

Übersicht

12 Statistische Syntaxmodelle

12.1 Induzierte PCFG-Modelle

- 12.1.1 *grammar induction* aus Treebank
- 12.1.2 Normalisierung und Evaluation
- 12.1.3 PCFGs mit abgeschwächten Unabhängigkeitsannahmen
- 12.1.4 Lexikalisierte PCFGs
- 12.1.5 *history-based* PCFGs

12.2 Dependenzbasierte Modelle

- 12.2.1 Dependenzgrammatiken
- 12.2.2 Übergangsbasiertes Dependency-Parsing
- 12.2.3 Graphbasiertes Dependency-Parsing
- 12.2.4 *Dependency-Treebanks*

12 Statistische Syntaxmodelle

12.1 Induzierte PCFG-Modelle

- **Grammatikentwicklung (*grammar writing*) ist aufwendig**
 - Grammatiken mit **von Experten geschriebenen Regeln** mit hoher *Abdeckung*
- **Alternative: Induktion von Grammatikregeln aus Korpora**
 - **empirisches Syntaxmodell**
 - Berücksichtigung **relativer Häufigkeiten der Regeln** ⇒ PCFG
 - als **statistisches Modell**: direkte Verwendung zur **Disambiguierung**

12.1.1 *grammar induction* aus Treebank

- **Treebank als implizite Grammatik**
 - jeder **Teilbaum** der Tiefe 1 als **implizite CFG-Regel**
 - **Expansion eines Nonterminals**
- **Extraktion von CFG-Regeln** aus den Ableitungen der Treebank
- Frequenzbestimmung der Regeln und Berechnung **Regelwahrscheinlichkeiten** über **relative Häufigkeiten** (\Rightarrow PCFG)
 - **Gewichtung** insbesondere **bei induzierter Grammatik notwendig**: viele Regeln \Rightarrow hohe Ambiguität
- Anwendung von **Smoothing und Normalisierung**

- **Form der induzierten Grammatik** hängt stark vom **Annotationschema** der dem Training des Modells zugrundeliegenden Treebank ab:
 - **flache Grammatik = viele Regeln:**
 - Penn-Treebank: 1 Mill. *Worttokens*, 1 Mill. *nicht-lexikalische Regel-Tokens*, 17.500 *Regel-Types*
 - z. B. jedes PP-Adjunkt mit eigener Regel:
 $VP \rightarrow V PP, VP \rightarrow V PP PP, VP \rightarrow V PP PP PP$ usw.
 - **tiefere Bäume: mehr Nonterminale, weniger Regeln**

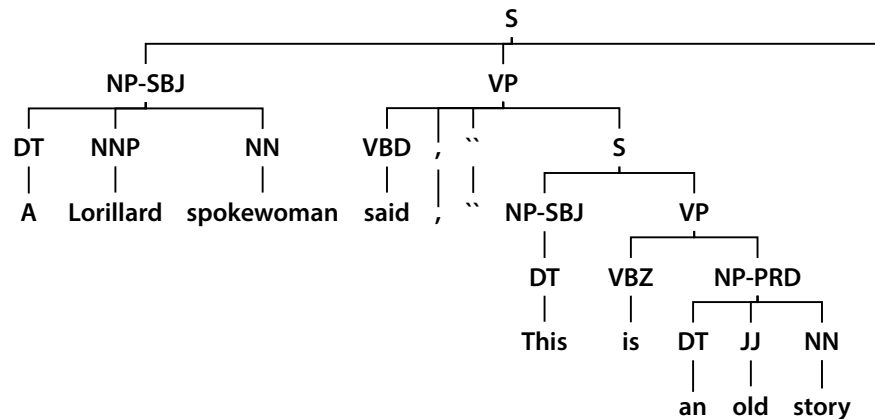
Auflistung 1: Ableitungsbaum in Penn-Treebank

```
1 from nltk.corpus import treebank
2 t=nltk.corpus.treebank.parsed_sents('wsj_0003.mrg')[4]
3 print(t)
4 # (S
5 #   (NP-SBJ (DT A) (NNP Lorillard) (NN
6 #     spokewoman))
7 #   (VP
8 #     (VBD said)
9 #     (, ,)
10 #    (` `)
11 #    (S
12 #      (NP-SBJ (DT This))
13 #      (VP (VBZ is) (NP-PRD (DT an) (JJ old)
14 #        (NN story))))))
15 #   (. .))
```

Auflistung 2: NLTK: Extraktion von Grammatikregeln aus Treebank (*nltk.induce_pcfg*)

```
1  ##http://www.nltk.org/book/ch08-extras.html
2  ##http://www.nltk.org/_modules/nltk/grammar.html#indu
3
4  import nltk
5  from nltk.corpus import treebank
6
7  productions = []
8  S = nltk.Nonterminal('S')
9  for tree in
    nltk.corpus.treebank.parsed_sents('wsj_0003.mrg')[4
10     productions += tree.productions()
11
12  grammar = nltk.induce_pcfg(S, productions)
13  for production in grammar.productions():
14     print(production)
```

NP-SBJ → DT NNP NN [0.5]
 DT → 'A' [0.333333]
 NNP → 'Lorillard' [1.0]
 NN → 'spokewoman' [0.5]
 VP → VBD , `` S [0.5]
 VBD → 'said' [1.0]
 , → ',' [1.0]
 `` → '``' [1.0]
 S → NP-SBJ VP [1.0]
 NP-SBJ → DT [0.5]
 DT → 'This' [0.333333]
 VP → VBZ NP-PRD [0.5]
 VBZ → 'is' [1.0]
 NP-PRD → DT JJ NN [1.0]
 DT → 'an' [0.333333]
 JJ → 'old' [1.0]
 NN → 'story' [0.5]
 . → '.' [1.0]



Stanford-PCFG-Parser:

- basiert auf **aus Treebanks extrahierten PCFG-Modellen**
→ <https://nlp.stanford.edu/software/lex-parser.shtml>
- Trainingskorpus des englischen Modells (`englishPCFG.ser.gz`):
Penn Treebank
- Trainingskorpus des deutschen Modells (`germanPCFG.ser.gz`):
NEGRA Korpus

12.1.2 Normalisierung und Evaluation

Normalisierung

- **Chomsky-Normalform (CNF)**

→ Einschränkung der Form von CFG-Regeln:

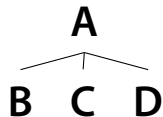
⇒ **RHS: 2 Nichtterminale oder 1 Terminal:** $A \rightarrow B C, A \rightarrow a$

→ **Binärbäume** (bis Präterminalknoten, dort: unäre Bäume)

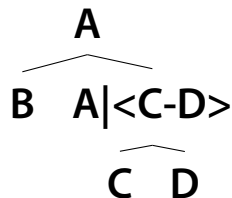
→ **jede CFG kann in CNF umgewandelt werden:**

$A \rightarrow B C D \Rightarrow A \rightarrow B X, X \rightarrow C D$

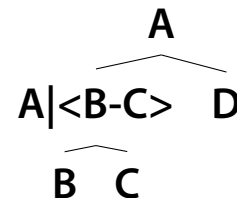
Original:



Right-Factored:



Left-Factored:



- **Anwendung Chomsky-Normalform:**
 - notwendig für **CYK-Chart-Parsing**
 - zur **Reduktion von extrahierten Grammatikregeln** aus flach annotiertem Korpus:
 - * $VP \rightarrow V PP$
 $VP \rightarrow V PP PP$
 $VP \rightarrow V PP PP PP$ usw.
 - * mit **Chomsky-adjunction** ($A \rightarrow A B$):
 $VP \rightarrow V PP$
 $VP \rightarrow VP PP$

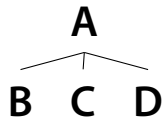
- **Parent Annotation**

→ Kategorie des **Mutterknoten** in **Kategoriensymbol** aufnehmen

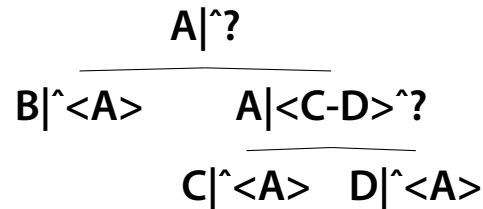
→ Modellierung von **Kontext**; s. unten: *history-based PCFGs*

→ ergibt anderes PCFG-Modell: **mehr Nichtterminale, andere Gewichtung**

Original:



Parent Annotation (+CNF):

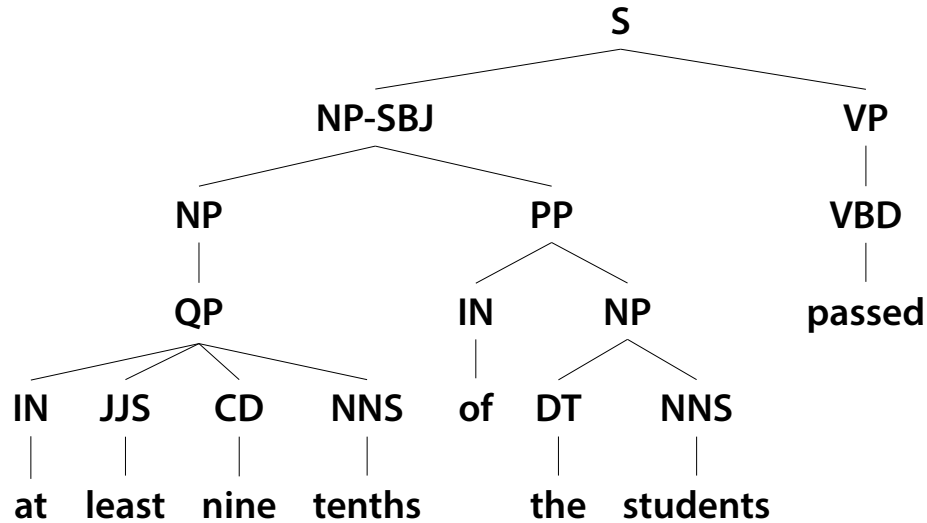


Auflistung 3: NLTK: Transformationen

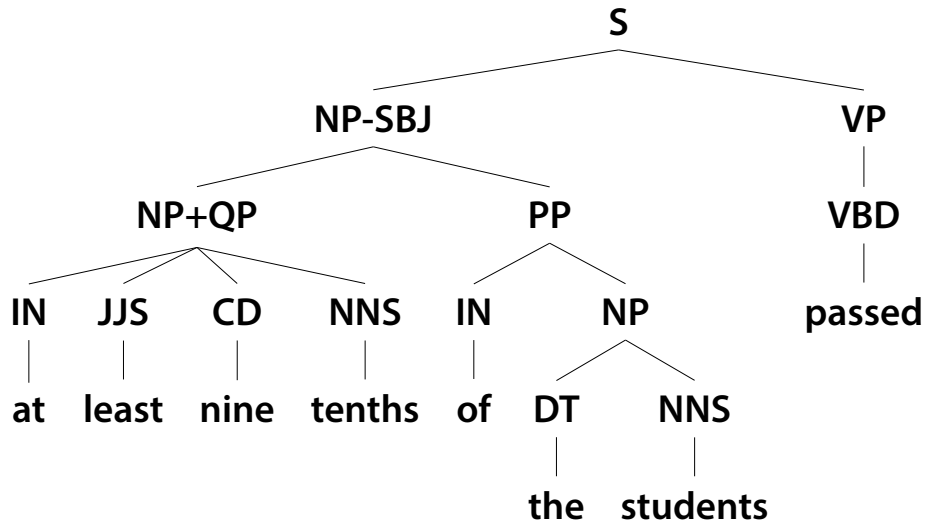
```
1  ##http://www.nltk.org/\_modules/nltk/treetransforms.ht
2  from nltk.draw.tree import draw_trees
3  from nltk import tree, treetransforms
4  from copy import deepcopy
5  sentence = """(S (NP-SBJ (NP (QP (IN at) (JJS
    least) (CD nine) (NNS tenths)) )
6      (PP (IN of) (NP (DT the) (NNS students)
7          ))) (VP (VBD passed))))"""
8
9  # collapse subtrees with only one child
10 collapsedTree = deepcopy(t)
11 treetransforms.collapse_unary(collapsedTree,
    collapsePOS=False)
```

```
12 # convert the tree to CNF
13 cnfTree = deepcopy(collapsedTree)
14 treetransforms.chomsky_normal_form(cnfTree)
15
16 # convert the tree to CNF with parent
   annotation (one level)
17 parentTree = deepcopy(collapsedTree)
18 treetransforms.chomsky_normal_form(parentTree,
   vertMarkov=1)
19
20 # convert the tree to CNF with parent
   annotation (two levels)
21 grandparentTree = deepcopy(collapsedTree)
22 treetransforms.chomsky_normal_form(grandparentTree,
   vertMarkov=2)
```

