TAG:
Statistisches Parsing mit TAG

Vorlesung “Grammatikformalismen”

Alexander Koller

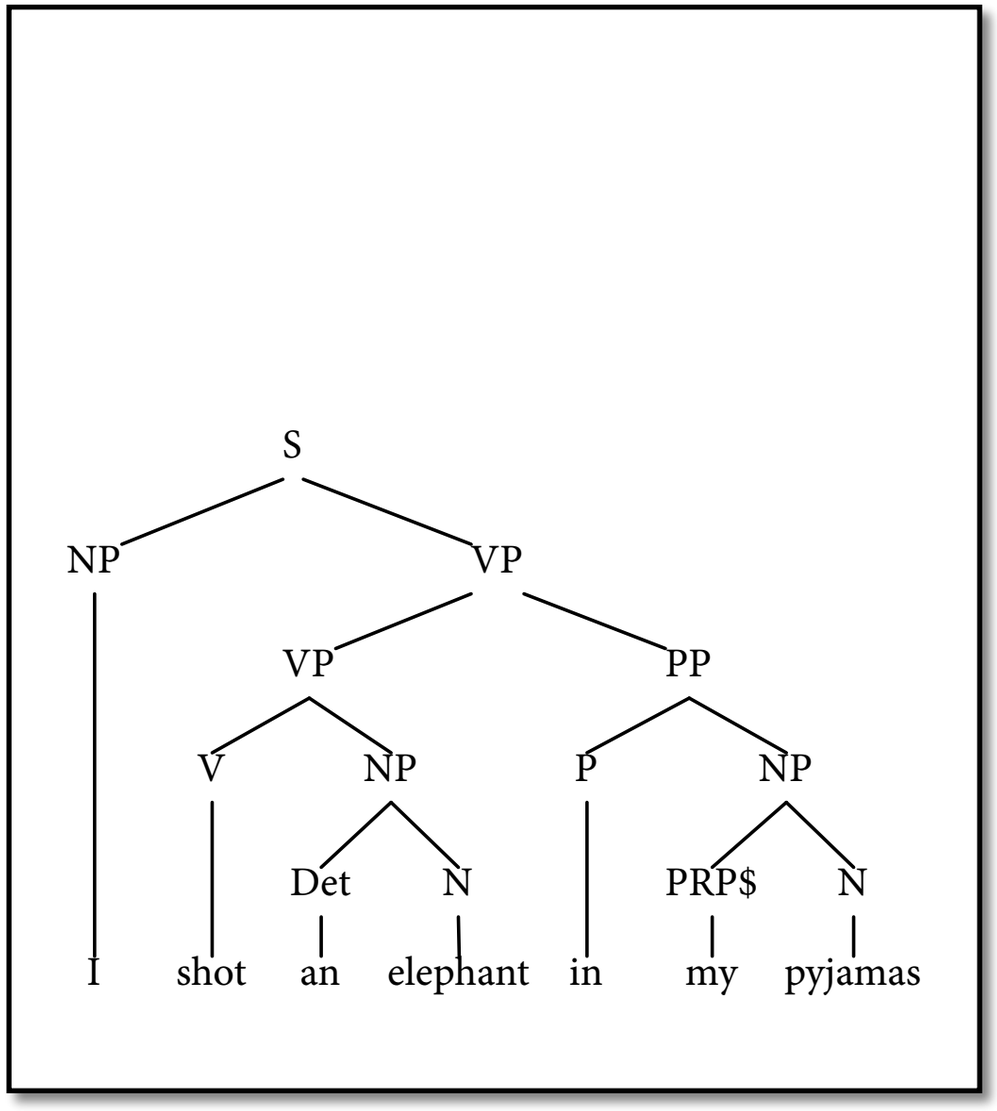
3. Mai 2019

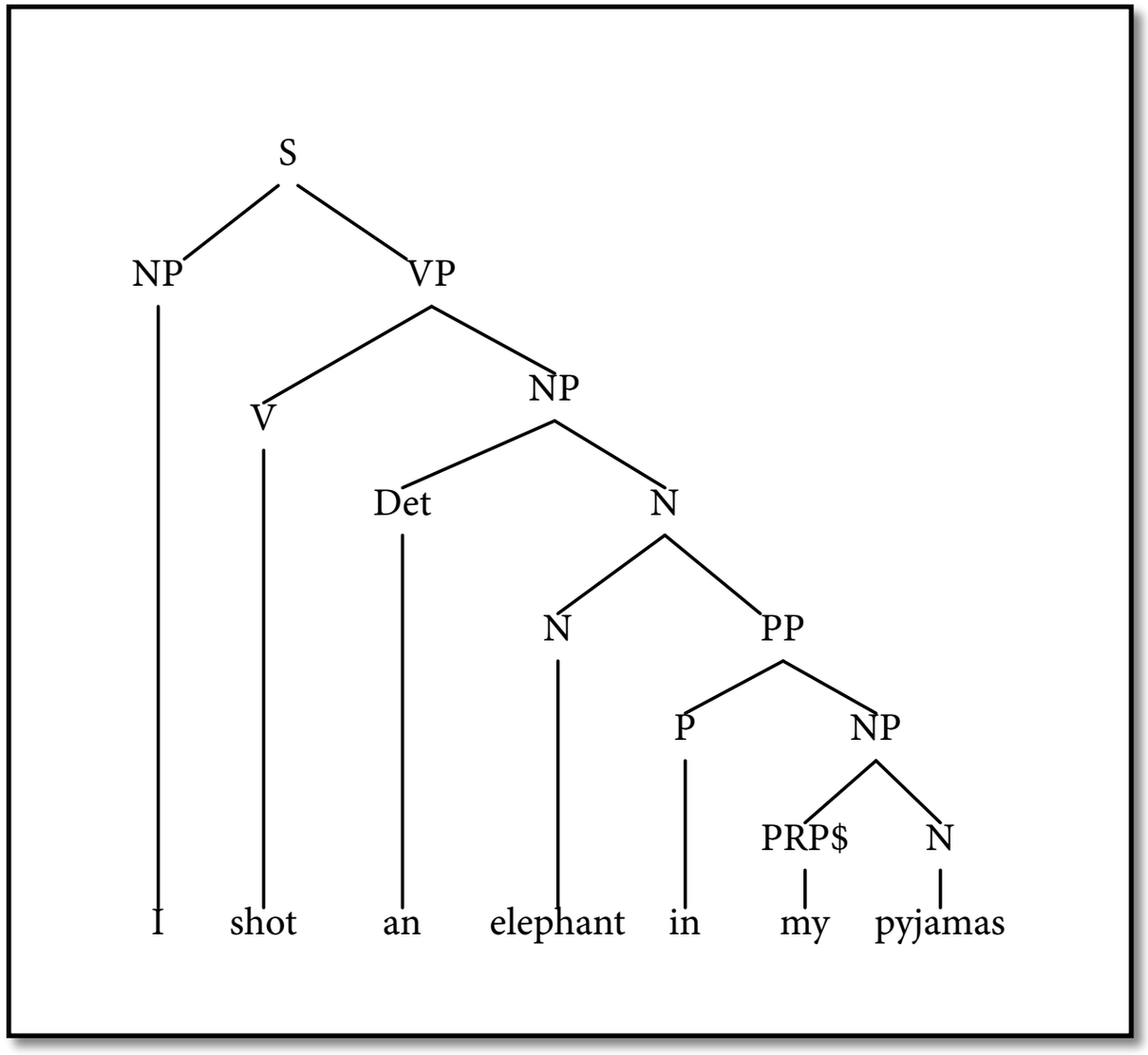
Statistisches Parsing

- Letztes Mal: Parsingalgorithmus für TAG, Laufzeit $O(n^6)$.
- Problem: Grammatiken mit hoher Abdeckung sind sehr ambig — wie finden wir den besten Parse?
  - Ansatz: Probabilistische Grammatiken.
- Problem: Woher kriegen wir eine Grammatik mit hoher Abdeckung?
  - handgeschriebene für einzelne Sprachen: EN, DE, FR
  - Ansatz: Aus Baumbank lernen.

Ambiguitäten

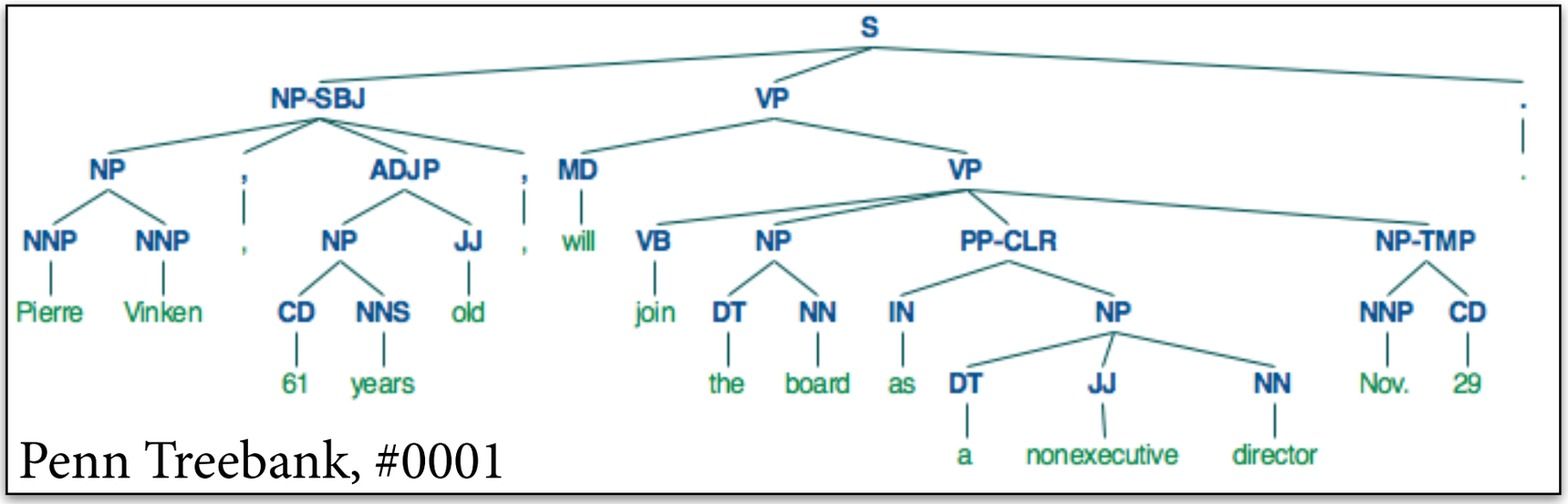
Wir wollen *disambiguieren*, d.h. “korrekten” Parse für ambigen Satz berechnen.



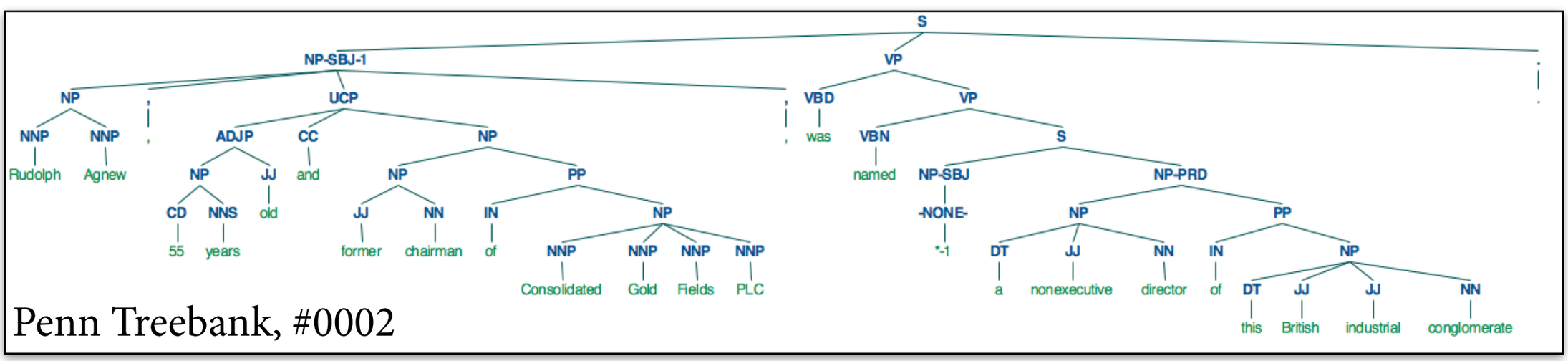


Woran erkennen wir den “korrekten” Baum?

