

Syntax natürlicher Sprachen

10: Probabilistisches Parsing

A. Wisiorek

Centrum für Informations- und Sprachverarbeitung,
Ludwig-Maximilians-Universität München

07.01.2025

1. Erweiterungen von CFG-Grammatiken

- 1 Erweiterungen von CFG-Grammatiken
 - Grammatikentwicklung
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - PCFG: Definitionen
 - Abschätzung der Regelwahrscheinlichkeiten
- 3 Probabilistische CFG-Parsing-Algorithmen

1.1. Grammatikentwicklung

- 1 Erweiterungen von CFG-Grammatiken
 - **Grammatikentwicklung**
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - PCFG: Definitionen
 - Abschätzung der Regelwahrscheinlichkeiten
- 3 Probabilistische CFG-Parsing-Algorithmen

- **Ziel automatischer Syntaxanalyse:**
 - *Entwicklung von Grammatik mit hoher **Abdeckung/coverage***
 - *beschreibungsadäquates **Modell der syntaktischen Struktur eines sehr großen Ausschnitts** einer natürlichen Sprache*
- **Unifikationsgrammatiken:**
 - *modellieren **Agreement, Rektion und Subkategorisierung** über Merkmalconstraints*
 - *Erkennung genau der **wohlgeformten Sätze***
 - ***beschreibungsadäquate Strukturzuweisung***

- **von Experten erstellte Grammatik-Systeme**, die den Anspruch haben, einen großen Ausschnitt der Syntax einer natürlichen Sprache abzubilden:
 - Head-Driven Phrase Structure Grammar (**HPSG**):
 - ***LinGO Matrix Framework***
 - ***DELPH-IN*** ("*Deep Linguistic Processing with HPSG - INitiative*"; auch *deutsche Grammatik*)
 - ***DELPH-IN Demo***
 - Lexical Functional Grammar (**LFG**): **Pargram** Projekt
 - Lexicalized **Tree Adjoining Grammar**: **XTAG** Projekt

- **hohe Abdeckung** (viele Regeln, großes Lexikon mit ambigen Einträgen) **und Input langer (komplexer) Sätze** führen zu:
 - *hoher Aufwand beim Parsing*
 - *große Anzahl an Ableitungen/Analysen (Ambiguität)*
- z. B. durch Ambiguität im Lexikon:
 - [NP Time] [V flies] like an arrow.*
 - [V Time] [NP flies] like an arrow.*
 - [NP Time flies] [V like] an arrow.*

1.2. Disambiguierung durch statistische Modelle

- 1 Erweiterungen von CFG-Grammatiken
 - Grammatikentwicklung
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - PCFG: Definitionen
 - Abschätzung der Regelwahrscheinlichkeiten
- 3 Probabilistische CFG-Parsing-Algorithmen

- **Erweiterung von CFGs um probabilistische Parameter**
 - **gewichtete Grammatik**: Produktionsregeln erhalten Bewertung
 - erlaubt **Ranking der Ableitungen** eines strukturell ambigen Satzes aufgrund von **Trainingsdaten aus Korpus**
- **Disambiguierung über empirisches Modell**
 - **statt Disambiguierung über explizite semantische Informationen** im Anschluss an syntaktisches Parsing durch semantisches Parsing:
 - Auswahl Ableitung aufgrund von **statistischen Informationen aus Korpusdaten zu Kollokationen von Wörtern und syntaktischen Kategorien**
 - **beste syntaktische Analyse eines Satzes = die im Sprachgebrauch häufigste**
 - **graduelle Modellierung von Grammatikalität**

- **Probabilistische CFG (= PCFG)** erlaubt in Kombination mit **dynamischem Parsing** das effiziente Auffinden der besten (= wahrscheinlichsten) Ableitung
- **ohne Gewichtung**: dynamische Programmierung (CYK, Earley) kann zwar Parsing-Aufwand bei großem Suchraum (großer Grammatik) reduzieren, aber **keine Auswahl** treffen aus den gefundenen Ableitungen
- **statistische Informationen** können auch **im Parsing von Unifikationsgrammatiken** (wie LFG, HPSG) zur Disambiguierung verwendet werden
- **nächste Sitzung**: statt bloßer Erweiterung einer gegebenen CFG um statistische Informationen aus Treebanks: **Extraktion von Grammatiken aus Treebanks**
→ *in solchen induzierten Grammatiken können auch lexikalische Informationen und Informationen zum strukturellen Kontext berücksichtigt werden, die der weiteren Disambiguierung dienen*

