Syntax natürlicher Sprachen

Tutorium 11:

Grammatikinduktion und Annotationen

Shuyan Liu

17.01.2025

Einige Beispiele kommt aus der Vorlesungsfolien, Aufgaben sowie Übungen.

Die Hauptteile der Slides dieser Woche stammen von Sarah Anna Uffelmann

aus Wintersemester 2023/24 und wurden bearbeitet. Verwendung mit Dank.

Grammatikinduktion

- Aus annotierte Kopura (Sg. Korpus) können Grammatikregeln extrahiert werden.
- Wahrscheinlichkeiten der Grammatikregeln basiert sich auf ihren relativen Häufigkeiten.
  - Für Generierung korrekter Parsebäume wichtig.
  - Formel: $P(\alpha \to \beta|\alpha) = \frac{count(\alpha \to \beta)}{\sum_\gamma count(\alpha \to \gamma)} = \frac{count(\alpha \to \beta)}{count(\alpha)}$

    P(α -> β | α): α -> β gegeben α. d.h. α (linke Seite einer Regel) ist schon festgelegt
    $\sum_\gamma count(\alpha \to \gamma)$: Anzahl von α als linke Seite und eine beliebe **γ** als rechte Seite

  - Beispiel siehe nächste Folie

Berechnung von Regelwahrscheinlichkeiten

Regelwahrscheinlichkeiten werden aus Regelhäufigkeiten berechnet.

Beispiel: In einer Treebank haben wir folgende Regelhäufigkeiten gezählt:

Regel	Häufigkeit
NP -> Det N	200
NP -> Pron	175
NP -> NP PP	125

Wie berechnen wir die Regelwahrscheinlichkeiten für die Regel NP -> Det N?

Formel: $$P(\alpha \to \beta | \alpha) = \frac{count(\alpha \to \beta)}{\sum_\gamma count(\alpha \to \gamma)} = \frac{count(\alpha \to \beta)}{count(\alpha)}$$

P (Det N | NP) = 200 / (200 + 175 + 125) = 200 / 500 = 0,4
P (Pron | NP) = 175 / 500 = 0,35
P (NP PP | NP) = 125 / 500 = 0,25

Berechnung von Regelwahrscheinlichkeiten

Regelwahrscheinlichkeiten werden aus Regelhäufigkeiten berechnet.

Beispiel: In einer Treebank haben wir folgende Regelhäufigkeiten gezählt:

Regel	Häufigkeit	Wahrscheinlichkeit
NP -> Det N	200	0,4
NP -> Pron	175	0,35
NP -> NP PP	125	0,25

Formel: $$P(\alpha \rightarrow \beta | \alpha) = \frac{count(\alpha \rightarrow \beta)}{\sum_{\gamma} count(\alpha \rightarrow \gamma)} = \frac{count(\alpha \rightarrow \beta)}{count(\alpha)}$$

P (Det N | NP) = 200 / (200 + 175 + 125) = 200 / 500 = 0,4
P (Pron | NP) = 175 / 500 = 0,35
P (NP PP | NP) = 125 / 500 = 0,25

Chomsky-Normalform (CNF)

Eine kontextfreie Grammatik (CFG) befindet sich in **Chomsky-Normalform (CNF)**, wenn alle Produktionsregeln die folgende Form haben:

1. **A → BC**
   1. Ein Nichtterminal *A*A wird in zwei Nichtterminale B und C deriviert.
   2. B und C sind ebenfalls Nichtterminale.
2. **A → a**
   1. Ein Nichtterminal A produziert genau ein Terminal a.
3. **S → ε** (optional, nur für leere Sprache)
   1. Startsymbol S produziert den leeren String ε, falls ε in der Sprache enthalten ist.

Chomsky-Normalform (CNF)

Anwendung:

- CYK-Algorithmus (Effizientes Parsing von CFGs)
- Formale Sprachen (Eindeutige Struktur)

Konversion nach CNF

Wie bringen wir diese Regel in Chomsky-Normalform?

A -> b C D e

1. Regeln für die Terminale einführen

B -> b
E -> e
A -> B C D E

2. Regeln verkürzen durch das Einführen von Zwischenebenen

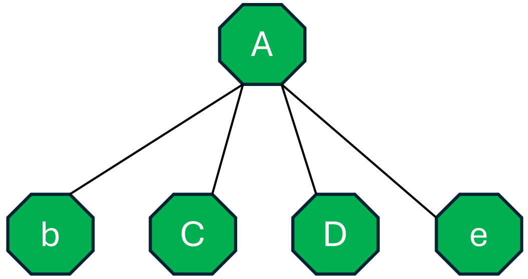
A –> B X1
X1 - > C X2
X2 -> D E

(Diese drei Regeln in Kombination leisten dasselbe wie die Regel A -> B C D E)

Konversion nach CNF

Wie bringen wir diese Regel in Chomsky-Normalform?

