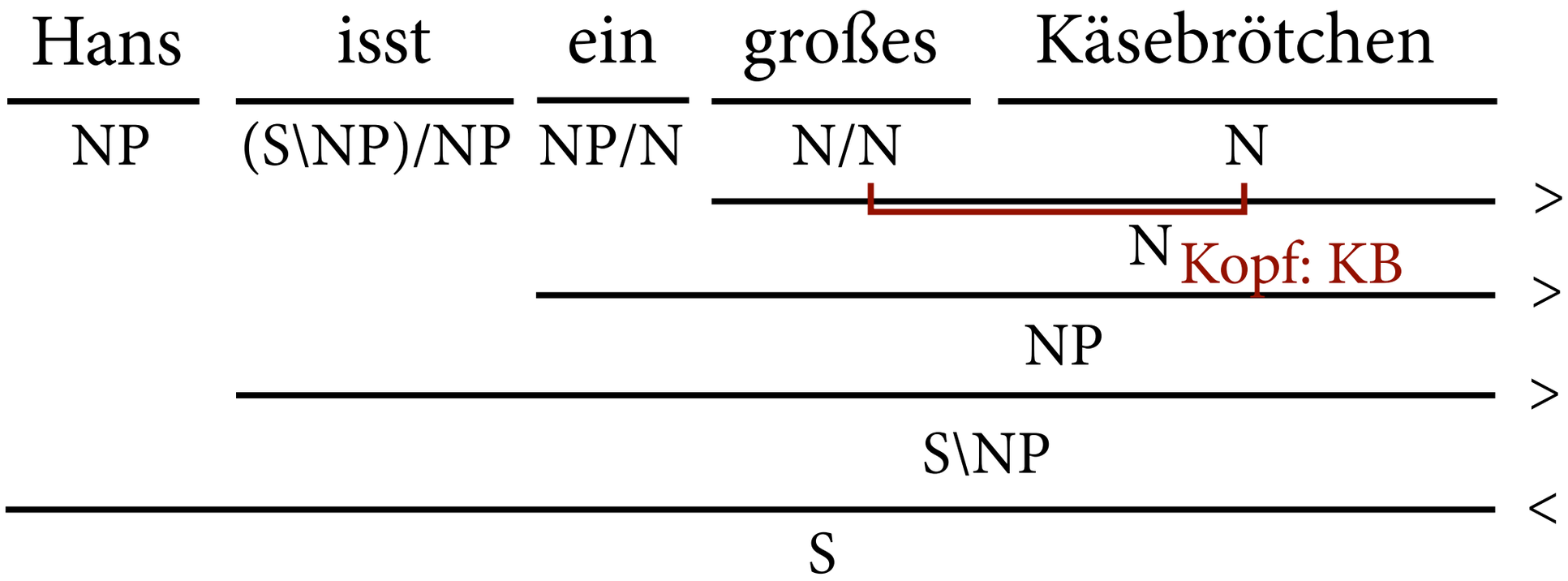


Beispiel

Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

$\langle großes_4, N/N_1, 1, KB_5 \rangle$

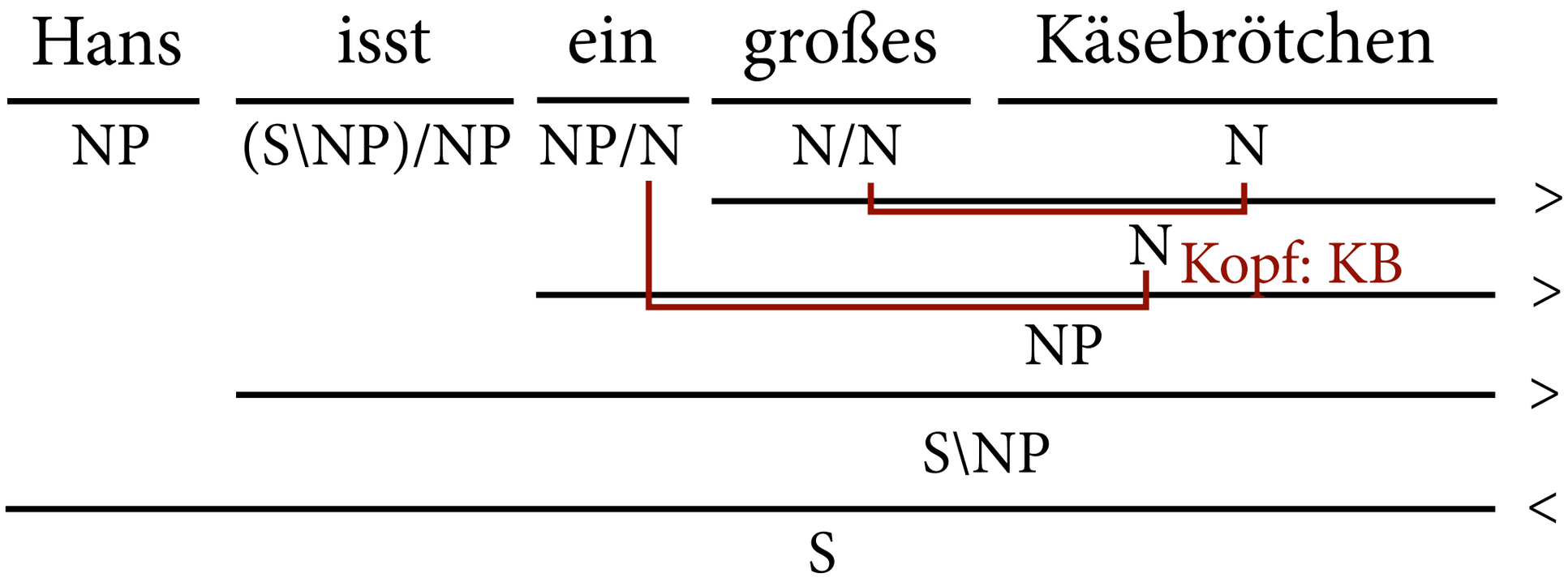