Baumbanken



Penn Treebank, #0001



Penn Treebank, #0002

```
nltk.corpus.treebank.parsed_sents("wsj_0001.mrg")[0].draw()
```

Probabilistische kfGen

- Eine *probabilistische kontextfreie Grammatik (PCFG)* ist eine kfG, in der
  - jede Produktionsregel A → w hat eine W. P(A → w | A): wenn wir A expandieren, wie w. ist Regel A → w?
  - für jedes Nichtterminal A müssen W. zu eins summieren:

$$\sum_{w} P(A \rightarrow w \mid A) = 1$$

  - wir schreiben abgekürzt P(A → w) für P(A → w | A)

Beispiel

S → NP VP	[1.0]	VP → V NP	[0.5]
NP → Det N	[0.8]	VP → VP PP	[0.5]
NP → i	[0.2]	V → shot	[1.0]
N → N PP	[0.4]	PP → P NP	[1.0]
N → elephant	[0.3]	P → in	[1.0]
N → pyjamas	[0.3]	Det → an	[0.5]
		Det → my	[0.5]

Generativer Prozess

- PCFG erzeugt zufällige Ableitung der kfG.
  - Ereignis = Expansion von NT durch Produktionsregel
  - alle statistisch unabhängig voneinander

$$S \overset{1.0}{\Rightarrow} NP\ \ VP \overset{0.2}{\Rightarrow} i\ VP \overset{0.5}{\Rightarrow} i\ VP\ PP$$

$$\Rightarrow^* \text{i shot an elephant in my pyjamas}$$

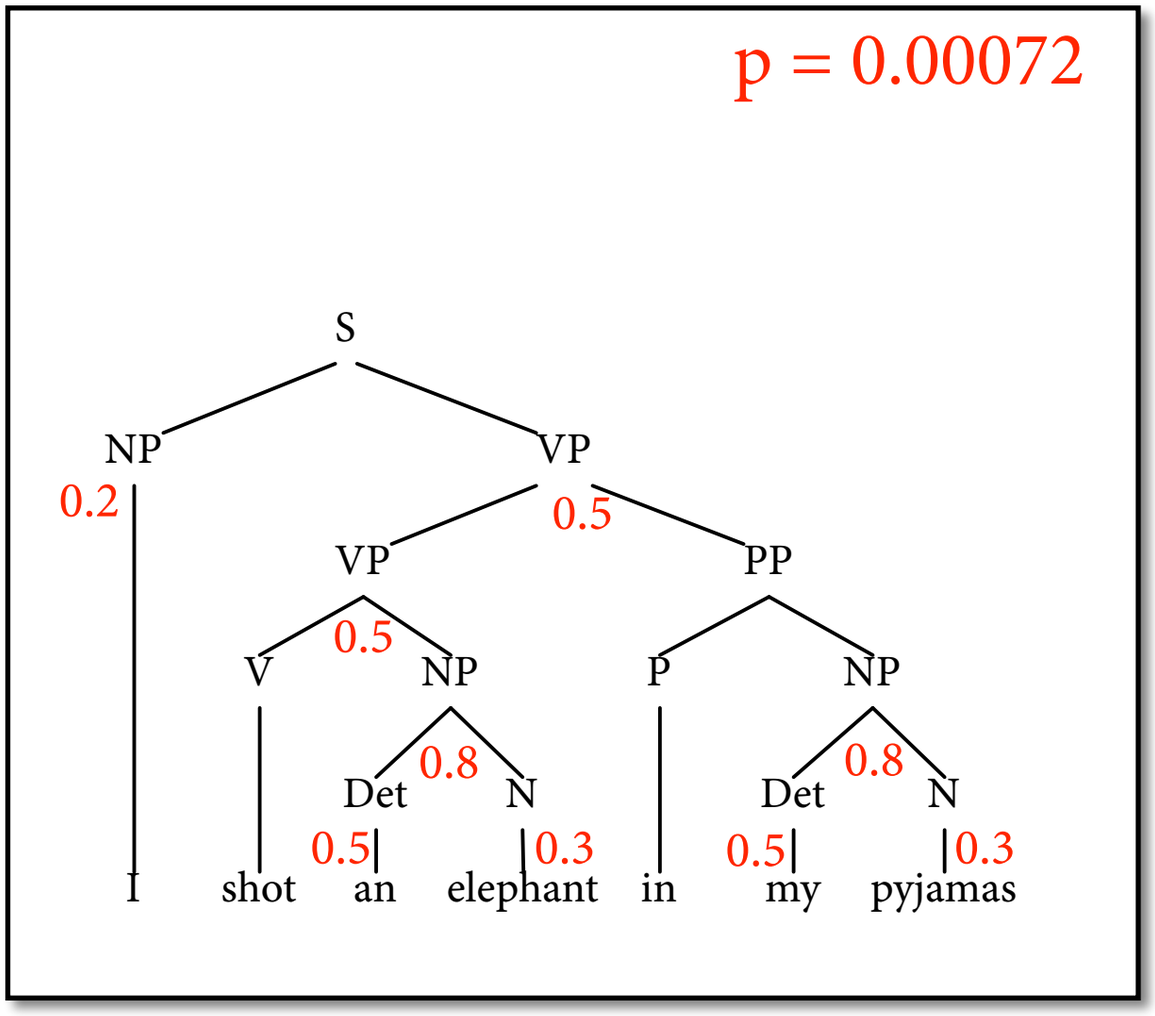
0.00072

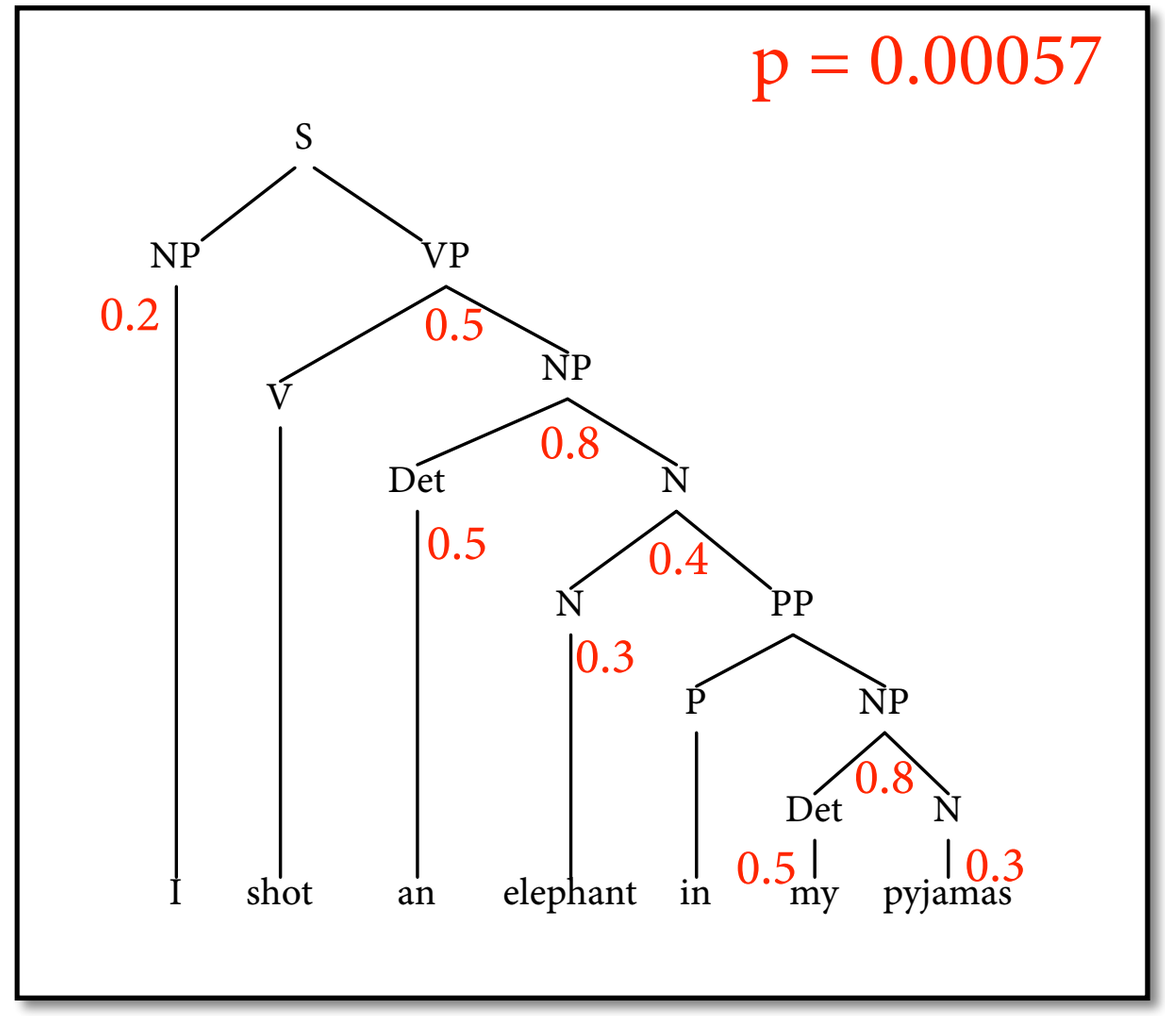
$$S \overset{1.0}{\Rightarrow} NP\ VP \overset{0.2}{\Rightarrow} i\ VP \overset{0.4}{\Rightarrow}^* i\ V\ Det\ N$$

$$\underset{0.4}{\Rightarrow} i\ V\ Det\ N\ PP \Rightarrow^* \text{i shot} \ldots \text{pyjamas}$$

0.00057

Parsebäume





“korrekt” = wahrscheinlichster Parsebaum

Algorithmen für PCFGs

- Erweitere CKY-Parser um *Viterbi-Algorithmus:* berechnet besten Parsebaum aus der Chart.

- Lies aus Baumbank (z.B. Penn Treebank) eine kfG mit hoher Abdeckung ab.