- Erweiterung von CFG-Grammatiken durch statistische Parameter zur **Disambiguierung**
- **strukturelle Disambiguierung** durch *parse selection* (Herausfiltern der wahrscheinlichsten Ableitung)
- **Wahrscheinlichkeiten der Regeln** müssen anhand von Korpusdaten gelernt werden (Parameter-Abschätzung)
- Algorithmen dynamischer Programmierung (**Viterbi-Algorithmus**) zur **effizienten Auffindung der wahrscheinlichsten Ableitung**

2. PCFGs: Probabilistische kontextfreie Grammatiken

- 1 Erweiterungen von CFG-Grammatiken
 - Grammatikentwicklung
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - PCFG: Definitionen
 - Abschätzung der Regelwahrscheinlichkeiten
- 3 Probabilistische CFG-Parsing-Algorithmen

- **MS:** Manning, Christopher D. & Schütze, Hinrich (1999): *Foundations of Statistical Natural Language Processing*.
- NLTK-Teilkapitel 8.6 ('*Grammar Development*') und 8.5.2 ('*Scaling up*'): <http://www.nltk.org/book/ch08.html>
- Teilkapitel 2.12 ('*Grammar Induction*') des Zusatzkapitels zu Kapitel 8: <http://www.nltk.org/book/ch08-extras.html>
- Die Teilkapitel 2.9-2.11 des Zusatzkapitels zu Kapitel 8 behandeln probabilistische Chart Parsing-Algorithmen: <http://www.nltk.org/book/ch08-extras.html>

2.1. PCFG: Definitionen

- 1 Erweiterungen von CFG-Grammatiken
 - Grammatikentwicklung
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - **PCFG: Definitionen**
 - Abschätzung der Regelwahrscheinlichkeiten
- 3 Probabilistische CFG-Parsing-Algorithmen

- **PCFG** = kontextfreie Grammatik, deren Regeln mit **Wahrscheinlichkeiten** gewichtet sind:
 $S \rightarrow NP VP \ 1.0$
 $VP \rightarrow VP NP \ 0.4$
 $VP \rightarrow VP NP PP \ 0.6$
 $NP \rightarrow NP PP \ 0.2$
 $NP \rightarrow N \ 0.8$
- **Wahrscheinlichkeiten** aller Regeln für die **Expansion** eines bestimmten Nonterminals addieren sich zu 1

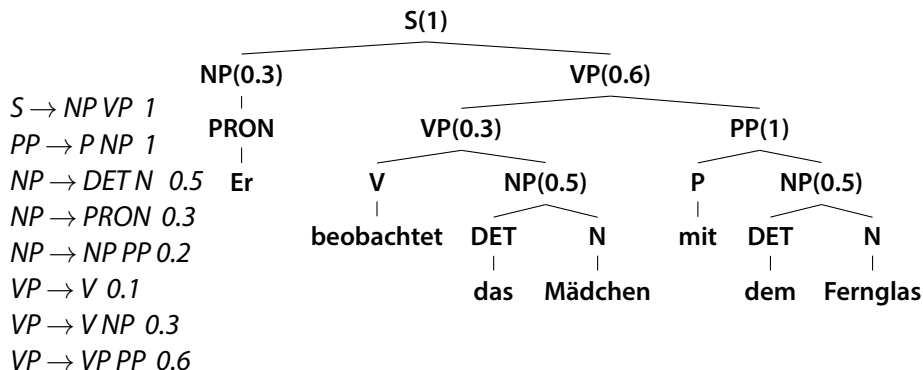
- **Ableitung/Baum** ist Menge an Regeln/Expansionen
→ *Teilbäume mit Tiefe 1*
- **Wahrscheinlichkeit einer Ableitung T** (*Tree*) als Multiplikation der Wahrscheinlichkeiten ihrer Regeln:

$$P(T) = \prod_{i=1}^n P(R_i) = \prod_{i=1}^n P(RHS_i | LHS_i)$$

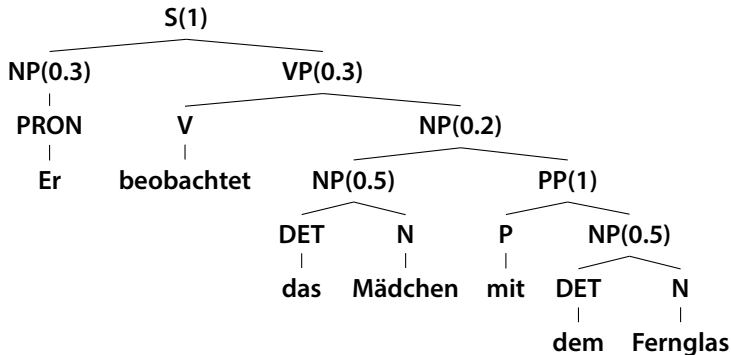
→ Iteration über die n Knoten im Baum: **Produkt der Wahrscheinlichkeit der Expansion des LHS-Knotens von R_i zu RHS-Symbolfolge von R_i**
→ **Annahme Unabhängigkeit der Regel-Auswahl**