Beispiel

Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

$\langle großes_4, N/N_1, 1, KB_5 \rangle$



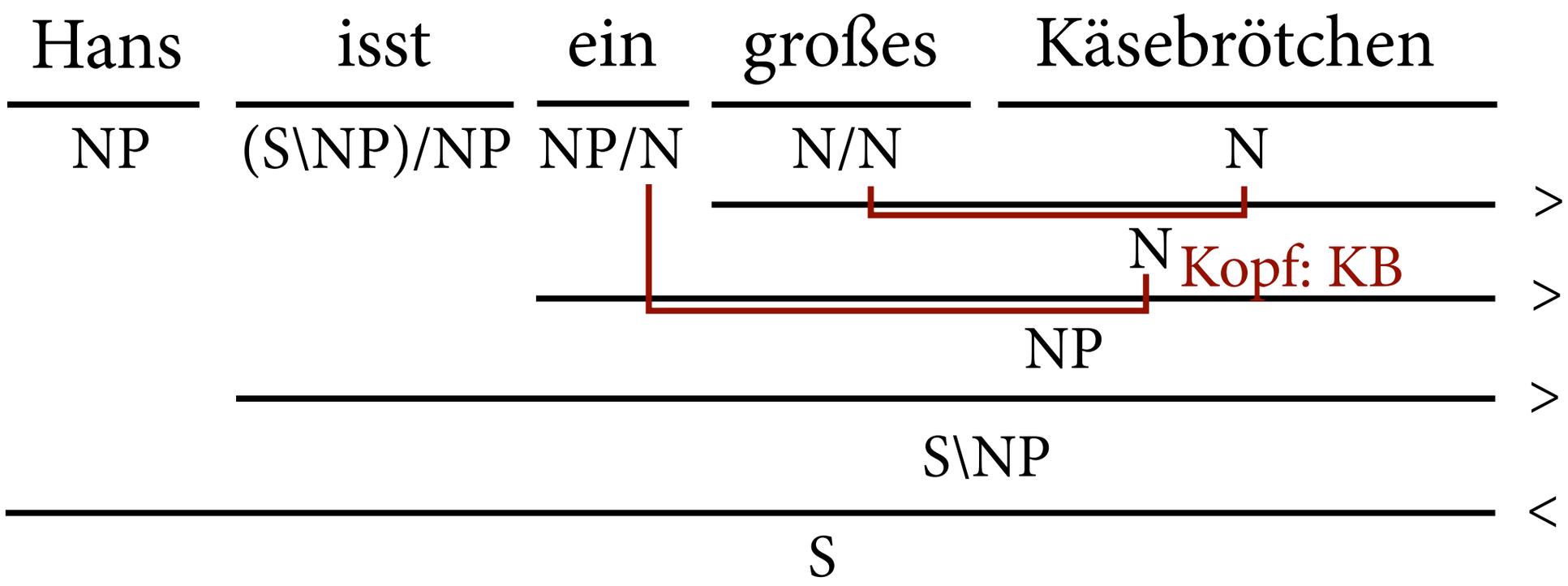
Beispiel

Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

⟨großes$_4$, N/N_1, 1, KB_5⟩

⟨ein$_3$, NP/N_1, 1, KB_5⟩



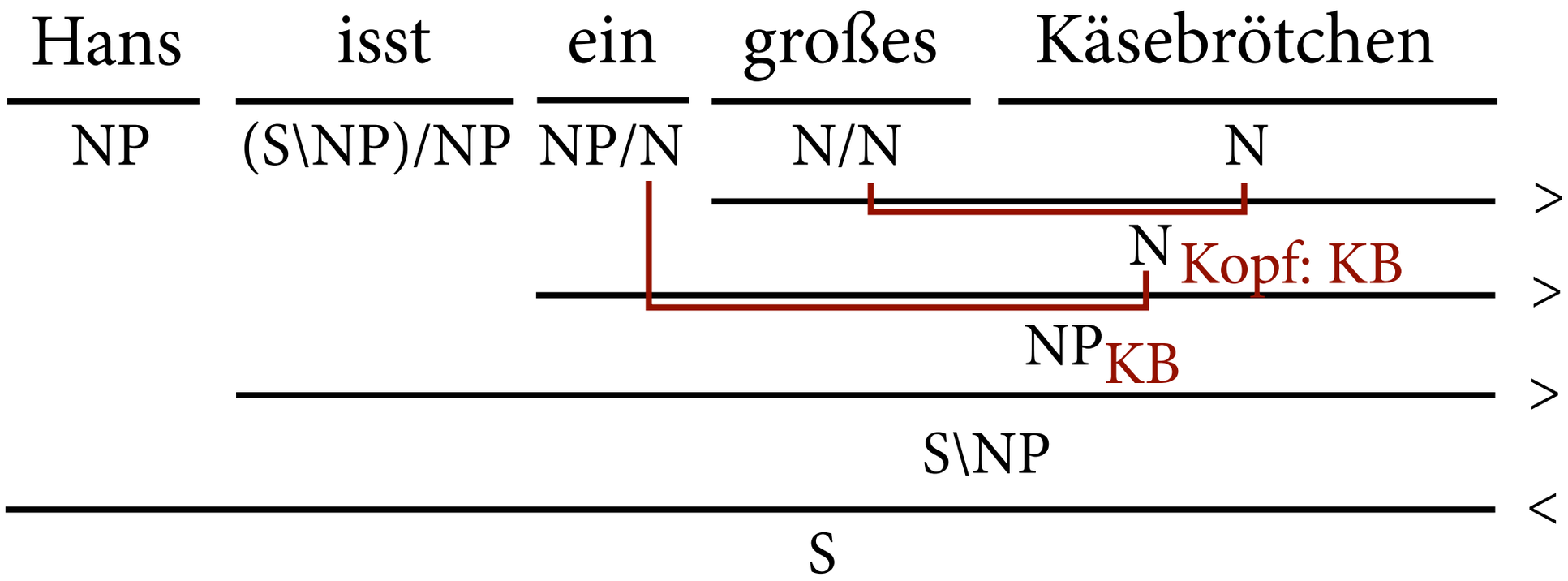
Beispiel

Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