- Schätze mit *Maximum Likelihood Estimation (MLE)* die Regelw. als relative Häufigkeiten:

$$P(A \to w) = \frac{C(A \to w)}{C(A \to \bullet)} = \frac{C(A \to w)}{\sum_{w'} C(A \to w')}$$

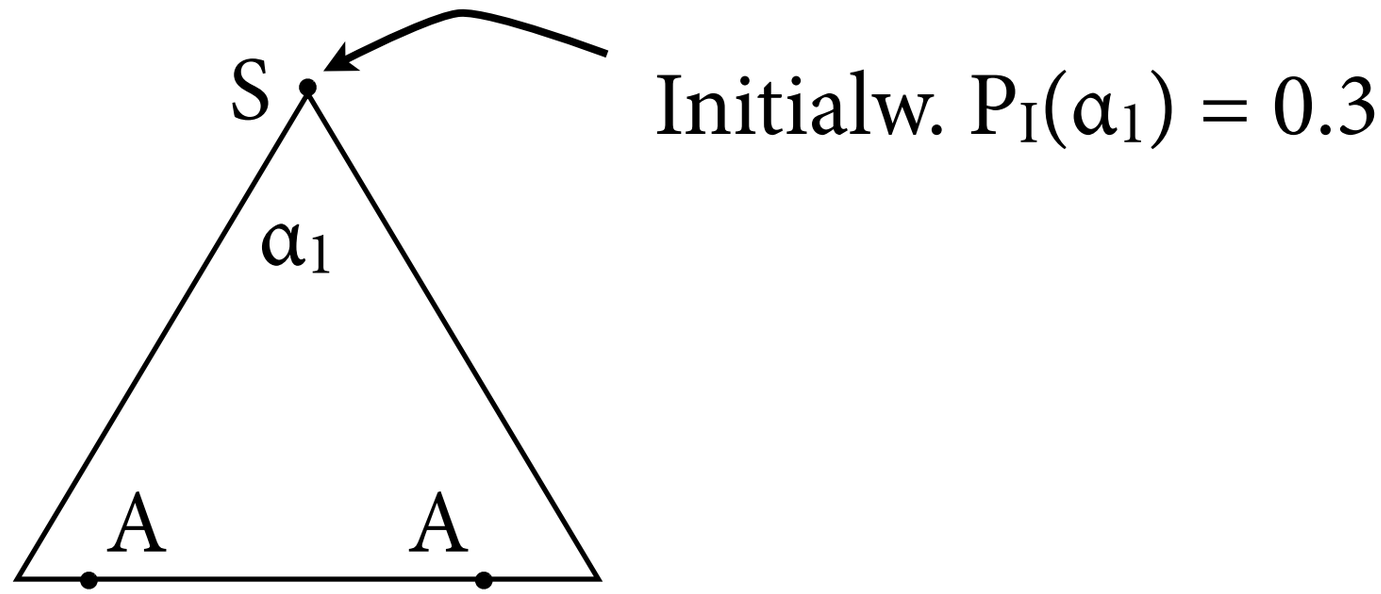
Probabilistische TAG

- Grundidee: Zufallsprozess baut Ableitungsbaum top-down.
- Mögliche Ereignisse (statistisch unabhängig):
  - Ableitung mit Baum α anfangen
  - Substitutionsknoten A↓ mit Baum α füllen
  - An Knoten mit Label A den Baum β adjungieren
  - An Knoten mit Label A nichts adjungieren
- Jedes Ereignis bekommt eigene W.