- zur **Disambiguierung** muss die **wahrscheinlichste Ableitung T^*** zu einem Satz S gefunden werden:
 - $T^* = \arg \max P(T|S) = \arg \max \frac{P(T, S)}{P(S)} = \arg \max \frac{P(T)}{P(S)}$
($P(T, S) = P(T)P(S|T)$, $P(S|T) = 1$; jeder Baum leitet genau einen Satz ab)
 - $T^* = \arg \max P(T)$
(da $P(S)$ konstant fuer ein S , also irrelevant fuer Auswahl Ableitung zu gegebenem Satz)
- **Satzwahrscheinlichkeit:** Summe der Wahrscheinlichkeiten aller möglichen Ableitungen eines Satzes:
$$P(S) = \sum P(T, S) = \sum P(T)$$

Beispiel-PCFG PP-Attachment-Ambiguität



$P(T_1) = 1 * 0.3 * \mathbf{0.6} * \mathbf{0.3} * 0.5 * 1 * 0.5 = \mathbf{0.0135}$ (ohne lexikalische Gewichte, diese sind konstant für einen Satz)



$$P(T_2) = 1 * 0.3 * \mathbf{0.3} * \mathbf{0.2} * 0.5 * 1 * 0.5 = \mathbf{0.0045}$$

⇒ **Auswahl adverbialer Lesart** : $P(T_1) > P(T_2)$

Grund: $P(VP, PP|VP) > P(NP, PP|NP)$

2.2. Abschätzung der Regelwahrscheinlichkeiten

- 1 Erweiterungen von CFG-Grammatiken
 - Grammatikentwicklung
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - PCFG: Definitionen
 - **Abschätzung der Regelwahrscheinlichkeiten**
- 3 Probabilistische CFG-Parsing-Algorithmen

- ***supervised*** = Bestimmung der relativen Häufigkeiten der Expansionen eines Nichtterminals in geparstem (syntaktisch annotiertem) Korpus (**Maximum Likelihood Estimation**)
- ***unsupervised*** = wiederholtes Parsen von Korpus mit der gegebenen kontextfreien Grammatik und sukzessive Verbesserung eines statistischen Modells (**Inside-Outside-Algorithmus**)

1. Maximum Likelihood Estimation (MLE)

- Abschätzung der Regelwahrscheinlichkeit als **relative Häufigkeit der Expansion des LHS-Nonterminals zu RHS-Symbolfolge in Treebank** (syntaktisch annotiertem Korpus)
- $$P(\alpha \rightarrow \beta | \alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{count}(\alpha \rightarrow \gamma)} = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)}$$
- **Expansionswahrscheinlichkeit:**
 $P(RHS|LHS) = P(Expansion|Nonterminal)$
→ Idee: gute probabilistische Grammatik **maximiert die Wahrscheinlichkeit der Trainingsdaten**

- Wahrscheinlichkeit für Expansion $VP \rightarrow V NP PP$:

$$P(V, NP, PP | VP) = \frac{\text{count}(VP \rightarrow V NP PP)}{\text{count}(VP \rightarrow \setminus^*)}$$

- $\text{count}(VP \rightarrow V NP PP) = 10$
 $\text{count}(VP \rightarrow V NP) = 50$
 $\text{count}(VP \rightarrow V) = 40$
 $\Rightarrow MLE(VP \rightarrow V NP PP | VP) = 1/10$

2. Inside-Outside-Algorithmus

- Abschätzung der Regelwahrscheinlichkeiten auch **ohne syntaktisch annotiertes Trainingskorpus**, d. h. *unsupervised* möglich mit **Inside-Outside-Algorithmus**
- Variante von **EM-Algorithmus** (*Expectation-Maximation*)
 - *iterativen Abschätzung der Regelwahrscheinlichkeiten (als Parameter des statistischen Modells)*
 - *Übertragung des Forward-Backward-Algorithmus (zur Abschätzung von Parametern bei HMMs) auf PCFGs*