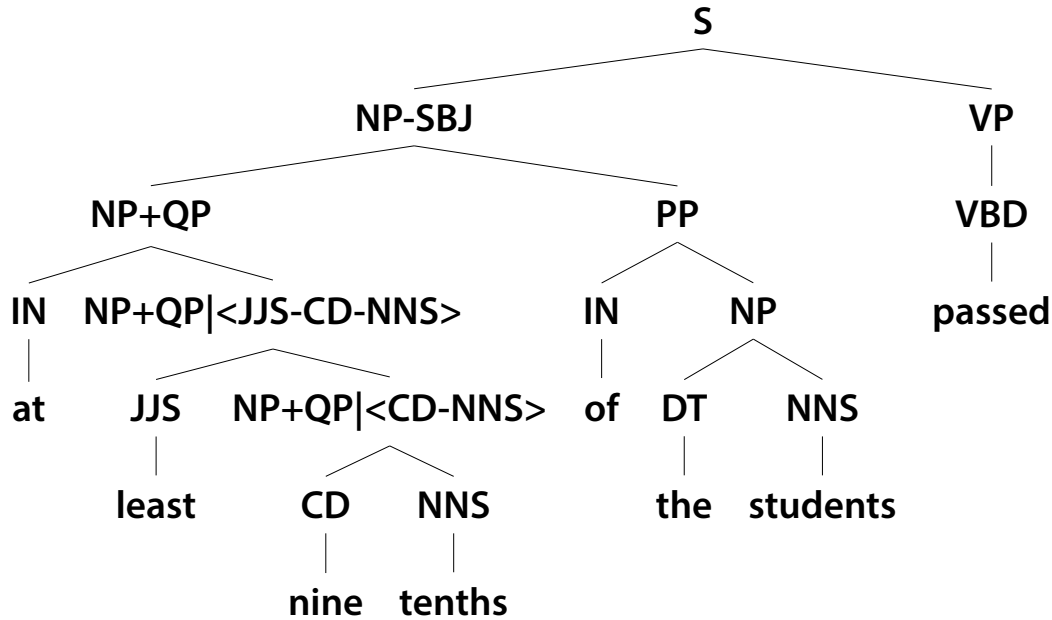
- **Original:**



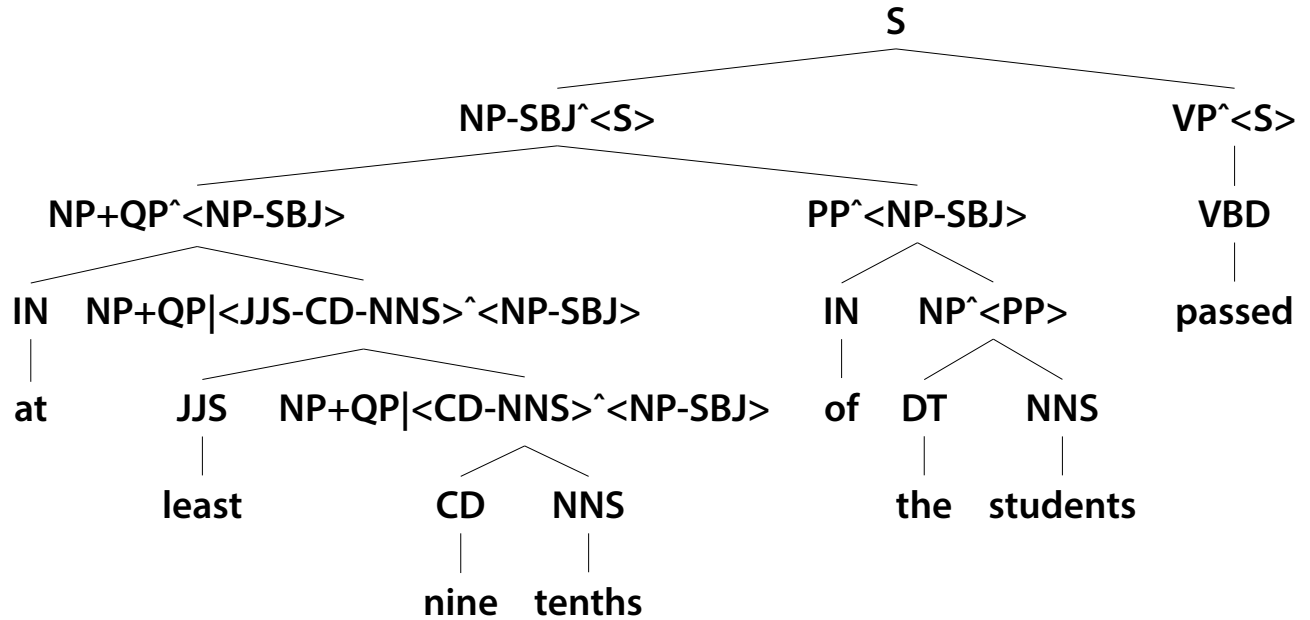
- ohne unäre Teilbäume = *unary reduction* (collapsePOS=False):



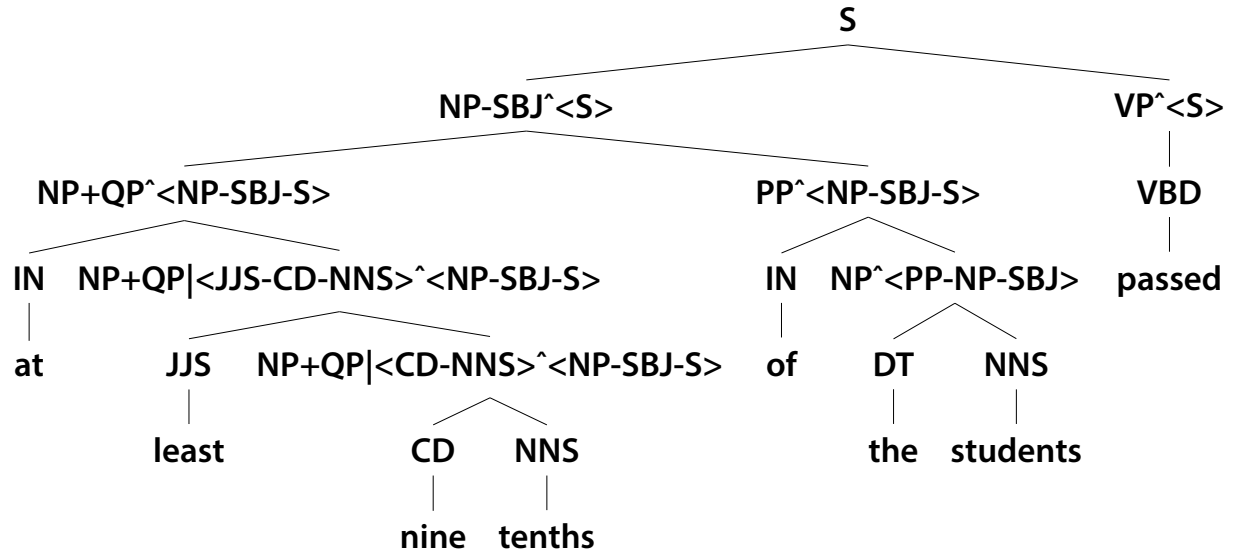
- Chomsky-Normal-Form (mit *unary reduction*):



- Parent Annotation (mit *unary reduction* und CNF):



- Parent Annotation mit Großvaterkategorie:



Evaluation

- Messen der **Güte von Grammatikmodellen/Parsern** durch Parsen von Sätzen einer **Testmenge**
→ **Teilmenge einer hand-annotierten Treebank** = *gold-standard*-Ableitungen, z. B. von Penn-Treebank
- **PARSEVAL-Maße** (Black et al. 1991): Übereinstimmung von Konstituenten in den Ableitungen von **geparsten Daten** (**Ableitungshypothese H**) mit denen der **Test-Daten** (**Referenz-Ableitung R**)
→ Konstituente ist **korrekt** wenn Übereinstimmung in **Nichtterminal-Symbol** und **Spanne (gleicher Start- und Endpunkt)**

- **Recall** = $\frac{(\text{Anzahl von korrekten Konstituenten in Hypothese})}{(\text{Anzahl von Konstituenten in Referenz-Ableitung})}$
- **Precision** = $\frac{(\text{Anzahl von korrekten Konstituenten in Hypothese})}{(\text{Anzahl von allen Konstituenten in Hypothese})}$
 - Hypothese: (A) (B C D)
 - Referenz: (A) (B) (C) (D)
 - Recall = 1/4; Precision: 1/2
- **cross-brackets**: Anzahl an Konstituenten mit ((A B) C) in Ableitungshypothese aber (A (B C)) in Referenz-Ableitung
- **moderne Parser**: ca. **90% Precision und Recall**, ca. **1% cross-brackets**-Konstituenten (trainiert und getestet mit Penn-Treebank)