Probabilistische TSG

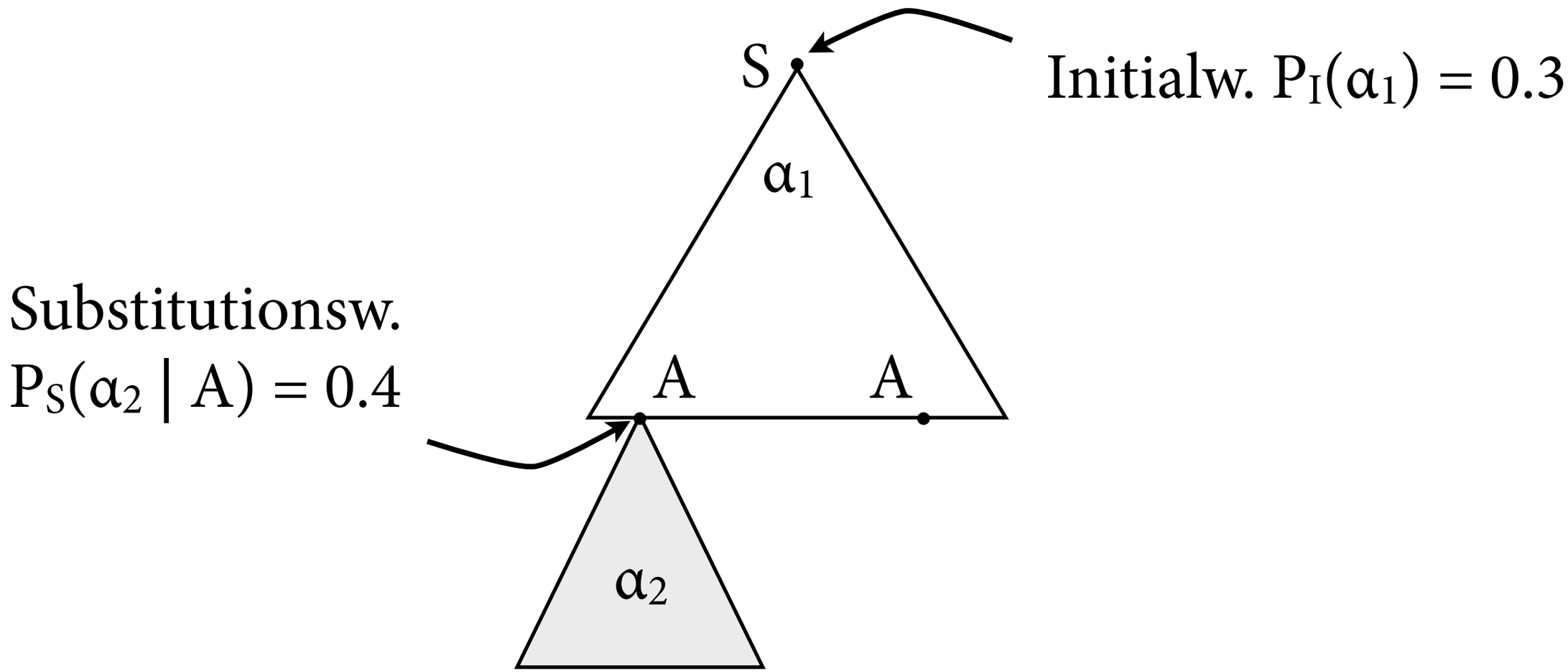
S •

Probabilistische TSG



$$\sum_{\alpha} P_I(\alpha) = 1$$

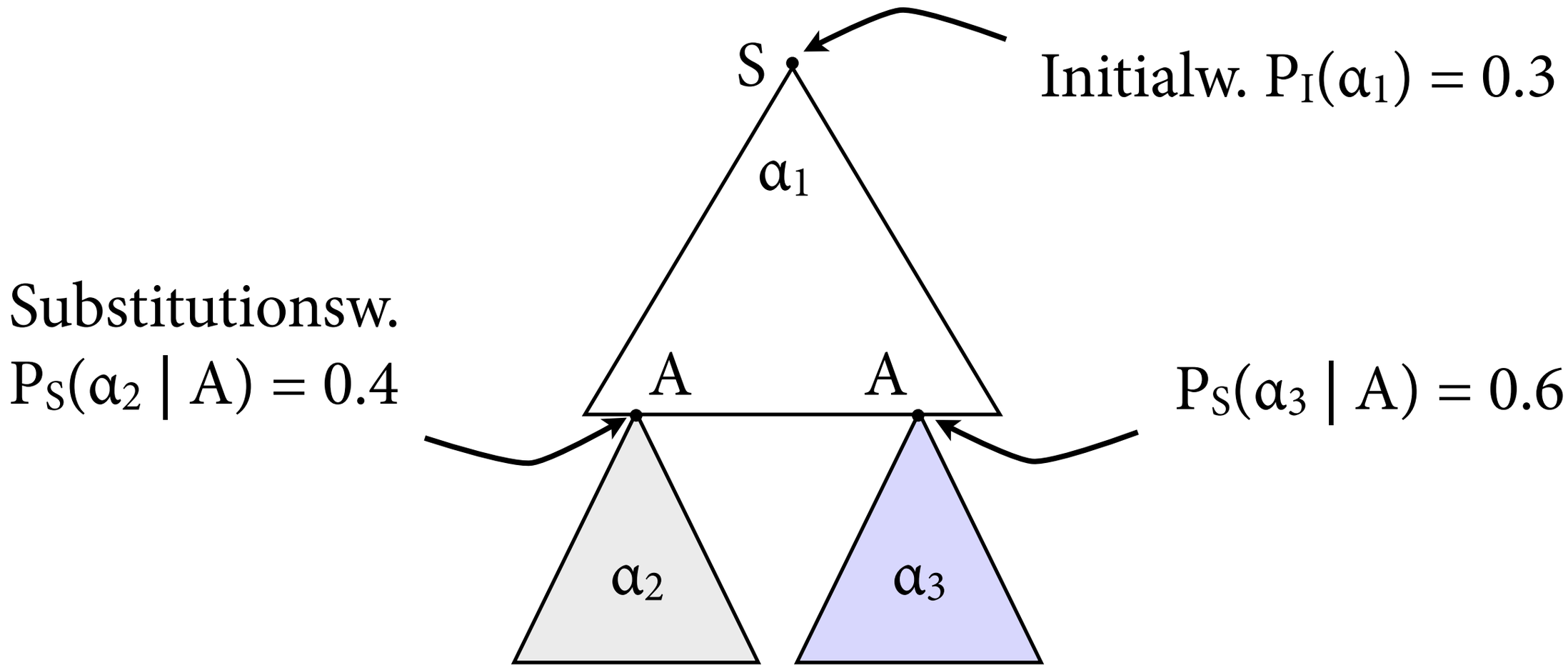
Probabilistische TSG



$$\sum_{\alpha} P_I(\alpha) = 1$$

für alle A: $$\sum_{\alpha} P_S(\alpha|A) = 1$$

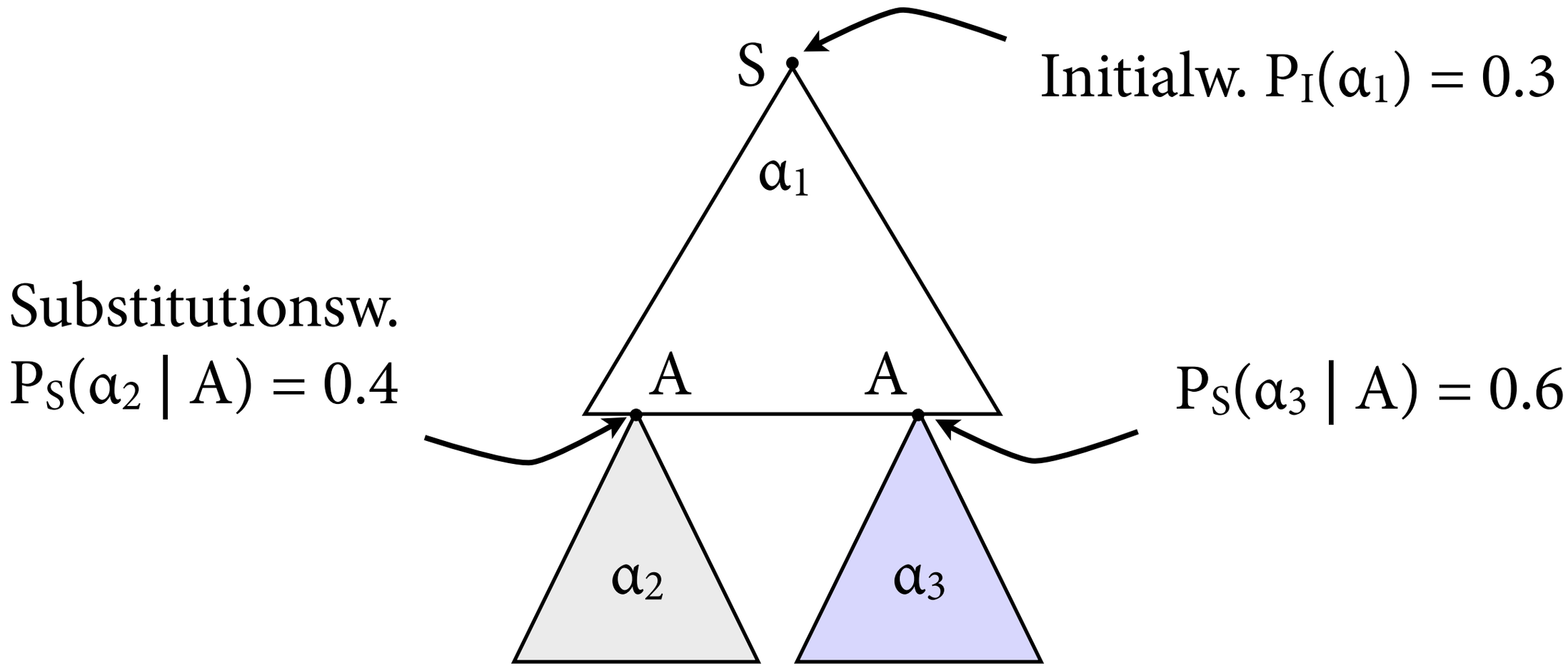
Probabilistische TSG



$$\sum_{\alpha} P_I(\alpha) = 1$$

für alle A: $$\sum_{\alpha} P_S(\alpha|A) = 1$$

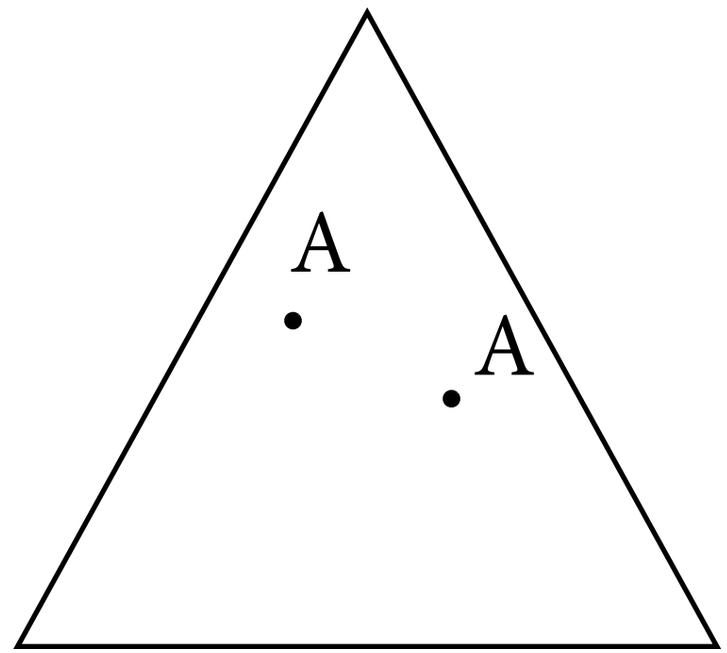
Probabilistische TSG



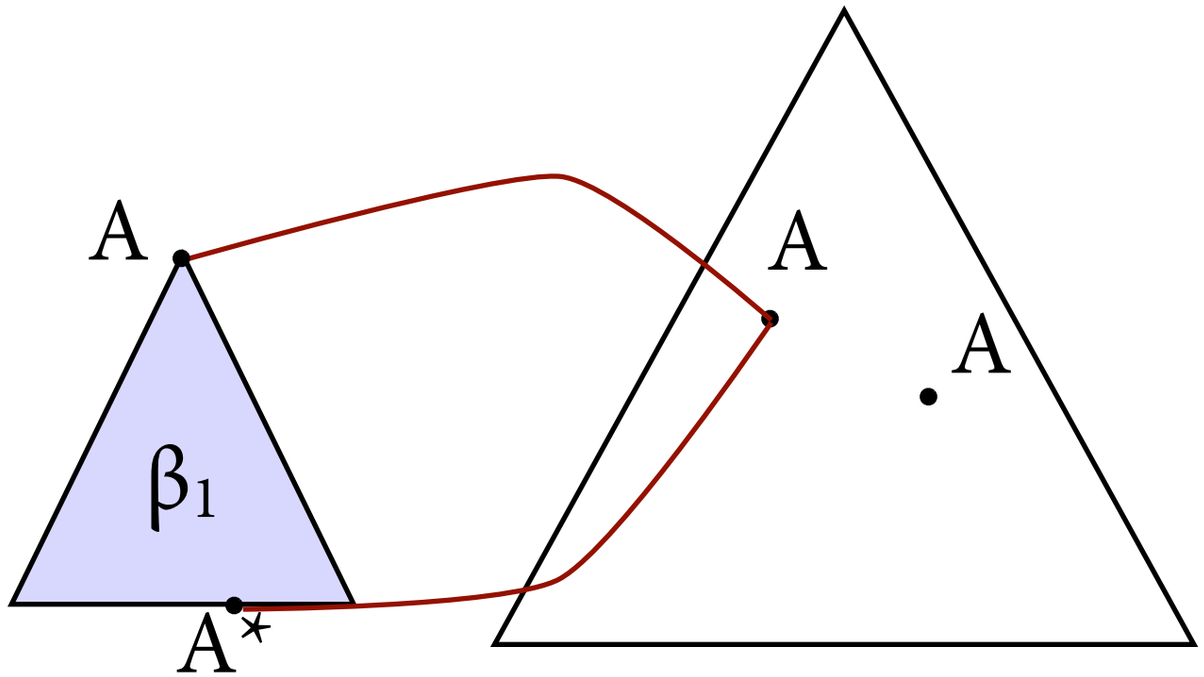
$$P(\alpha_1(\alpha_2, \alpha_3)) = P_I(\alpha_1) \cdot P_S(\alpha_2|A) \cdot P_S(\alpha_3|A) = 0.072$$

$$\sum_{\alpha} P_I(\alpha) = 1 \qquad \text{für alle A:} \quad \sum_{\alpha} P_S(\alpha|A) = 1$$

Probabilistische TAG



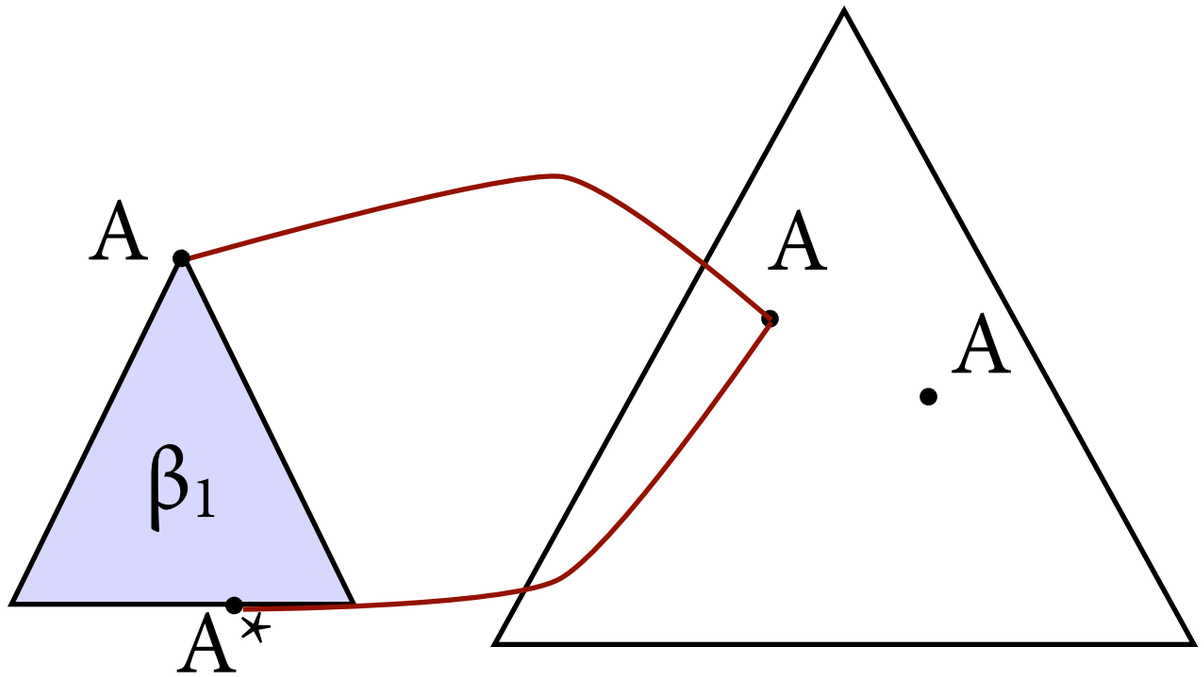
Probabilistische TAG



Adjunktionsw.
$P_A(\beta_1|A) = 0.2$

$$\text{für alle A:} \quad \sum_{\alpha \in \mathsf{Aux} \cup \{\mathsf{none}\}} P_A(\alpha|A) = 1$$

Probabilistische TAG

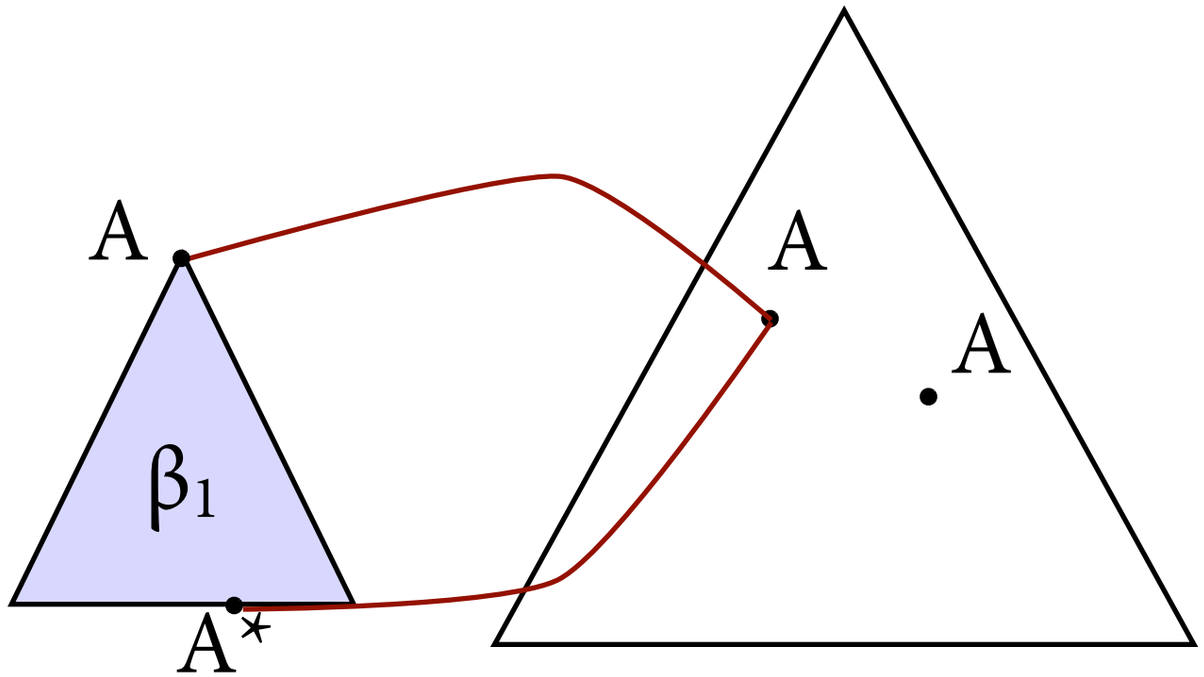


Nicht-Adjunktionsw.
$P_A(\text{none}|A) = 0.8$

Adjunktionsw.
$P_A(\beta_1|A) = 0.2$

$$\text{für alle A:} \quad \sum_{\alpha \in \mathsf{Aux} \cup \{\mathsf{none}\}} P_A(\alpha|A) = 1$$

Probabilistische TAG



Nicht-Adjunktionsw.
$P_A(\text{none}|A) = 0.8$

Adjunktionsw.
$P_A(\beta_1|A) = 0.2$

$$P(\alpha_1(\beta_1)) = P_I(\alpha_1) \cdot P_A(\beta_1|A) \cdot P_A(\mathsf{none}|A) = 0.048$$

$$\text{für alle A:} \quad \sum_{\alpha \in \mathsf{Aux} \cup \{\mathsf{none}\}} P_A(\alpha|A) = 1$$

Algorithmen für PTAG

- Standardalgorithmen für PCFGs übertragen sich weitgehend auf PTAG.