3. Probabilistische CFG-Parsing-Algorithmen

- 1 Erweiterungen von CFG-Grammatiken
 - Grammatikentwicklung
 - Disambiguierung durch statistische Modelle
- 2 PCFGs: Probabilistische kontextfreie Grammatiken
 - PCFG: Definitionen
 - Abschätzung der Regelwahrscheinlichkeiten
- 3 Probabilistische CFG-Parsing-Algorithmen

- **Suche der wahrscheinlichsten Ableitung:**
 $\arg \max P(T|S) = \arg \max P(T)$
- **Suche aller Ableitungen** und Berechnung ihrer Wahrscheinlichkeiten wird **bei großen Grammatiken sehr aufwendig**
- besser: **probabilistische Varianten von Chart-Parsing-Algorithmen** wie CYK- oder Earley-Algorithmus
- Verwendung statistischer Informationen in **dynamischer Programmierung** zum effizienten Auffinden der wahrscheinlichsten (Teil)bäume

- **PCFG-Version des Viterbi-Algorithmus** (analog zu HMM): **Finden der wahrscheinlichsten verborgenen Zustandsfolge** (*Ableitung T*), die die beobachtete Sequenz emittiert (*Satz S*)
 - *Bestimmung des **wahrscheinlichsten Baumes** durch Zurückgreifen auf **berechnete Teilbäume***
 - *die **Wahrscheinlichkeit größerer Teilbäume** ergibt sich **aus den Wahrscheinlichkeiten der kleineren**, da aufgrund der Kontextfreiheit die Wahrscheinlichkeit eines Teilbaums unabhängig von seiner Position ist*
 - ***nur die Teilbäume mit höchster Wahrscheinlichkeit** werden behalten und zur Berechnung verwendet*
- **Performanz-Optimierung** des Parsings durch Verwendung statistischer Informationen
 - *statt **allen möglichen** **nur die wahrscheinlichsten Teilergebnisse** verwenden*

- `nltk.ViterbiParser`
 - **Bottom-up-PCFG-Parser**
 - *berechnet inkrementell (beginnend mit Spanne Länge 1) die wahrscheinlichsten (Teil)bäume durch Ausfüllen einer 'Most Likely Constituents Table'*
- für **gegebene Spanne und Knoten-Wert** (LHS einer Regel):
 - *Suche nach Folgen von Tabellen-Einträgen, die gemeinsam die Spanne abdecken*
 - *Überprüfung, ob Tabellen-Einträge die RHS-Werte der Regel als Knotenwerte haben (LHS der Tabellen-Einträge)*

Span	Node	Tree
[0 : 1]	NP	(NP I)
[6 : 7]	NP	(NN telescope)
[5 : 7]	NP	(NP the telescope)
[4 : 7]	PP	(PP with (NP the telescope))
[0 : 4]	S	(S (NP I) (VP saw (NP the man)))
[0 : 7]	S	(S (NP I) (VP saw (NP (NP the man) (PP with (NP the telescope)))))

Abbildung: Most Likely Constituents Table (Ausschnitt)

- Tabelle enthält **nur die wahrscheinlichste Ableitung für eine Spanne und Knoten-Wert**: z. B. wird nur die *NP-attachment*-Variante für Spanne [1 : 7] und Knoten-Wert VP aufgenommen:

[1 : 7]	VP	(VP saw (NP (NP the man) (PP with (NP the telescope)))))
[1 : 7]	VP	(VP saw (NP (NP the man)) (PP with (NP the telescope)))

Listing 1: NLTK: PCFG-Parsing mit Viterbi-Parser

```
1 #http://www.nltk.org/\_modules/nltk/parse/viterbi.html
2 #http://www.nltk.org/book/ch08-extras.html
3
4 grammar = nltk.PCFG.fromstring('''
5     NP  -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
6     NNS -> "cats" [0.1] | "dogs" [0.2] | "mice" [0.3] |
7           NNS CC NNS [0.4]
8     JJ  -> "big" [0.4] | "small" [0.6]
9     CC  -> "and" [0.9] | "or" [0.1]
10    ''')
11 sent = 'big cats and dogs'.split()
12
13 viterbi_parser = nltk.ViterbiParser(grammar)
14 for tree in viterbi_parser.parse(sent):
15     print(tree)
16 #\(NP \(JJ big\) \(NNS \(NNS cats\) \(CC and\) \(NNS dogs\)\)\) \(p
17 =0.000864\)
18 viterbi_parser.trace(3)
```

```

19 for tree in viterbi_parser.parse(sent):
20     print(tree)
21
22
23 #Inserting tokens into the most likely constituents
    table...
24 #   Insert: |=...| big
25 #   Insert: |.=..| cats
26 #   Insert: |..=.| and
27 #   Insert: |...=| dogs
28 #Finding the most likely constituents spanning 1 text
    elements...
29 #   Insert: |=...| JJ -> 'big' [0.4]
    0.4000000000
30 #   Insert: |.=..| NNS -> 'cats' [0.1]
    0.1000000000
31 #   Insert: |.=..| NP -> NNS [0.5]
    0.0500000000
32 #   Insert: |..=.| CC -> 'and' [0.9]
    0.9000000000
33 #   Insert: |...=| NNS -> 'dogs' [0.2]