12.1.3 PCFGs mit abgeschwächten Unabhängigkeitsannahmen

2 Unabhängigkeitsannahmen von PCFGs

- **Annahme Unabhängigkeit von lexikalischem Material**
→ Wahrscheinlichkeiten von Teilbäumen sind unabhängig von Terminalen
- **Annahme Unabhängigkeit von Kontext**
→ Wahrscheinlichkeiten von Teilbäumen sind unabhängig von Elternknoten
- **Zurücknahme von Unabhängigkeitsannahmen:**
 - ⇒ **beschreibungsadäquatere Syntaxmodelle**
 - ⇒ **Berücksichtigung linguistischer Abhängigkeiten**

- Berücksichtigung **lexikalischer Abhängigkeiten**:
 - ⇒ ***lexikalisierte PCFGs***
 - ⇒ Auflösung **lexikalischer Ambiguität**
- Berücksichtigung **struktureller Abhängigkeiten zwischen Regeln**:
 - ⇒ ***history-based PCFGs***
 - ⇒ Auflösung **kontextabhängiger struktureller Ambiguität**

12.1.4 Lexikalisierte PCFGs

Lexikalisierte PCFGs

- PCFGs basierend auf einfachen CFG-Regeln:
⇒ nur **strukturelle Disambiguierung**
- Probleme mit **lexikalisch determinierter Ambiguität**, z. B. bei **Subkategorisierung** oder **PP-Attachment**
- **statisches Modellierung lexikalischer Abhängigkeiten**
- bekannter lexikalisierte Parser: Collins Parser (Collins, 1999)

Vorgehen Lexikalisierung

- **bottom-up-Annotation** nichtterminaler Kategorien **mit lexikalischer Information** (Kopf-Perkolation): VP (kennt)
- auch Annotation mit **Part-of-Speech-Tag** möglich: NP (er, PRON)

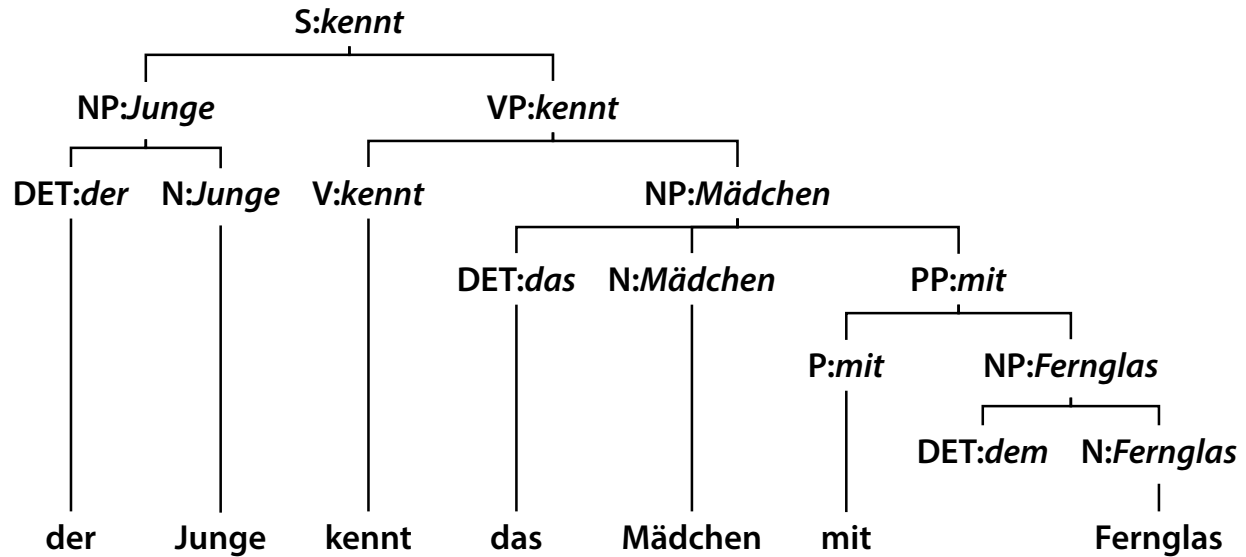
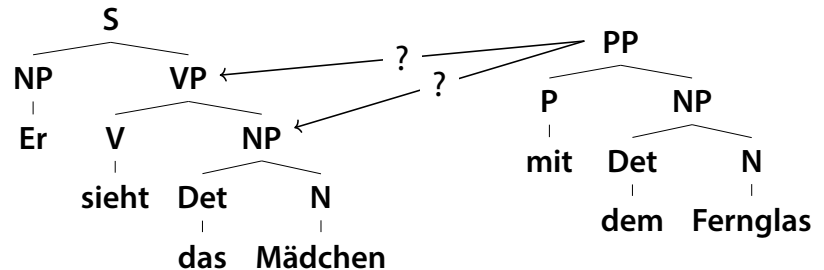


Abbildung 1: Beispiel für lexikalisierte Phrasenstruktur

PP-Attachment



- strukturelle Ambiguität:
NP- oder VP-Anbindung?
 ⇒ 2 strukturelle Lesarten:
 → (VP V (NP N PP))
 → (VP V (NP N) PP)
- **unlexikalisierte PCFG:** immer Entscheidung für eine Variante
 → z. B. **englisches Trainigskorpus: NP-Attachment-Frequenz etwas höher**

- häufig: Anbindung lexikalisch konditioniert (**lexikalische Abhängigkeit**):
 - **Bevorzugung von VP-Anbindung:** *Sie stellt die Blumen ins Wasser.*
→ engere Verbindung von *stellt* mit *ins* als zwischen *Blumen* und *ins*
 - **Bevorzugung von NP-Anbindung:** *Der Junge kennt das Mädchen mit dem Fernglas.*
→ engere Verbindung von *Mädchen* mit *mit* als zwischen *kennt* und *mit*

Subkategorisierung

- **statische Modellierung Subkategorisierung** statt regelbasiert über Subkategorisierungsrahmen
- **transitive Verben:** hohe Wahrscheinlichkeit $P(VP \rightarrow V \ NP)$
 $\rightarrow P(V \ NP \mid VP, \text{sehen}) > P(V \mid VP, \text{sehen})$
- **intransitive Verben:** hohe Wahrscheinlichkeit $P(VP \rightarrow V)$
 $\rightarrow P(V \mid VP, \text{laufen}) > P(V \ NP \mid VP, \text{laufen})$

Probleme lexikalisierter PCFGs

- **Modell wird sehr groß**
 - Grund: **viel mehr Ereignisse** durch lexikalisierte Regeln
 - **Regelvervielfachung:**
 $VP(sieht) \rightarrow V(sieht) NP(Mädchen)$
 $VP(kennt) \rightarrow V(kennt) NP(Mädchen)$
- **umfangreiche Trainingsdaten** notwendig für Parameterabschätzung des Modells