- Parsing: Kombiniere CKY-Parser für TAG (letzte Sitzung) mit Viterbi-Algorithmus.

- Training: Schätze Parameter mit MLE, z.B. so:

$$P_S(\alpha \mid A) = \frac{\alpha \text{ in } A\text{-Knoten substituiert}}{\text{irgendetwas in } A\text{-Knoten substituiert}}$$

Algorithmen für PTAG

- Standardalgorithmen für PCFGs übertragen sich weitgehend auf PTAG.

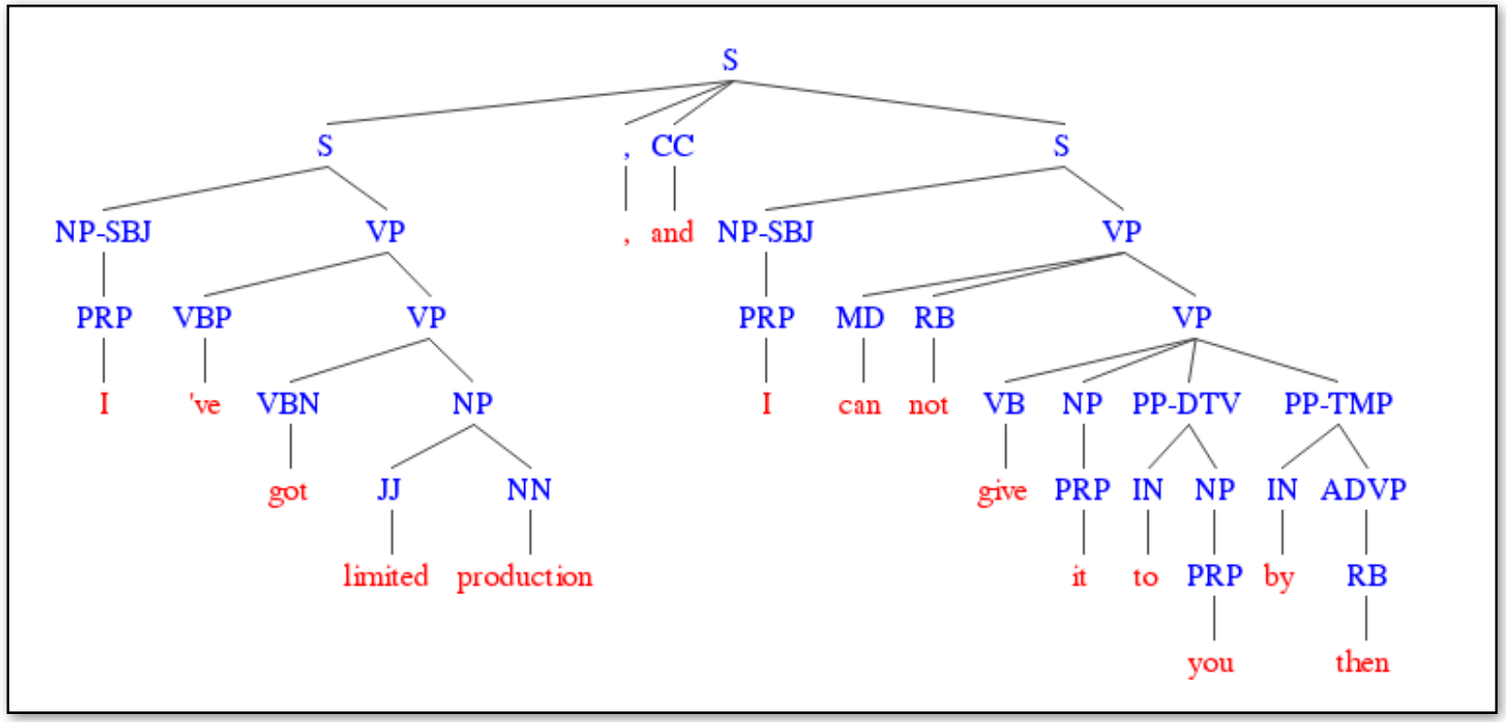
- Parsing: Kombiniere CKY-Parser für TAG (letzte Sitzung) mit Viterbi-Algorithmus.

- Training: Schätze Parameter mit MLE, z.B. so:

$$P_S(\alpha \mid A) = \frac{\alpha \text{ in } A\text{-Knoten substituiert}}{\text{irgendetwas in } A\text{-Knoten substituiert}}$$

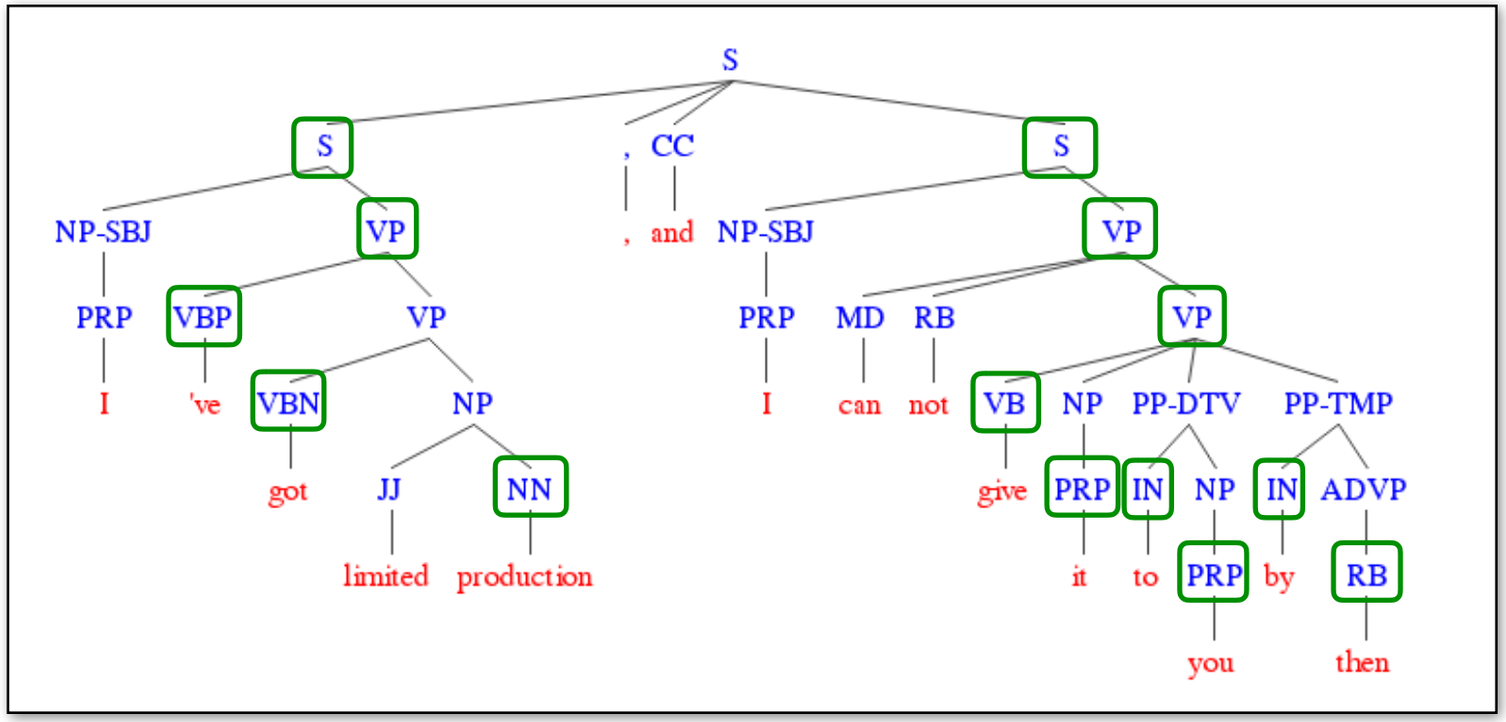
erfordert Baumbank
mit TAG-Annotationen

Bäume zerlegen



(Xia 99)

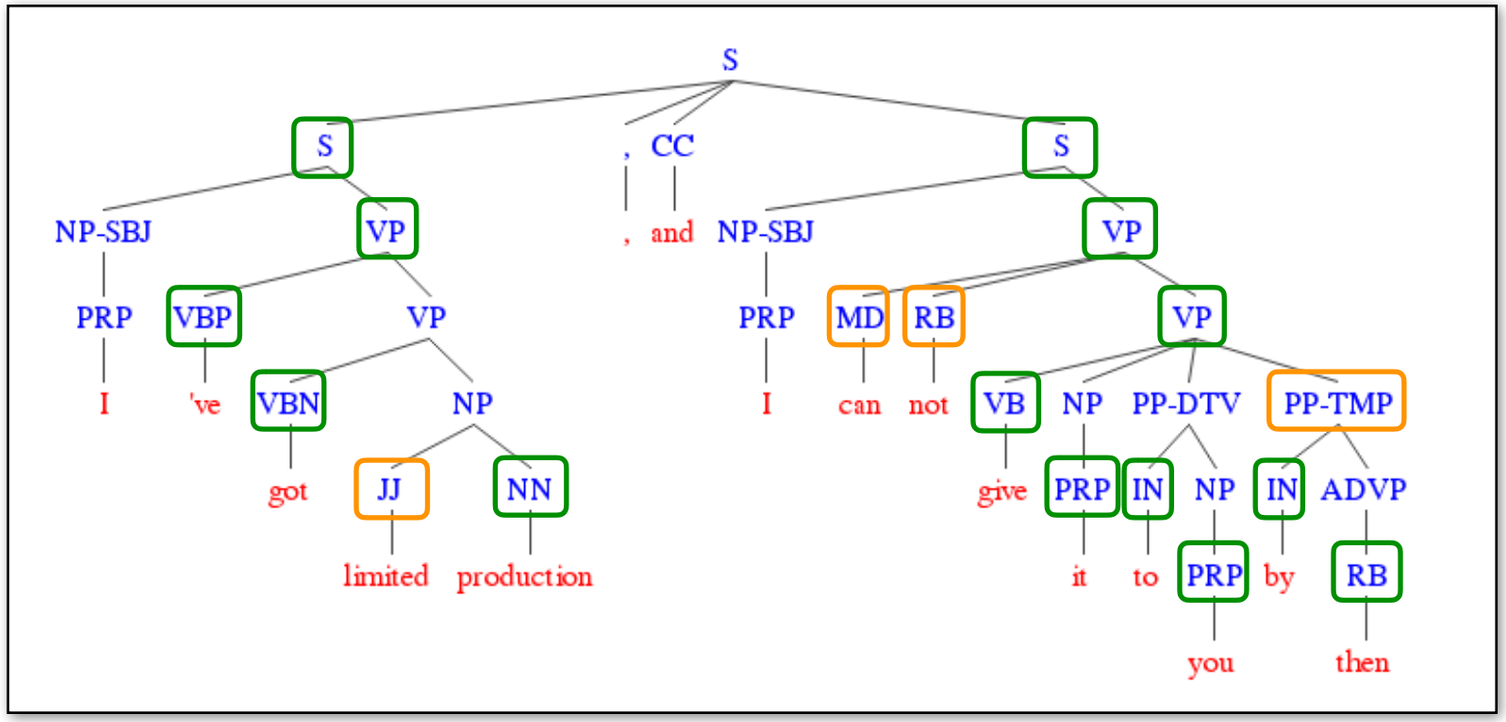
Bäume zerlegen



1. Kopfkinder identifizieren

(Xia 99)