⟨großes$_4$, N/N_1, 1, KB_5⟩

⟨ein$_3$, NP/N_1, 1, KB_5⟩



Beispiel

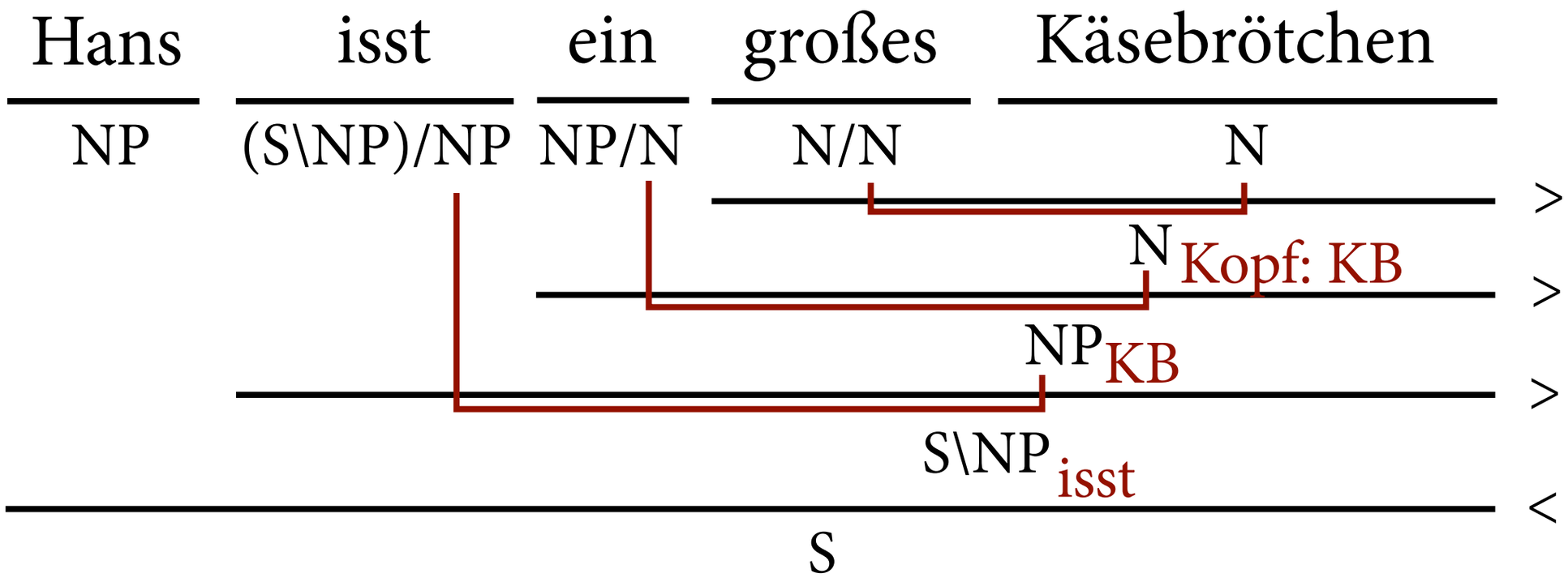
Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