- **neue Abschätzung** für Regelwahrscheinlichkeiten notwendig
→ **MLE-Abschätzung** über $P(\alpha \rightarrow \beta | \alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)}$ ist **zu spezifisch**
→ **geht meistens gegen 0**, da **nur sehr wenige Instanzen** der lexikalisierten Regeln in Trainingskorpus vorhanden
- ***sparse data***-Problem aufgrund von in Trainingsdaten **ungesehenen Wörtern/Instanzen** (\Rightarrow keine Regel vorhanden)
→ Lösung: **Backoff** = **Verzicht auf Lexikalisierung** bei **unbekanntem lexikalischem Kopf**

- dazu notwendig: **Smoothing** (Glättung der Regelwahrscheinlichkeiten)
 - **Reservierung von Wahrscheinlichkeitsmasse** für Regeln bei Backoff bei ungesehenen Köpfen
 - Zuordnung von Wahrscheinlichkeit für Regel mit **ungesehenem Kopf**
 - z. B. **Laplace-Smoothing**: zu jeder Häufigkeit im Korpus: **Wert addieren** (1 = **Add-One-Smoothing**) \Rightarrow Backoff-Regel: $P > 0$
- **Backoff bei Collins Parser**: unbekannte Köpfe aus Testmenge und aus Trainingsmenge mit Frequenz < 6 werden mit UNKNOWN ersetzt

12.1.5 *history-based* PCFGs

***history-based* PCFGs**

- Berücksichtigung **Abhängigkeit Expansion von Kontext**
 - **Regelauswahl abhängig von vorheriger Regelauswahl**
 - Wahrscheinlichkeit einer Expansion ist abhängig von der **Position im Strukturbaum**
- z. B. **unterschiedliche Expansionswahrscheinlichkeiten für NPs** in Subjekt- bzw. Objektposition
 - **Subjekt-NP** (S-dominiert) erweitert **wahrscheinlicher zu Pronomen als Objekt-NP** (VP-dominiert)
 - $P(\text{NP} \rightarrow \text{PRON} \mid \text{S}) > P(\text{NP} \rightarrow \text{PRON} \mid \text{VP})$
 - $P(\text{PRON} \mid \text{NP}, \text{S}) > P(\text{PRON} \mid \text{NP}, \text{VP})$

- Grund = **Informationsstruktur**

→ **Subjekt** typischerweise Topik = **bekannte Information**, die durch Pronomen ausgedrückt wird

	Pronomen	Nicht-Pronomen
Subjekt	91%	9%
Objekt	34%	66%

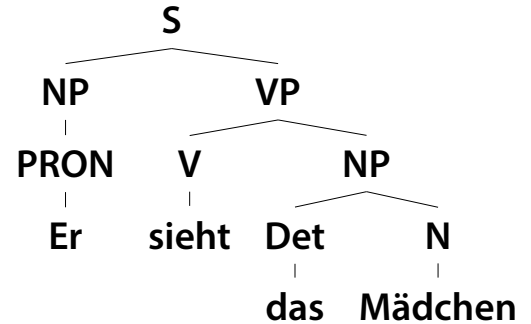


Abbildung 2: Verteilung der Form von Subjekt und Objekt in englischem Korpus (nach Francis et al., 1999, vgl. SLP2, 502)

- **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*):**

$\text{NP}^{\text{S}} \rightarrow \text{PRON } 0.91$

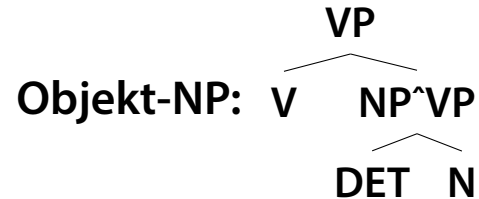
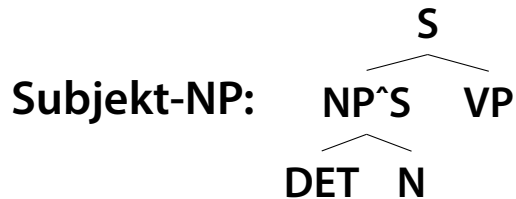
$\text{NP}^{\text{S}} \rightarrow \text{DET N } 0.09$

$\text{NP}^{\text{VP}} \rightarrow \text{PRON } 0.34$

$\text{NP}^{\text{VP}} \rightarrow \text{DET N } 0.66$

Vorgehen

- **Annotation nichtterminaler Kategorien mit Kategorie des Mutterknotens (= *history*)**
 - ⇒ ***parent annotation***
 - Subjekt-NP: NP^S
 - Objekt-NP: NP^{VP}
 - "*Splitting* von Nicht-Terminalen"



Probleme von *history-based* PCFGs

- **ähnlich** wie bei **Lexikalisierung**, aber weniger stark ausgeprägte Regelvervielfachung durch *parent annotation*
→ *sparse data*: **unbekannte Vorgängerkategorie**
- kleinere Regelmenge durch **selektive *parent annotation***
→ **nur Splitten**, wenn ***accuracy* erhöht** wird

12.2 Dependenzbasierte Modelle

12.2.1 Dependenzgrammatiken

- in Computerlinguistik sind traditionell **Konstituenten-basierte Formalismen dominant** (Chomsky-Tradition Generativer Grammatik)
- **Dependenzbasierte Syntaxmodelle** werden immer wichtiger
- statt Konstituentenstruktur (= Regeln der Zusammensetzung syntaktischer Einheiten):
⇒ **Modellierung Syntax über binäre Abhängigkeitsrelationen zwischen Wörtern im Satz (grammatische Relationen)**

- **Dependenzgraph** $G = (V, A)$:
 - Menge von Knoten ($V = Vertices$)
 - Menge von (gelabelten) gerichteten Kanten ($A = Arcs$)
- **Induktion von Dependenz-Parsing-Modellen aus Treebank möglich**
 - **Dependency-Treebank** kann handannotiert sein oder abgeleitet aus PSG-Treebank
- Dependenzanalysen können auch **sekundär aus Analysen mit konstituentenbasierten Parsern** erzeugt werden
 - z. B. Stanford-PCFG-Parser

Vorteile von Dependenzmodellen

- **Relationale Informationen direkt** vorhanden statt indirekt über Position in Strukturbaum
→ Verwendung z. B. für **Informationsextraktion**
- **Wortgrammatik** = direkte Modellierung von Relation zwischen Wörtern
→ **keine Lexikalisierung notwendig**

- **Dependenzgrammatik als Wortgrammatik**
 - ⇒ reduziert *sparse data*-Problem bei Parameterabschätzung
 - bei induziertem PCFG-Modell mit flacher Struktur treten häufig ungesehene Bäume auf, z. B.: $VP \rightarrow V \ NP \ PP \ PP$
 - ⇒ nur Backoff zu 'UNKNOWN'-Wahrscheinlichkeit möglich
 - **Dependenzgrammatik teilt Struktur auf in binäre Relationen** zwischen den Wörtern
 - jede *head-dependent*-Relation wird **separat gewichtet**
 - **Beispiel:** $VP \rightarrow V \ NP \ PP$ und $VP \rightarrow V \ PP \ PP$ in Treebank
 - ⇒ man kann **aus den gesehenen *head-dependent*-Beziehungen** ($V \Rightarrow N$, $V \Rightarrow P$, $V \Rightarrow P$) die von $VP \rightarrow V \ NP \ PP \ PP$ **ohne Backoff abschätzen**

wichtige Anwendungsgebiete von Dependenzanalysen

- **Informationsextraktion**
→ *relation extraction* (s. Übung)
- **Semantisches Parsing**
- **Question Answering**