Bäume zerlegen

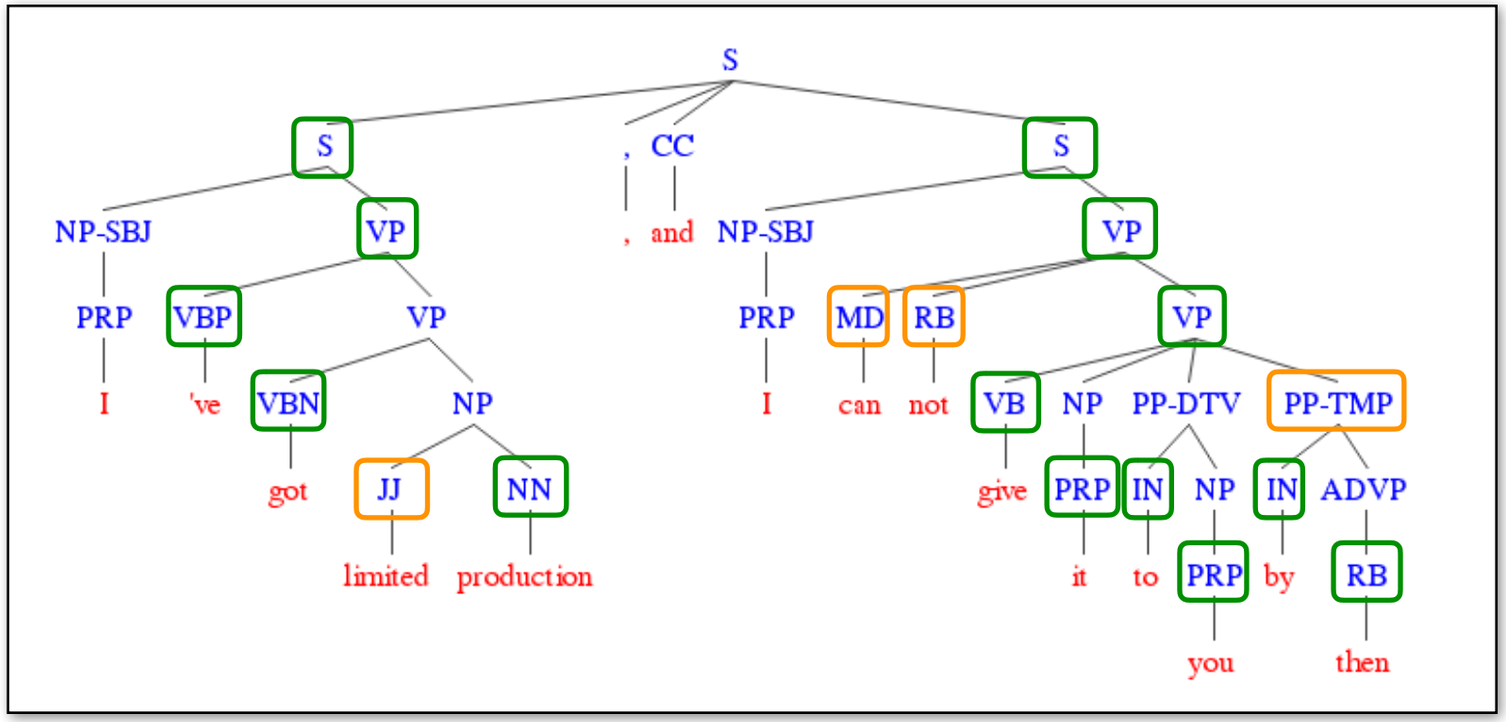


1. Kopfkinder identifizieren

2. Komplemente und Adjunkte unterscheiden

(Xia 99)

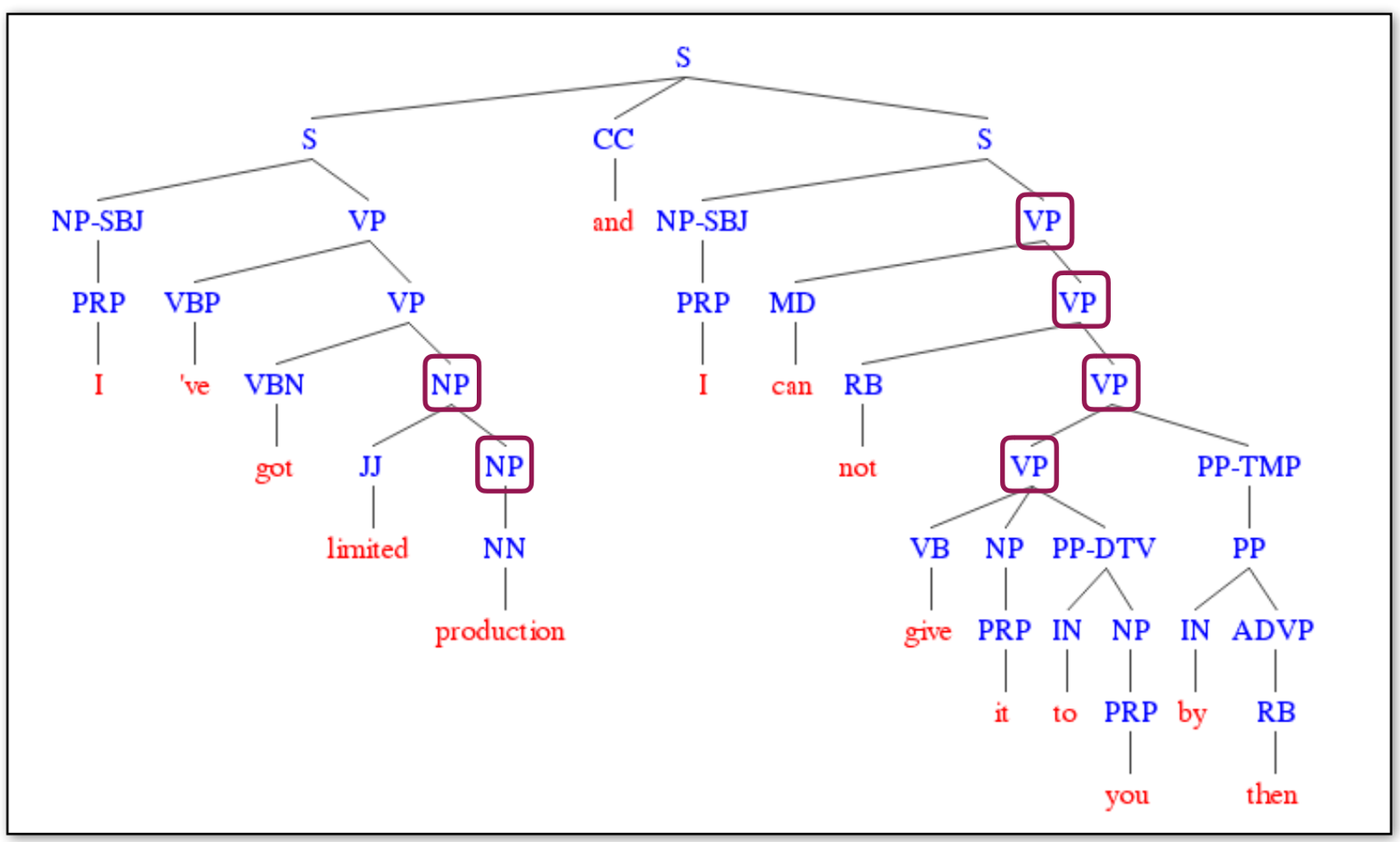
Bäume zerlegen



1. Kopfkinder identifizieren

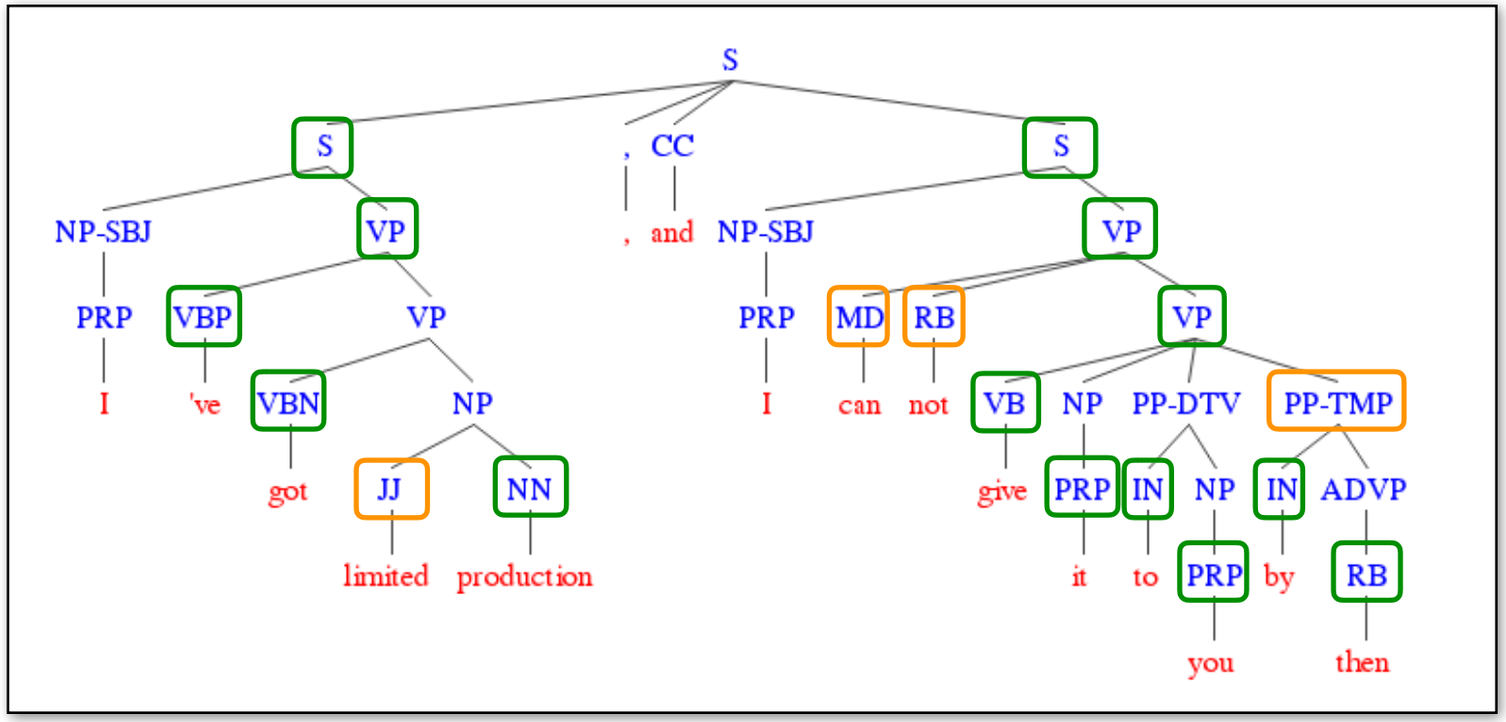
2. Komplemente und Adjunkte unterscheiden