⟨großes$_4$, N/N_1, 1, KB_5⟩

⟨ein$_3$, NP/N_1, 1, KB_5⟩

⟨isst$_2$, $(S\backslash NP_1)/NP_2$, 2, KB_5⟩



Beispiel

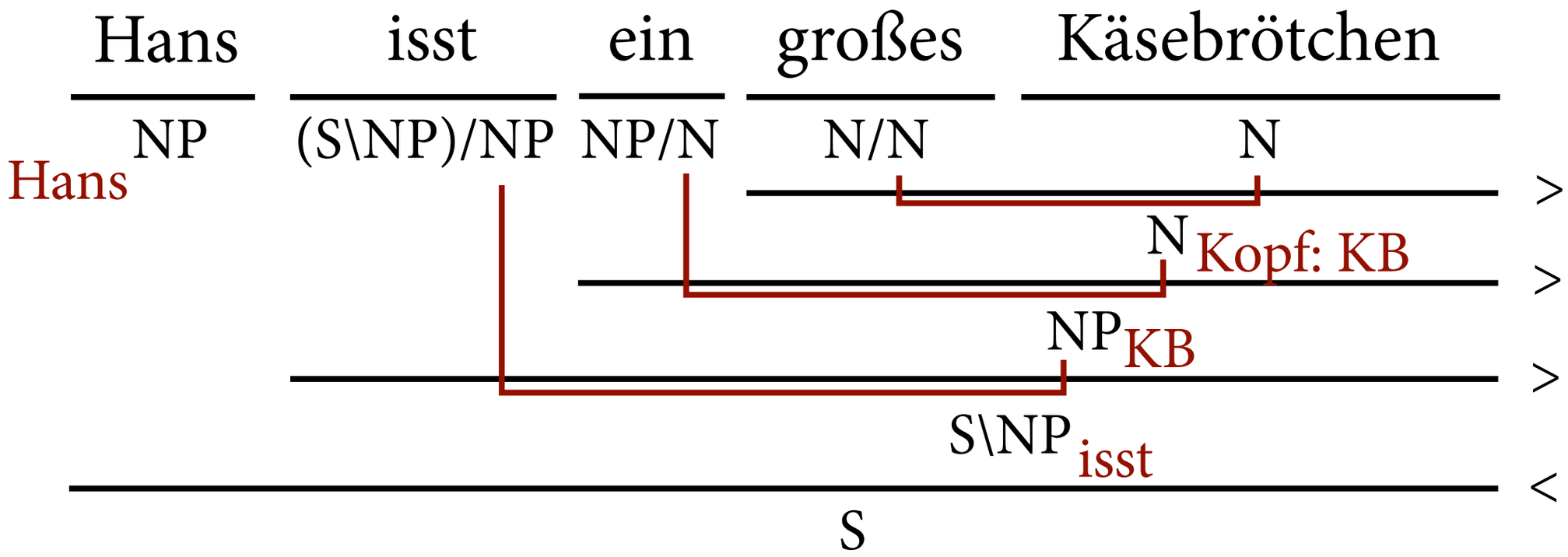
Lexikon:

Hans: NP	ein: NP/N_1
isst: $(S\backslash NP_1)/NP_2$	Käsebrötchen: N
großes: N/N_1	

$\langle großes_4, N/N_1, 1, KB_5 \rangle$

$\langle ein_3, NP/N_1, 1, KB_5 \rangle$

$\langle isst_2, (S\backslash NP_1)/NP_2, 2, KB_5 \rangle$



Beispiel

Lexikon:

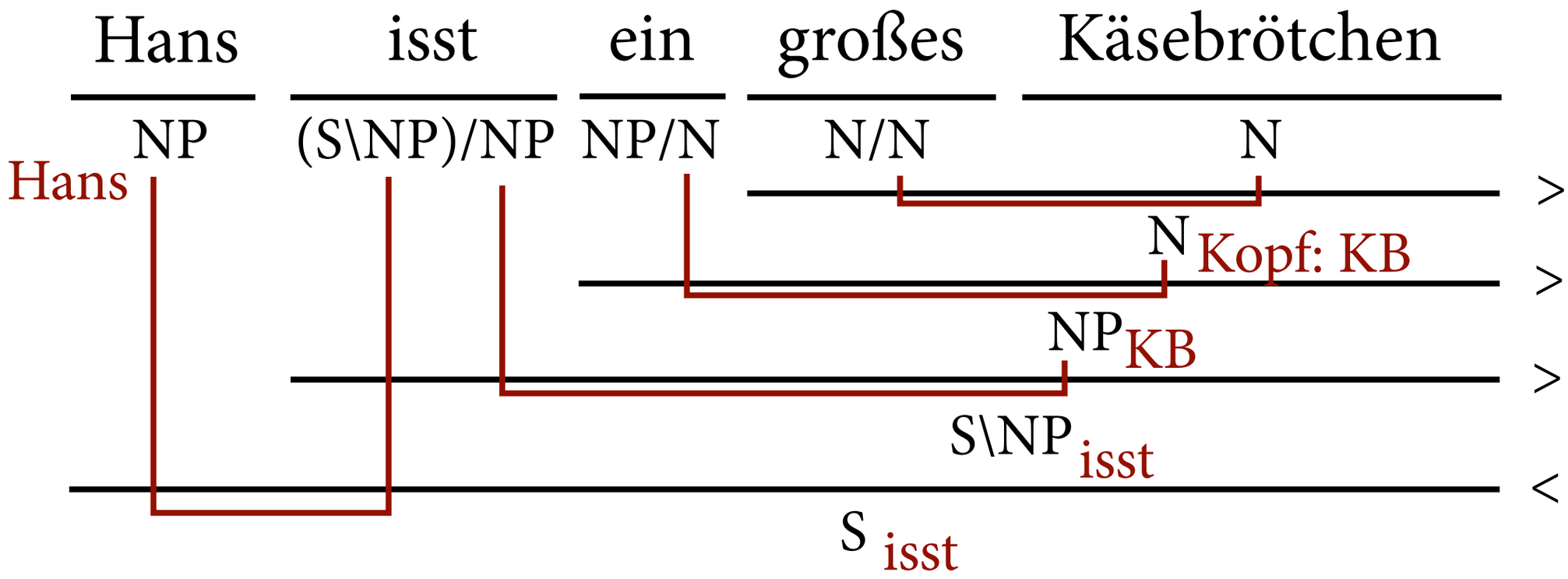
Hans: NP	ein: NP/N_1
isst: (S\NP$_1$)/NP$_2$	Käsebrötchen: N
großes: N/N_1	

⟨großes$_4$, N/N_1, 1, KB_5⟩

⟨ein$_3$, NP/N_1, 1, KB_5⟩

⟨isst$_2$, (S\NP$_1$)/NP$_2$, 2, KB_5⟩

⟨isst$_2$, (S\NP$_1$)/NP$_2$, 1, Hans$_1$⟩



Log-lineare W.modelle

- Extrem 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 | 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	$(S/S)/NP$ + Before
LexCat + POS	$(S/S)/NP$ + IN
RootCat	$S[dcl]$
RootCat + Word	$S[dcl]$ + was
RootCat + POS	$S[dcl]$ + VBD
Rule	$S[dcl] \rightarrow NP\ S[dcl]\backslash NP$
Rule + Word	$S[dcl] \rightarrow NP\ S[dcl]\backslash NP$ + bought
Rule + POS	$S[dcl] \rightarrow NP\ S[dcl]\backslash NP$ + VBD

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

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 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.

(Lewis & Steedman 14; Lewis et al. 16)

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*-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*-Suche:
  - Bisherige W. für String mit Supertags $s_i, \ldots, s_k$:
    $P(s_i \mid w) * \ldots * P(s_k \mid w)$
  - Schätzung für restliche Kosten:
    $(\max P(s_1 \mid w)) * \ldots * (\max P(s_{i-1} \mid w))$
    $* (\max P(s_{k+1} \mid w)) * \ldots * (\max P(s_n \mid w))$