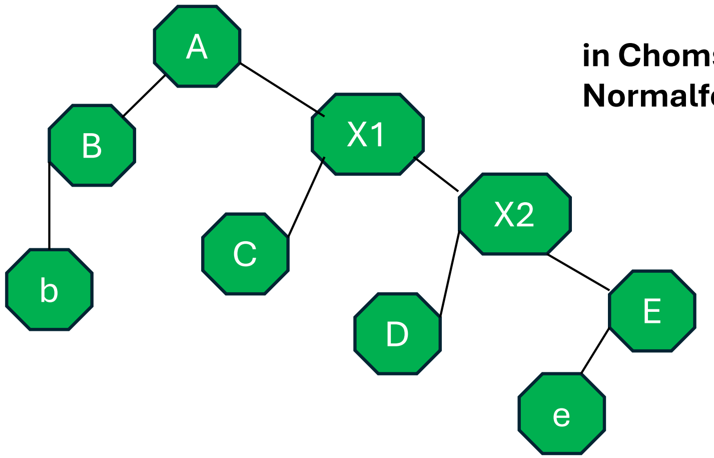
A -> b C D e



flache Struktur

Lösung:

B -> b
E -> e
A –> B X1
X1 - > C X2
X2 -> D E



in Chomsky-Normalform

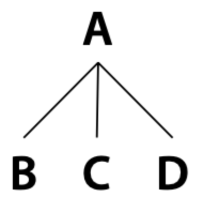
CNF mit NLTK

In NLTK können wir eine Grammatik mit der Methode **chomsky_normal_form( )** in die Chomsky-Normalform umformen.

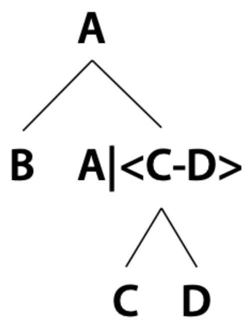
```python
from nltk import Tree
treebank_string = """(S (NP-SBJ (NP (QP (IN at) (JJS least) (CD nine) (NNS tenths)) ) (PP (IN of) (NP (DT the) (NNS students) ))) (VP (VBD passed)))"""
t = nltk.Tree.fromstring(treebank_string)
```

```python
t3 = copy.deepcopy(t)
t3.chomsky_normal_form(factor='left')
```

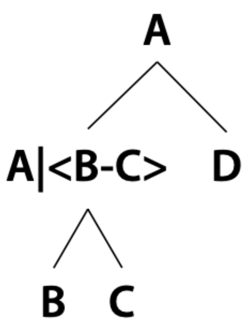
Original:



Right-Factored:



Left-Factored:



factor (*str = [left|right]*) –
Right or left factoring method (default = “right”)

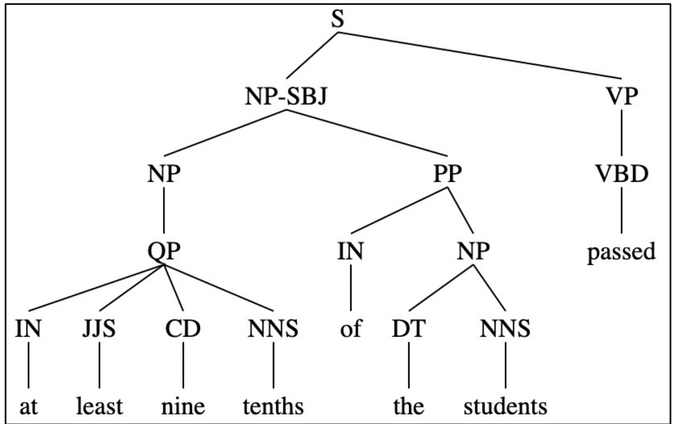
Weitere Infos zur Methode:
https://www.nltk.org/_modules/nltk/tree.html