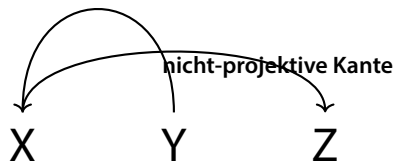
Regelbasierte Dependenzgrammatik-Modelle

- **von Experten erstellte** Dependenzgrammatiken
- wichtige Dependenzgrammatik-Formalismen:
Bedeutung-Text-Modell (I.A. Mel'čuk), Word Grammar, Link Grammar, Constraint-Grammar

- Modellierung über (lexikalisierte) **kontextfreie Grammatiken**
→ nur projektive Strukturen möglich
- Modellierung über **Constraint-basierte Dependenzgrammatiken**
→ **Angabe von Wohlgeformtheitsbedingungen**
→ **Entfernung von Constraint-verletzenden Graphen im Parsing**
- **Constraint-Parsing: Verarbeitung von nicht-projektiven Strukturen**

Nichtprojektivität

- **projektive Struktur:** alle Kanten sind **projektiv**, d. h. es gibt einen Pfad vom Kopf der Relation zu jedem Wort zwischen Kopf und Dependent
- **nicht-projektive Struktur:** Überschneidung von Kanten
→ z. B.: Dependent eines Wortes folgt nach dessen Kopf



- bei **Ableitung Dependenzgrammatik** von PSG-Treebanks durch *head-finding-rules* ergeben sich **automatisch projektive** Strukturen
- **linguistisch: nicht-projektive Strukturen** entstehen durch **diskontinuierliche Elemente**
→ **freie Wortstellung** und *long distance dependencies*

- Dependenzgrammatiken sind **besser** als Konstituentengrammatiken **geeignet, diskontinuierliche Strukturen abzubilden**
 - Modellierung **relationaler Struktur**, nicht der linearen Anordnung
 - Dependenzstruktur **abstrahiert von der linearen Anordnung**
 - **bei Verarbeitung** (Parsing) können **nicht-projektive Strukturen aber problematisch** sein

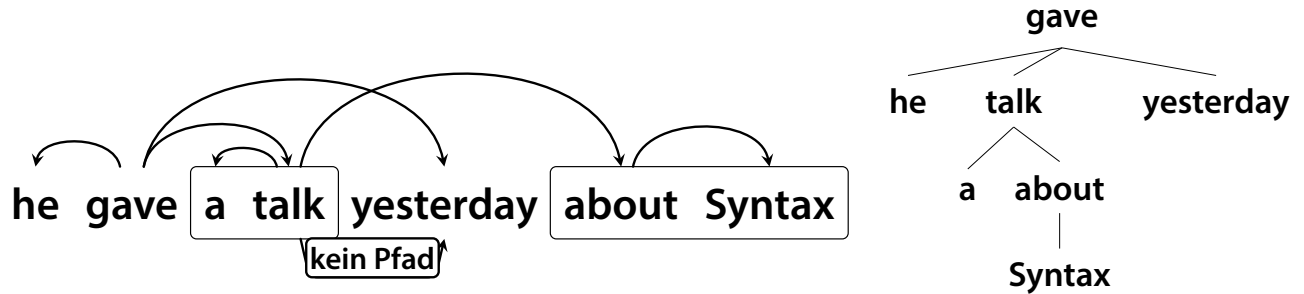


Abbildung 3: *Dependenzanalyse diskontinuierlicher = nicht-projektiver Struktur (mit und ohne Berücksichtigung linearer Ordnung)*

Auflistung 4: NLTK: nicht-projektive Dependenzgrammatik

```
1  #non-projective dep parsing in nltk
2  #http://www.nltk.org/howto/dependency.html
3
4  from nltk.grammar import DependencyGrammar
5  grammar = DependencyGrammar.fromstring("""
6      'gave' → 'he' | 'talk' | 'yesterday'
7      'talk' → 'a' | 'about'
8      'about' → 'Syntax'
9      """)
10
11 from nltk import NonprojectiveDependencyParser
12 dp = NonprojectiveDependencyParser(grammar)
13 g, =
14     dp.parse(['he', 'gave', 'a', 'talk', 'yesterday', 'about'])
15 print(g.root['word'])
```

```
15 #gave
16 for _, node in sorted(g.nodes.items()):
17     if node['word'] is not None:
18         print('{address} {word}:
           {d}'.format(d=node['deps'][''],
           **node))
19 #1 he: []
20 #2 gave: [1, 4, 5]
21 #3 a: []
22 #4 talk: [3, 6]
23 #5 yesterday: []
24 #6 about: [7]
25 #7 Syntax: []
26 print(g.tree())
27 #(gave he (talk a (about Syntax)) yesterday)
```

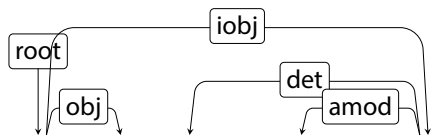

12.2.2 Übergangsbasiertes Dependency-Parsing

- **Literatur:** Speech and Language Processing (Ed. 3). Daniel Jurafsky & James H. Martin. **Chapter 14:** <https://web.stanford.edu/~jurafsky/slp3/14.pdf>
- **Induktion** von Dependenz-Parsing-Modellen **aus Treebanks**
- **Dependency-Treebanks** können direkt von **Experten** erstellt sein oder **abgeleitet aus Phrasenstruktur-Treebanks** (s. u.)

Übergangsbasiertes Parsing

- verwendet Stack-basierten **Shift-Reduce-Algorithmus**
- **SHIFT-Operation:** Wörter in **Wortliste** (Buffer) auf Stack
 - Stack wird mit `root`-Knoten **initialisiert**
 - Abschluss, wenn Wortliste leer und nur noch `root` auf Stack
- **REDUCE-Operation:**
 - statt Ersatz durch Nonterminal (CFG):
 - ⇒ **Hinzufügen von Relation zwischen den beiden obersten Elementen auf dem Stack**
 - ⇒ **Löschen des Dependents vom Stack**

- **2 mögliche REDUCE-Operationen** (je nach Position Kopf):
 - **LEFTARC** = *head-final*: the \leftarrow flights
 - **RIGHTARC** = *head-initial*: book \rightarrow me
 - (für Labeling: Subdifferenzierung in **RIGHTARC**(nsubj) usw.)
- Je nach ***Konfiguration von Stack, Buffer und Menge der erkannten Relationen*** wird eine bestimmte **Operation ausgeführt** \Rightarrow **Übergang (*transition*)**
 - \rightarrow bei CFG: Grammatikregeln entscheiden über Operation
 - \rightarrow hier: **Entscheidung** über anzuwendende Operation **durch trainiertes Modell**, dass **Konfigurationen auf Übergänge** abbildet