```

```

34 #      Insert: |...=| NP -> NNS [0.5]
      0.2000000000
      0.1000000000
35
36
37 #Finding the most likely constituents spanning 2 text
      elements...
38 #      Insert: |=...| NP -> JJ NNS [0.3]
      0.0120000000
39 #Finding the most likely constituents spanning 3 text
      elements...
40 #      Insert: |.===| NP -> NP CC NP [0.2]
      0.0009000000
41 #      Insert: |.===| NNS -> NNS CC NNS [0.4]
      0.0072000000
42 #      Insert: |.===| NP -> NNS [0.5]
      0.0036000000
43 #      Discard: |.===| NP -> NP CC NP [0.2]
      0.0009000000
44 #      Discard: |.===| NP -> NP CC NP [0.2]
      0.0009000000

```

```
45 #Finding the most likely constituents spanning 4 text
    elements...
46 #   Insert: |====| NP -> JJ NNS [0.3]
    0.0008640000
47 #   Discard: |====| NP -> NP CC NP [0.2]
    0.0002160000
48 #   Discard: |====| NP -> NP CC NP [0.2]
    0.0002160000
49 #(NP (JJ big) (NNS (NNS cats) (CC and) (NNS dogs))) (p
    =0.000864)
```


- `nltk.parse.pchart` = Klasse von Bottom-up-PCFG-Chart-Parsern
- Chart-Parsing mit zusätzlicher Datenstruktur *edge queue*, deren **Sortierung** die Reihenfolge der Abarbeitung der Zustände festlegt
→ *edge* in Chart-Parsing nach Kay = **Zustand** bei Earley/CYK
- im Gegensatz zu Viterbi-Parser wird **nicht nur die wahrscheinlichste Ableitung** gefunden, sondern die **n-besten Ableitungen**
- Verwendung von **statistischen Daten zur Sortierung**

- **Lowest Cost First** = `nltk.InsideChartParser`
 - **Sortierung nach Wahrscheinlichkeit** der Ableitungen
 - findet immer die **optimale** Lösung (wahrscheinlichste Ableitung)
 - **Problem: kürzere Teilergebnisse haben üblicherweise eine höhere Wahrscheinlichkeit** ($P = \text{Produkt der Regelwahrscheinlichkeiten}$) und werden so zuerst abgearbeitet; **vollständige Ableitung wird erst spät produziert**
- **Best-First Search** = `nltk.LongestChartParser`
 - **Sortierung nach Länge** (für vollständige Ableitung: längste Spanne gesucht)
 - i. A. **schneller** als *Lowest Cost First*
 - **garantiert nicht, dass optimale Ableitung gefunden wird**

- **Beam Search (Pruning)** = `nltk.InsideChartParser(grammar, beam_size=20)`
 - *Lowest-Cost-First, aber **nur die n-besten partiellen Ergebnisse behalten** (= Pruning)*
 - **schneller** als *Lowest-Cost-First* ohne Pruning
 - *garantiert nicht, dass optimale Ableitung gefunden wird*
 - *garantiert nicht, dass überhaupt eine Ableitung gefunden wird (wenn notwendige edges fehlen)*

Listing 2: NLTK: PCFG-Parsing mit ChartParser

```
1 #http://www.nltk.org/\_modules/nltk/parse/pchart.html
2 #http://www.nltk.org/book/ch08-extras.html
3
4 inside_parser = nltk.InsideChartParser(grammar)
5 longest_parser = nltk.LongestChartParser(grammar)
6 beam_parser = nltk.InsideChartParser(grammar, beam_size
    =20)
7
8 for tree in inside_parser.parse(sent):
9     print(tree)
10 #\(NP \(JJ big\) \(NNS \(NNS cats\) \(CC and\) \(NNS dogs\)\)\) \(p
    =0.000864\)
11 #\(NP \(NP \(JJ big\) \(NNS cats\)\) \(CC and\) \(NP \(NNS dogs\)\)\)
    \(p=0.000216\)
```