CNF mit NLTK

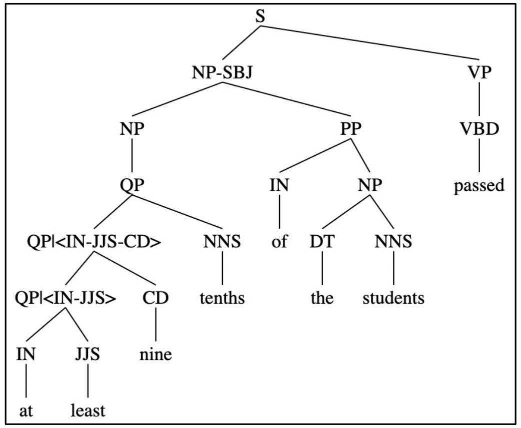
In NLTK können wir eine Grammatik mit der Methode **chomsky_normal_form( )** in die Chomsky-Normalform umformen.

```
from nltk import Tree
treebank_string = """(S (NP-SBJ (NP (QP (IN at) (JJS least) (CD nine) (NNS tenths)) ) (PP (IN of) (NP (DT the) (NNS students) ))) (VP (VBD passed)))"""
t = nltk.Tree.fromstring(treebank_string)
```

```
t3 = copy.deepcopy(t)
t3.chomsky_normal_form(factor='left')
```



ursprünglicher Parsebaum



in Chomsky-Normalform

factor (*str = [left|right]*) –
Right or left factoring method (default = “right”)

Weitere Infos zur Methode: https://www.nltk.org/_modules/nltk/tree.html

Unabhängigkeitsannahmen von PCFGs

1. Annahme: Unabhängigkeit von lexikalischem Material

- **Erklärung**:
  - In einfachen Modellen wird angenommen, dass die **Wahrscheinlichkeiten von Teilbäumen unabhängig von den Terminalen** (den Wörtern) im Satz sind.
  - Beispiel: In einer CFG betrachten die Produktionsregeln oft nur die syntaktischen Strukturen (z. B. NP→Det+N), unabhängig davon, **welche Wörter tatsächlich als Det oder N auftreten** (z. B. "der Hund" vs. "die Katze").
- **Problem**:
  - Diese Annahme ist unrealistisch, weil das lexikalische Material (Wörter) oft Einfluss auf die Syntax hat (z. B. regiert ein bestimmtes Verb einen bestimmten Kasus).
- Model:
  - **Lexikalisierte PCFGs** ⇒ Auflösung lexikalischer Ambiguität

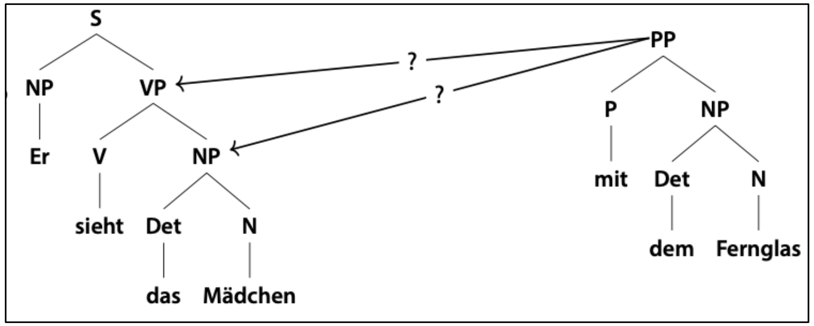
Unabhängigkeitsannahmen von PCFGs

2. Annahme: Unabhängigkeit vom Kontext

- **Erklärung**:
  - In probabilistischen Modellen könnte angenommen werden, dass die **Wahrscheinlichkeiten von Teilbäumen unabhängig vom Kontext in der Struktur** sind (z. B. welches Elternknoten den Teilbaum dominiert).
  - Beispiel: Die Wahrscheinlichkeit von VP→V+NP wird unabhängig betrachtet von ihrer Umgebung, etwa davon, ob der Satz ein Fragesatz oder ein Hauptsatz ist.
- **Problem**:
  - Dies ignoriert linguistische Zusammenhänge – z. B. beeinflusst der Hauptsatz möglicherweise die Struktur des VP.
- Model:
  - **<u>History-based PCFGs</u>** ⇒ Auflösung kontextabhängiger struktureller Ambiguität

Lexikalische PCFGs

Eine nicht-lexikalisierte PCFG gibt bei ambigen Sätzen immer dieselbe (die wahrscheinlichere) Struktur zurück, unabhängig von den verwendeten Wörtern.