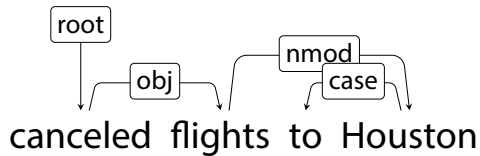
Book me the morning flight

Step	Stack	Word List (Buffer)	Transition	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	□	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	□	LEFTARC	(the ← flight)
8	[root, book, flight]	□	RIGHTARC	(book → flight)
9	[root, book]	□	RIGHTARC	(root → book)
10	[root]	□	Done	

Training übergangsbasiertes Parsing-Modell

- **Klassifikator**: sagt den **besten Übergang** für die **aktuelle Konfiguration** voraus
- wird durch anhand **von einer Treebank erzeugten Beispielen** trainiert
- Beispiele = **Konfiguration-Übergang-Paare**
 - Anwendung des Algorithmus auf die Sätze der Treebank
 - Verwendung der *head-dependent*-Relationen der Treebank zur **Bestimmung des Übergangsoperators** jeder Konfiguration

- bei jeder Konfiguration: **Abgleich**, ob eine **ARC-Operation** eine **Relation aus der Menge der Relationen** in der Referenzanalyse der Treebank für den Satz produziert
 - **Einschränkung bei *RIGHTARC***: nur, wenn der Dependent der möglichen Relation nicht Kopf einer der Relationen aus der Menge offener Relationen ist
 - Einschränkung verhindert, dass **Wort zu früh vom Stack** genommen wird



	Stack	Word List (Buffer)	Transition	
	[root,canceled,flights]	[to, Houston]	SHIFT oder RIGHTARC ?	
gewählte Operation				Relation Added
SHIFT	[root,canceled,flights,to]	[Houston]		-
RIGHTARC	[root,canceled]	[to, Houston]		(canceled → flights)

– richtiger Übergang: SHIFT

→ bei RIGHTARC wird *flights* zu früh vom Stack entfernt; Relation (*flights* → *Houston*) wäre dann nicht mehr möglich

- **Klassifikationsaufgabe: Einteilung von Konfigurationen** (Menge von Objekten) **in Klassen** (= **Übergangsoperationen**)
 - ***supervised learning***: **Klassen** (Übergänge) sind aus Trainingsdaten **bekannt**
 - Lernen einer **Funktion**, die von **Konfigurationen auf Übergangs-Operatoren** abbildet
- Konfigurationen werden **durch Merkmalsvektor repräsentiert**
- Merkmale werden durch **Feature-Extraktion** gewonnen

- **häufig verwendete Merkmale:**
 - **Token, Lemma und POS** von Wörtern **auf Stack** (die Obersten)
 - **Token, Lemma und POS** von Wörtern in **Wortliste** (die Vordersten)
 - **Relationen** zwischen den berücksichtigten Wörtern
- **verwendete Klassifikationsverfahren:** logistische Regression, Support Vector Machine oder Neuronale Netze (z. B.: <https://nlp.stanford.edu/software/nndep.shtml>)

Übergangsbasierte Parsing-Systeme:

- **Malt-Parser** (Nivre et al.): *transition-based Dependency Parser*
→ <http://maltparser.org>
- **Stanford-Dependency-Parser** (Manning et al.):
 - neben der **Transformation von PCFG-geparsten Konstituentenbäumen in Dependenzgraphen** (`englishPCFG.ser.gz`):
 - **Transition-based Dependency-Parsing-Modell:**
→ `englishFactored.ser.gz`: verwendet PCFG-Parser und Dependenz-Parser und vergleicht Ergebnisse
→ <https://nlp.stanford.edu/software/parser-faq.html#y>

12.2.3 Graphbasiertes Dependency-Parsing

Graphbasiertes Parsing

- **Suche im Raum möglicher Dependenzbäume** nach dem am besten bewerteten
- **Konstruktion von gerichtetem Graph** mit den **Wörtern des Satzes als Knoten** und allen möglichen *head-dependent*-Relationen als Kanten
- trainiertes Modell gibt **Gewichte für head-dependent-Relationen** (ähnlich wie bei PCFG)

- **Berechnung des Baumes** (Teilgraph) mit **maximalem Gewicht** (*maximum spanning tree*)
→ jeder Knoten außer Root-Knoten hat **genau eine eintreffende Kante** (Dependent hat genau einen Kopf)
- **Graph-basierter Parser: MSTParser** (Baldrige und McDonald)
→ <http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html>

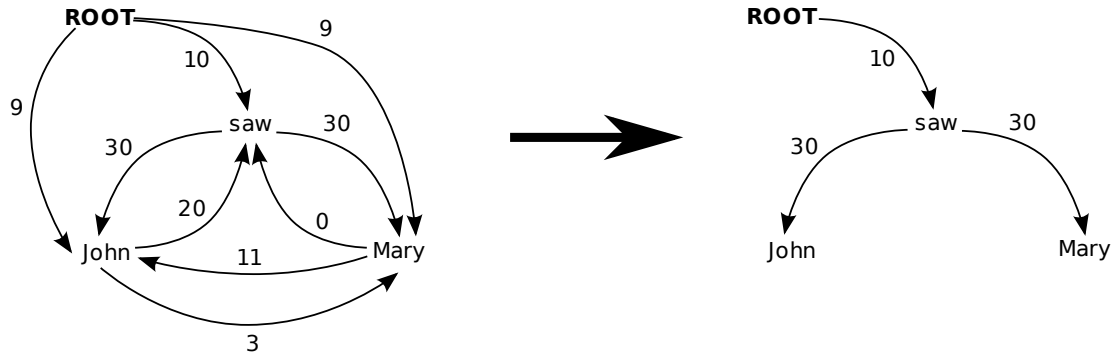


Abbildung 4: Graph-basiertes Parsing, <http://stp.lingfil.uu.se/~nivre/docs/eacl1.pdf>

Vorteile Graph-basiert

- **nicht-projektive Bäume** möglich
- Graphbasiert: **Bewertung ganzer Bäume** statt lokaler Entscheidungen basierend auf der jeweiligen Konfiguration
→ bessere *accuracy* mit **steigendem Abstand zwischen Kopf und Dependent**
- **ohne Backtracking** ist **nicht garantiert**, dass ein übergangsbasierter Parser die **optimale Lösung** findet
→ **Greedy-Algorithmus** basierend auf lokalen Entscheidungen

Evaluation von Dependenz-Parsing-Systemen

- **Überprüfung an Testmenge** (Teilmenge Dependency-Treebank)
- ***unlabeled attachment accuracy***: korrekte Zuweisung Dependent zu Kopf
- ***labeled attachment accuracy***: korrekte Zuweisung und korrekte Relation zwischen Dependent und Kopf

12.2.4 *Dependency-Treebanks*

- von Experten erstellte **dependenzsyntaktisch annotierte Korpora** (relationsannotierte Tokenlisten)
- Einsatz zu **Training und Evaluation** von Dependenz-Parsing-Systemen