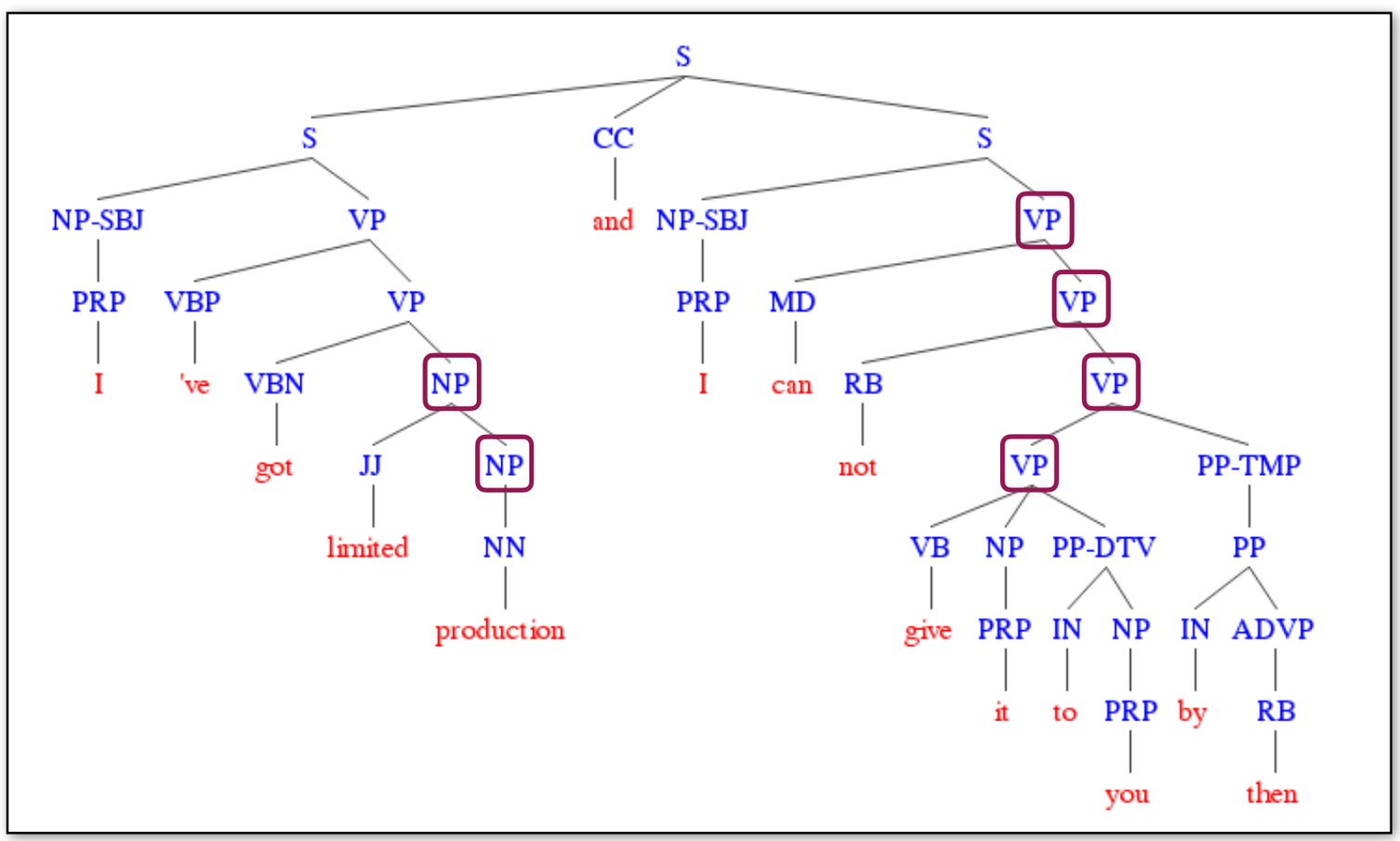
3. Wo nötig, Baum tiefer machen, damit Adjunktion stattfinden kann



(Xia 99)

Bäume zerlegen





1. Kopfkinder identifizieren

2. Komplemente und Adjunkte unterscheiden

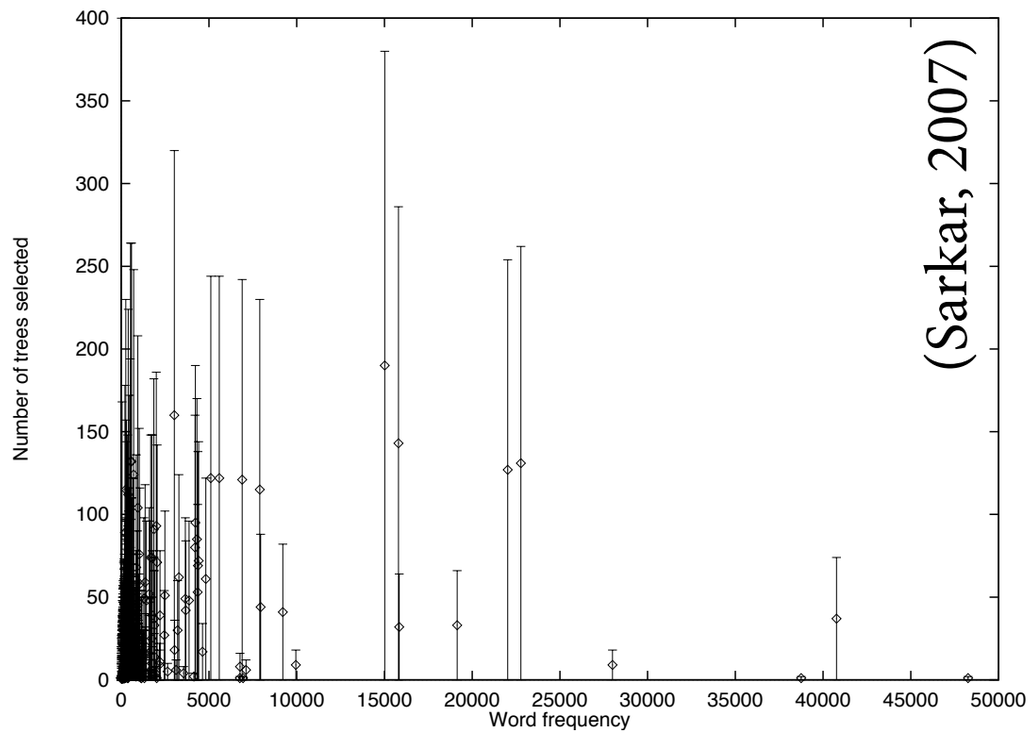
3. Wo nötig, Baum tiefer machen, damit Adjunktion stattfinden kann

4. Elementarbäume ablesen

(Xia 99)

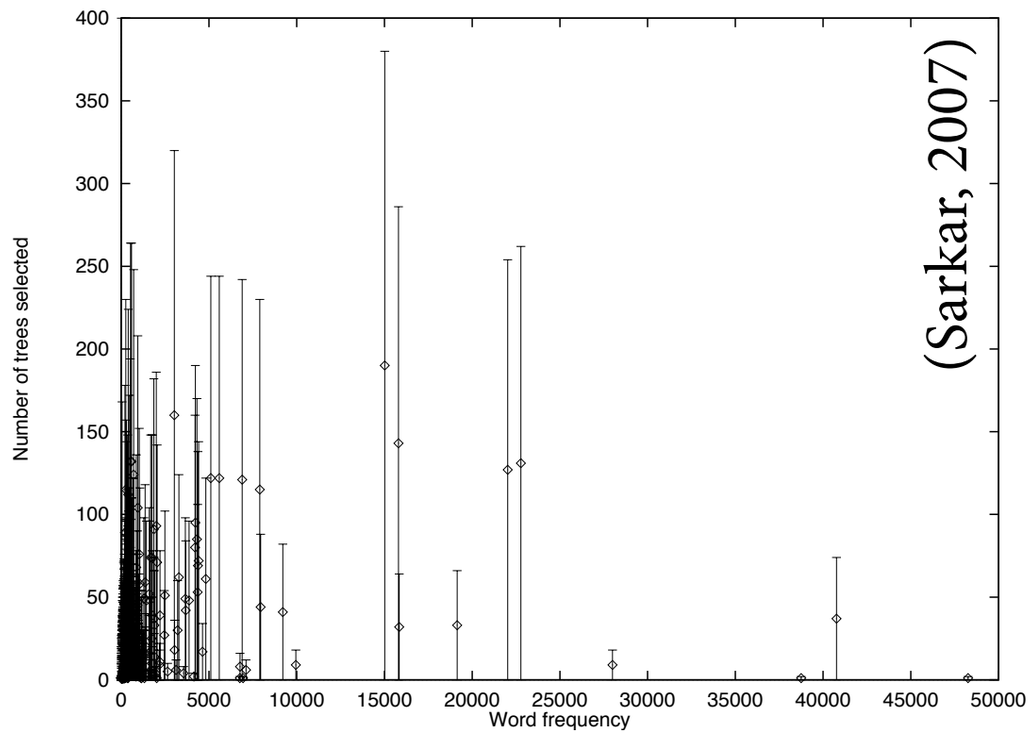
Ergebnis

- Sehr viele Elementarbäume pro Wort:



Ergebnis

- Sehr viele Elementarbäume pro Wort:



- Probleme:
  - Sparse-Data-Problem bei Parameterschätzung
  - Parsing wird ineffizient

Lexikalisierung: Segen und Fluch

- Modellierung von bilexikalischen Abhängigkeiten:
  - $P_S(\alpha_1, w_1 \mid \alpha_2, w_2, \eta_2)$ für Substitution an Knoten η_2
  - erfasst z.B. Selektionspräferenzen
  - ähnliche Modelle ermöglichen sehr akkurates Parsing für PCFGs (z.B. Collins-Parser)
- Modell hat sehr viele Parameter:
  - ca. 100.000 lexikalisierte Elementarbäume in Section 02-21
  - nicht annähernd genug Daten, um für jedes Paar von lexikalisierten Bäumen Wahrscheinlichkeiten zu lernen

Backoff-Modelle

- Zerlege Parameter in zwei Teile:

$$P_S(\alpha_1, w_1 \mid \alpha_2, w_2, \eta_2) = P_S^{(u)}(\alpha_1 \mid \alpha_2, w_2, \eta_2) \cdot P_S^{(\ell)}(w_1 \mid \alpha_1, \alpha_2, w_2, \eta_2)$$
$$\approx \underbrace{P_S^{(u)}(\alpha_1 \mid \alpha_2, \eta_2)}_{\text{unlexikalisiert}} \cdot \underbrace{P_S^{(\ell)}(w_1 \mid \alpha_1, \alpha_2, w_2, \eta_2)}_{\text{lexikalisiert}}$$

- Interpoliere P_S aus diesen beiden Teilen:
  - für häufige Wortpaare lexikalisiertes Modell hoch gewichten
  - für seltenere auf unlexikalisiertes zurückfallen