Parsing-Algorithmus

1. Supertagging:
   Berechne für jedes Wort die W. für alle lex. Kategorien.

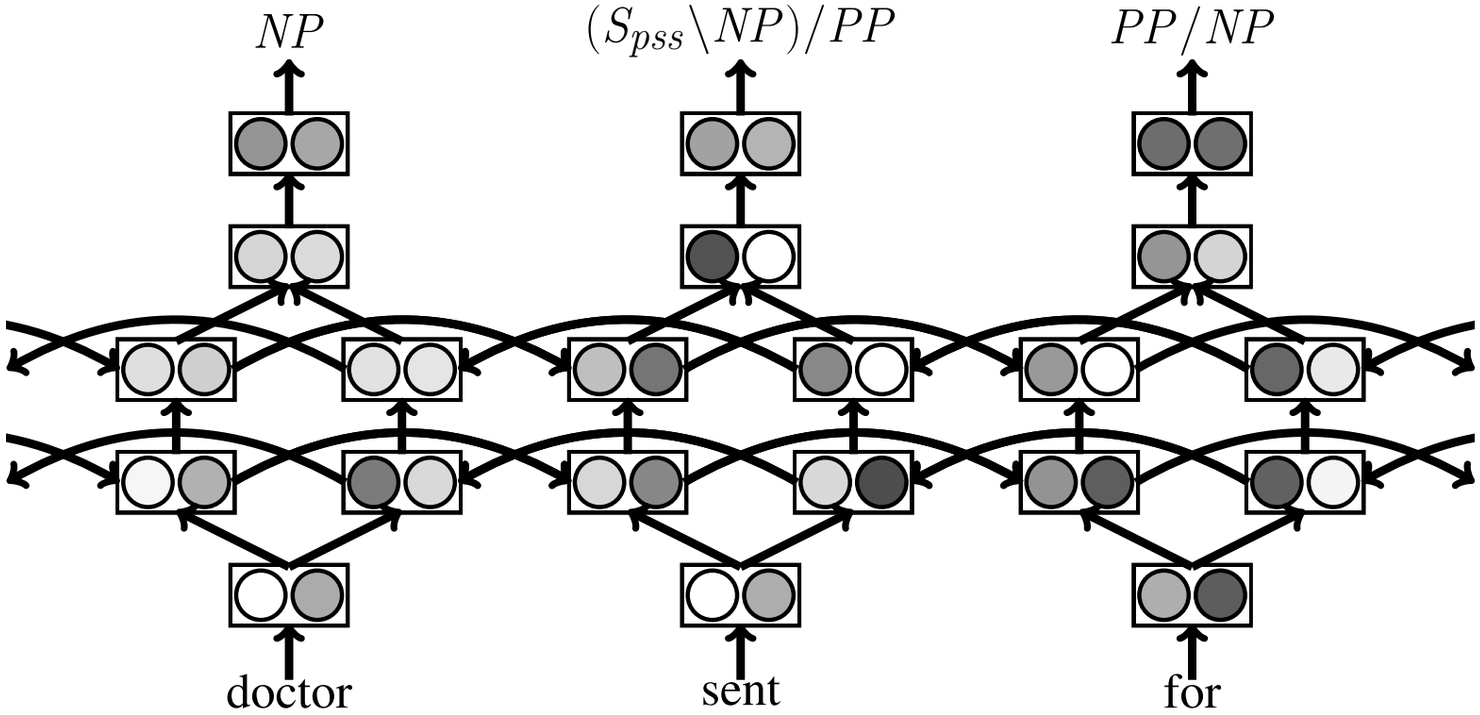
2. A*-Parsing:
   So lange Items von der Agenda nehmen und neue Items in Chart + Agenda einsortieren, bis Ziel-Item gefunden wurde.

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

(Lewis & Steedman 14)

Supertagging für CCGs

- Mit neuronalen Netzwerken (bidirectional LSTM):



(Lewis et al. 16)

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	**94.9**	**94.7**

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.

(Lewis et al. 16)

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	**88.6**	**87.5**	**88.1**
LSTM + Tri-training + Dependencies	88.2	87.3	87.8

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

(Lewis et al. 16)

Evaluation: Geschwindigkeit

Parser	Sentences per second
SpaCy*[4]	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.

(Lewis et al. 16)

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.