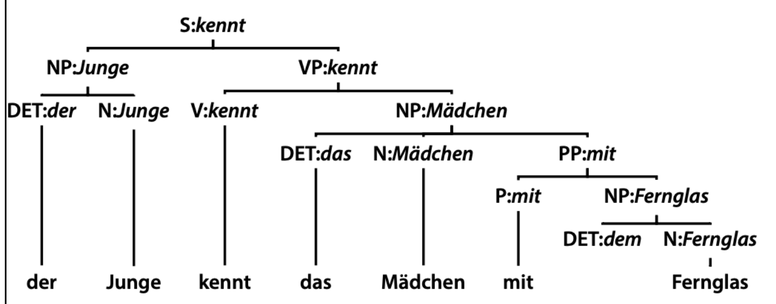
Bsp. PP-Attachment Ambiguität

Welche Struktur jedoch tatsächlich wahrscheinlichere (bzw. sinnvollere) ist, hängt vom Vokabular ab:

Er **sieht** das **Mädchen** mit dem Fernglas. - Beide Lesarten sinnvoll
Er **sieht** das **Huhn** mit dem Fernglas. - VP-Attachment bevorzugt
Er **kennt** das Mädchen mit dem Fernglas. - NP-Attachment bevorzugt

Lexikalische PCFGs

Lösung: Kopfannotation



Phrasenköpfe werden hochgereicht (**Kopf-Perkolation**) und jedes Nicht-Terminal wird mit dem Phrasenkopf annotiert.

Einige Probleme:

- Regelvervielfachung: statt der allgemeinen Regel VP –> V NP haben wir jetzt die Regeln:
  VP(sieht) –> V(sieht) NP(Mädchen)
  VP(kennt) -> V(kennt) NP(Mädchen). usw. für das gesamte Vokabular
- umfangreiche Trainigsdaten notwendig
- Probleme bei ungesehenen Wörtern (sparse-data problem)

Lexikalische PCFGs

Beispiel einer **Kopfannotation** mit Hilfe eines **HEAD-Features** in einer FCFG

```
sentence = "er erklimmt den Berg"

gramstring = r"""
% start S
    S[HEAD=?v]    -> NP[] VP[HEAD=?v]
    VP[HEAD=?v]   -> V[HEAD=?v] NP[]
    NP[HEAD=?n]   -> Det[] N[HEAD=?n]
    NP[HEAD=?n]   -> N[HEAD=?n]

    Det[] -> "den"
    N[HEAD="er"]    -> "er"
    N[HEAD="Berg"]    -> "Berg"
    V[HEAD="erklimmt"]    -> "erklimmt"
"""
```

```
                              S[HEAD='erklimmt']
         ┌───────────────────────────┴────────────┐
         │                                 VP[HEAD='erklimm
         │                                        t']
         │             ┌──────────────────────────┴──────────┐
NP[HEAD='er']          │                                NP[HEAD='Berg']
         │             │                          ┌──────────┴──────────┐
 N[HEAD='er'] V[HEAD='erklimmt']               Det[]                N[HEAD='Berg']
         │             │                          │                     │
        er         erklimmt                      den                   Berg
```

History-based PCFGs

Eine herkömmliche PCFG berücksichtigt nicht die Position einer Regelanwendung im Parsebaum, d.h. der **Kontext** wird in die Berechnung einer Regelwahrscheinlichkeit nicht mit einbezogen.
Oft sind die Regelwahrscheinlichkeiten jedoch abhängig von den zuvor angewandten Regeln.

Bsp.:
Die **Wahrscheinlichkeit der Regel NP -> Pron**
ist **höher bei Subjekt-NPs** (wenn die zuvor angewandte Regel S –> NP VP war) **als bei Objekt-NPs** (wenn die zuvor angewandte Regel VP -> V NP war).

History-based PCFGs

erwünschte Regelgewichtung Subjekt (S-dominiert):

NP → PRON **0.91**
NP → DET N 0.09

erwünschte Regelgewichtung Objekt (VP-dominiert):

NP → PRON 0.34
NP → DET N **0.66**

normale PCFG (keine Differenzierung, Daten aus Korpus):

NP → PRON **0.25**
NP → DET N **0.28**

Lösung: Splitting NP-Kategoriensymbol (*parent annotation*):

NP^S → PRON 0.91
NP^S → DET N 0.09
NP^VP → PRON 0.34
NP^VP → DET N 0.66

Lösung: Parent Annotation

- nicht-terminale Knoten werden mit der Kategorie des Elternknotens (= history) annotiert
- als Trennzeichen verwenden wir das Zeichen ^ (z.B. NP^S)
- Nicht-Terminale werden dadurch in mehrere Kategorien aufgespalten

Probleme:

- Regelvervielfachung
- Probleme bei unbekannter Vorgängerkategorie

History-based PCFGs

Beispiel einer **Parent Annotation**

```
sentence = "er erklimmt den Berg"

grammar = nltk.CFG.fromstring("""
    S     -> NP^S VP^S
    VP^S   -> V^VP NP^VP
    NP^VP   -> Det^NP N^NP
    NP^S   -> N^NP

    Det^NP -> "den"
    N^NP    -> "er"
    N^NP    -> "Berg"
    V^VP    -> "erklimmt"
```