Akkuratheit

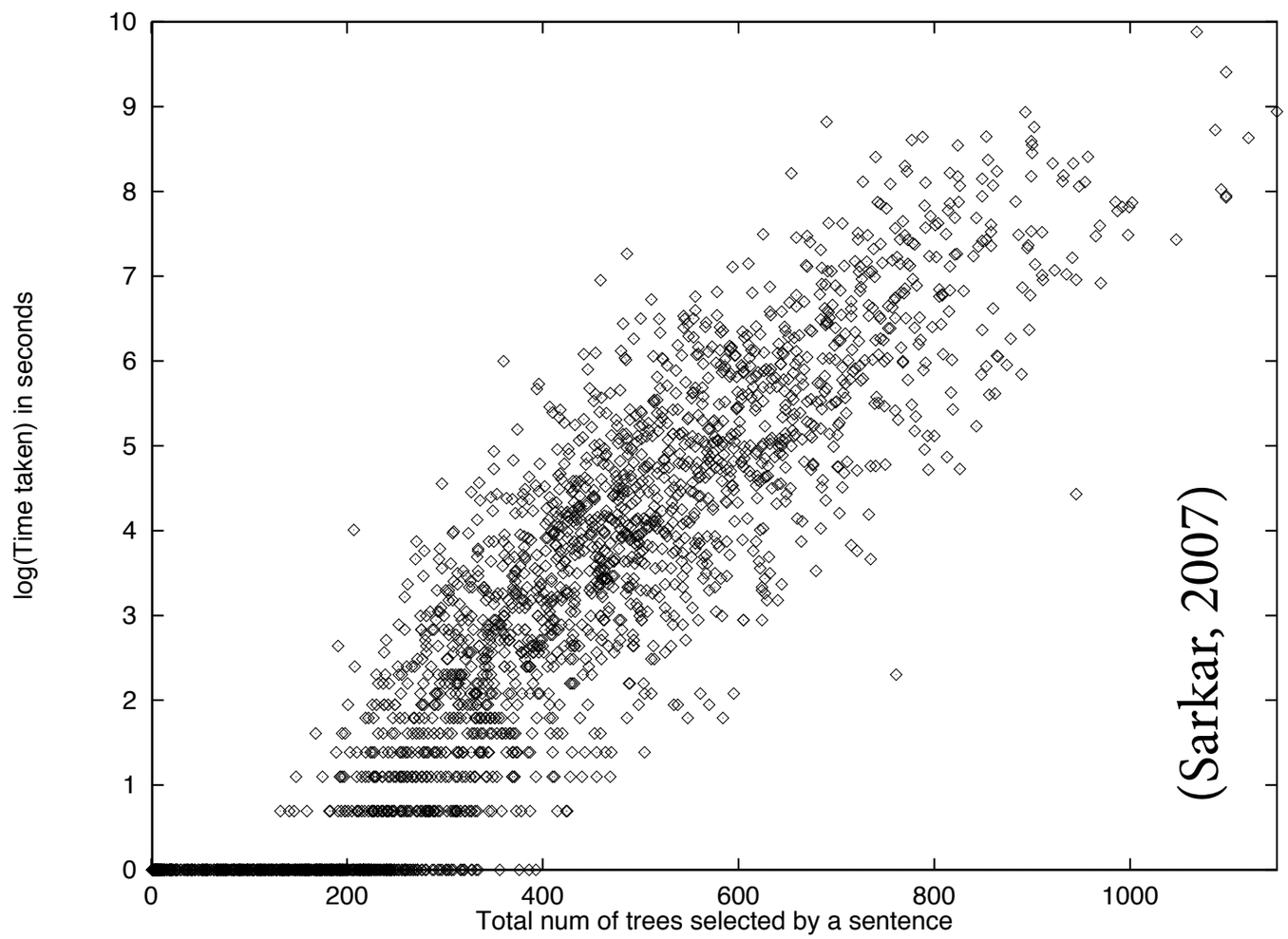
	$\leq$ 40 words					$\leq$ 100 words				
	LR	LP	CB	0 CB	$\leq$ 2 CB	LR	LP	CB	0 CB	$\leq$ 2 CB
(Magerman, 1995)	84.6	84.9	1.26	56.6	81.4	84.0	84.3	1.46	54.0	78.8
(Collins, 1996)	85.8	86.3	1.14	59.9	83.6	85.3	85.7	1.32	57.2	80.8
present model	86.9	86.6	1.09	63.2	84.3	86.2	85.8	1.29	60.4	81.8
(Collins, 1997)	88.1	88.6	0.91	66.5	86.9	87.5	88.1	1.07	63.9	84.6
(Charniak, 2000)	90.1	90.1	0.74	70.1	89.6	89.6	89.5	0.88	67.6	87.7

(Chiang, 2000)

Parsingeffizienz

- Größere TAG-Grammatiken tendieren zu hohem Maß an lexikalischer Ambiguität.

- Wesentlicher Faktor in Parsingkomplexität:



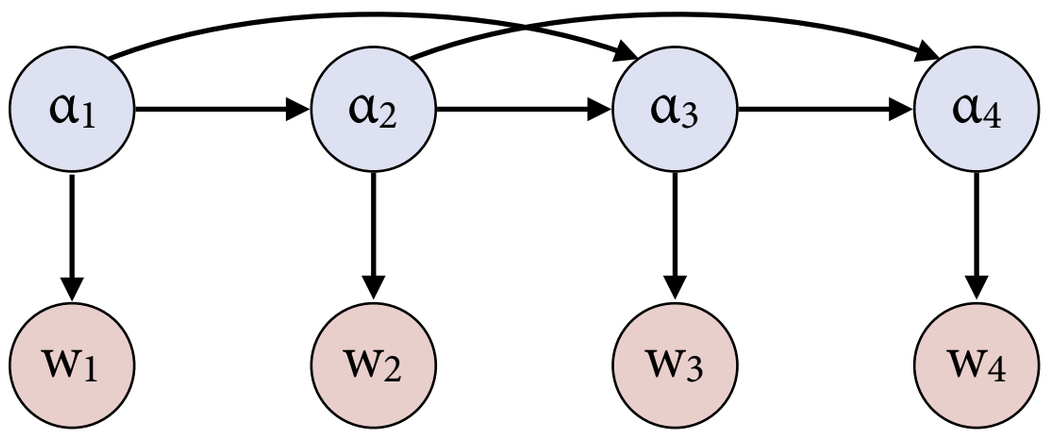
Supertagging

- Idee: Wenn man wüsste, welcher Elementarbaum der jeweils richtige für jedes Wort wäre, wäre Parsing viel einfacher.
  - Sarkar 07: 548K sec (≈ eine Woche) → 30 sec für 2250 Sätze
- Können wir richtige Elementarbäume raten, bevor wir mit Parsing anfangen?
  → *Supertagging* (Bangalore & Joshi 02)

Supertagging

- Ursprünglicher Ansatz: Supertagging ≈ POS-Tagging, nur mit mehr Klassen.

- Verwende z.B. Hidden Markov Models:



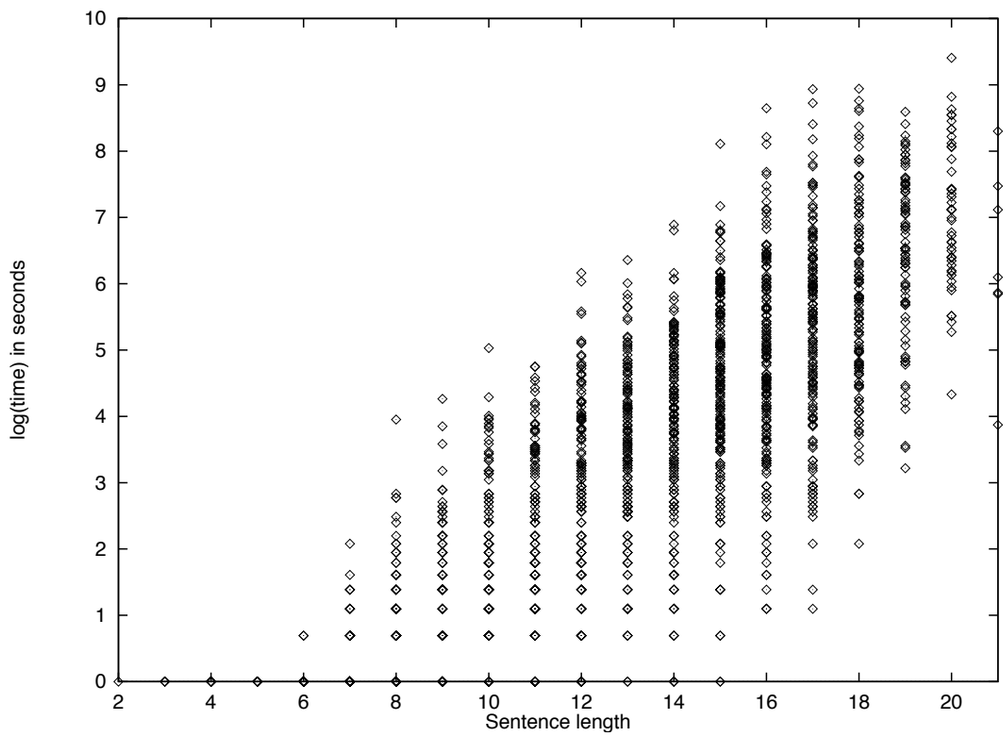
- Heute nimmt man neuronale Netze, mit sehr hoher Akkuratheit.

Tagging-Accuracy

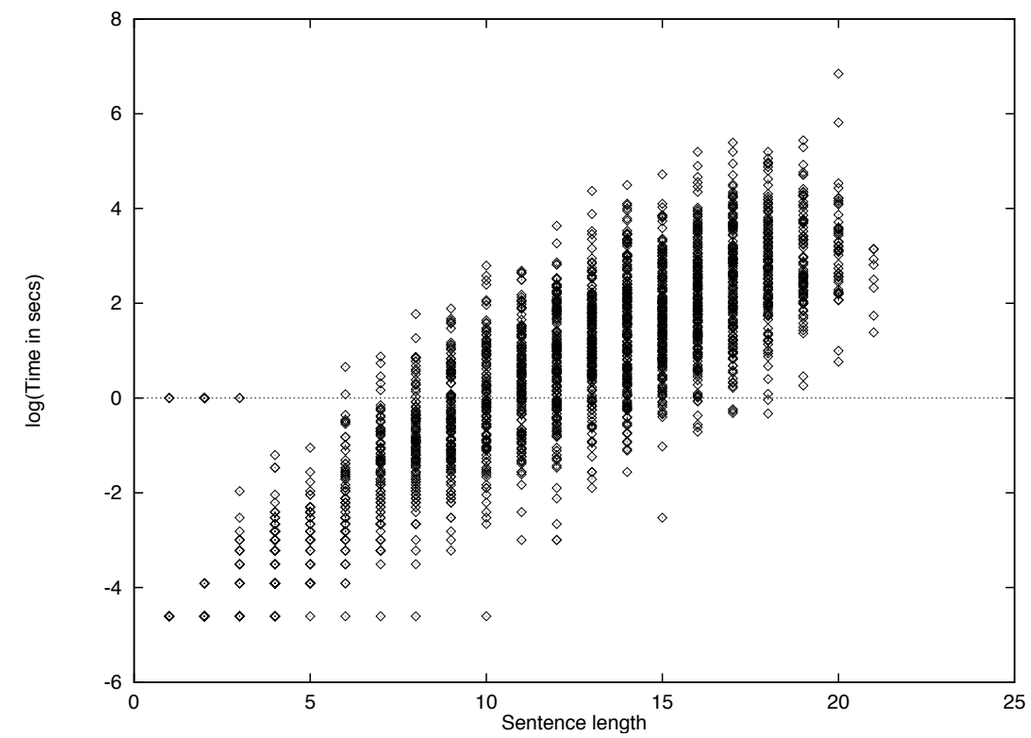
Data Set	Size of training set (words)	Training	Size of test set (words)	% Correct
XTAG Parses	8,000	Unigram (Baseline)	3,000	73.4%
		Trigram	3,000	86.0%
Converted Penn Treebank Parses	200,000	Unigram (Baseline)	47,000	75.3%
		Trigram	47,000	90.9%
	1,000,000	Unigram (Baseline)	47,000	77.2%
		Trigram	47,000	92.2%

(Bangalore & Joshi 02)

Parsingeffizienz



kein Supertagging



k-best Supertagging
mit k = 60

(Sarkar 07)

Zusammenfassung

- W.modell von PCFG überträgt sich recht einfach auf PTAG.

- Herausforderungen:
  - Parameterschätzung erfordert TAG-annotiertes Korpus
    → Korpus automatisch konvertieren
  - Lexikalisierung macht Trainingsdaten sparse
    → Smoothing
  - Hohes Maß an lexikalischer Ambiguität
    → Supertagging