- ***Dependency-Treebanks*** können auch aus **PSG-Treebanks** gewonnen werden (s. Sitzung 5)
 - **Transformation von kopfannotierten Konstituenten-Bäumen** in einen Dependenzgraph:
 1. **Finden aller *head-dependent*-Relationen** über *head-finding-rules*
 2. **Labeln der Relationen** über handgeschriebenen Regeln
 - Bestimmung Relationstyp **über Strukturposition**:
NP mit Mutterknoten S ist subj
 - bei Penn-Treebank: Verwendung **funktionaler Informationen in den Nichtterminalen**: NP-SBJ

Funktionale Kategorien in Penn Treebank

- **Grammatische Relationen/funktionale Angaben** in den phrasalen Kategorien, z. B.: NP-SBJ
 - PP-CLR: 'closely related', z. B. für präpositionales Objekt
 - NP-PUT: adverbiales Komplement von *put*
 - NP-ADV: für Kasusadverbial
 - <http://web.mit.edu/6.863/www/PennTreebankTags.html>

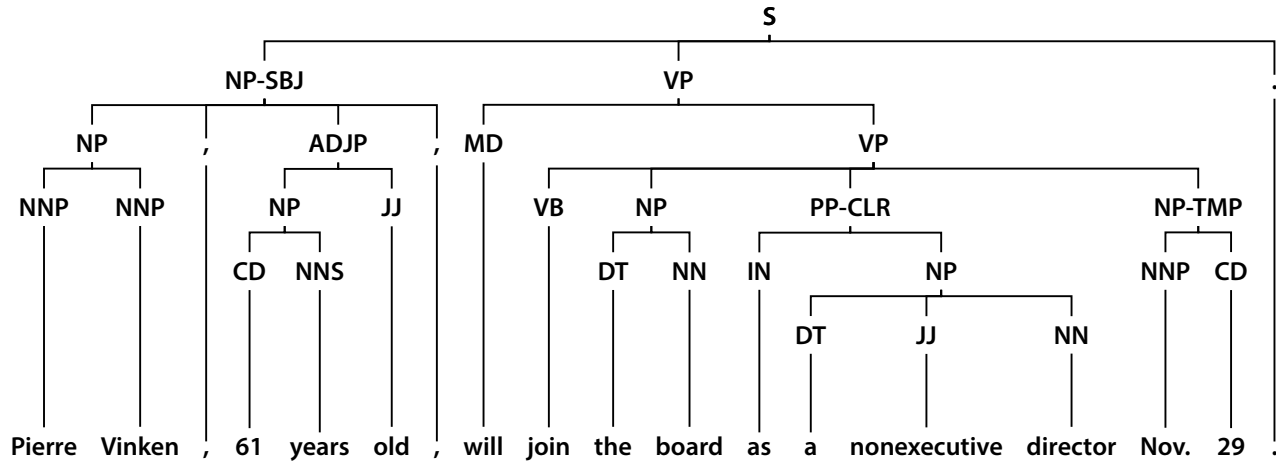


Abbildung 5: Beispiel-Parse Satz aus Penn-Treebank

Dependency Tagsets:

- **Stanford Dependencies:**

https://nlp.stanford.edu/software/dependencies_manual.pdf

- **Universal Dependencies** (basiert u.a. auf *Stanford Dependencies*; Stanford Parser verwendet in neuester Version UD-Tagset)

<http://universaldependencies.org/u/dep/all.html>

Dependency Treebanks:

- PDT = Prague Dependency Treebank (>1.5 Mill Tokens)
- **UD Treebanks** (> 30 Sprachen):

<http://universaldependencies.org/>

→ 2 Deutsche UD-Treebanks im CoNLL-Format:

<http://universaldependencies.org/de/index.html>

CoNLL-Dependency-Treebanks

- CoNLL = *Conference on Computational Natural Language Learning*
- Shared Tasks zu **Dependency Parsing**: mit annotierten Treebanks für Evaluation der Systeme (*training, dev und gold standard*)
- **TIGER Dependency Bank** (in Dependency-Format konvertiertes TIGER-Korpus, deutsch) verwendet in CoNLL und UD-Treebanks, konvertiert in Stanford bzw. Universal Dependencies

Dependenz-Formate:

- **CoNLL-Formate**

- in UD-Framework: CoNLL-U

- <http://universaldependencies.org/format.html>

- **dot-Format**

- Visualisierung mit graphviz (`display()` im NLTK)

- **NLTK: `DependencyGraph`-Objekt** (z. B. mit Stanford-Dep-Parser)

- **NLTK Ausgabemethoden für *DependencyGraph*:**

- `to_dot()`, `to_conll()`, `triples()`

Auflistung 5: CoNLL-Format (komplexer Satz aus dt. UD-Korpus (CoNLL-Format))

```

1  Alle  alle  PRON  PIS  Case=Nom|PronType=Tot  2  nsubj  _  _
2  wußten  wissen  VERB  VVFIN  Number=Plur|Person=3|VerbForm=Fin  0  root  _
SpaceAfter=No
3  ,  ,  PUNCT  $,  _  2  punct  _  _
4  daß  daß  SCONJ  KOUS  _  10  mark  _  _
5  uns  wir  PRON  PPER  Case=Dat|Number=Plur|Person=1|PronType=Prs  10  iobj
-  -
6  nicht  nicht  PART  PTKNEG  Polarity=Neg  7  advmod  _  _
7  mehr  mehr  ADV  ADV  _  10  advmod  _  _
8  viel  viel  ADJ  PIAT  Case=Nom|Degree=Pos|Gender=Neut|Number=Sing  9  amod
-  -
9  Zeit  Zeit  NOUN  NN  Case=Nom|Gender=Neut|Number=Sing  10  nsubj  _  _
10  blieb  bleiben  VERB  VVFIN  Number=Sing|Person=3|VerbForm=Fin  2  ccomp  _
SpaceAfter=No
11  .  .  PUNCT  $.  _  2  punct  _  _

```

Aufistung 6: NLTK: Umwandlung von CoNLL-Format zu dot-Format

```
1 #http://www.nltk.org/howto/dependency.html
2 #http://www.nltk.org/_modules/nltk/parse/dependencygr
3 from nltk import DependencyGraph
4 dg = DependencyGraph(treebank_data)
5 print(dg.to_dot())
6 # digraph G{
7 # 0 [label="0 (None)"]
8 # 0 → 2 [label="root"]
9 # 1 [label="1 (Alle)"]
10 # 2 [label="2 (wußten)"]
11 # 2 → 1 [label="nsubj"]
12 # 2 → 3 [label="punct"]
13 # 2 → 11 [label="punct"]
14 # 2 → 10 [label="ccomp"]
15 # 3 [label="3 (,)"]
```

```
16 # 4 [label="4 (daß)"]
17 # 5 [label="5 (uns)"]
18 # 6 [label="6 (nicht)"]
19 # 7 [label="7 (mehr)"]
20 # 7 → 6 [label="advmod"]
21 # 8 [label="8 (viel)"]
22 # 9 [label="9 (Zeit)"]
23 # 9 → 8 [label="amod"]
24 # 10 [label="10 (blieb)"]
25 # 10 → 4 [label="mark"]
26 # 10 → 5 [label="iobj"]
27 # 10 → 7 [label="advmod"]
28 # 10 → 9 [label="nsubj"]
29 # 11 [label="11 (.)"]
30 # }
```

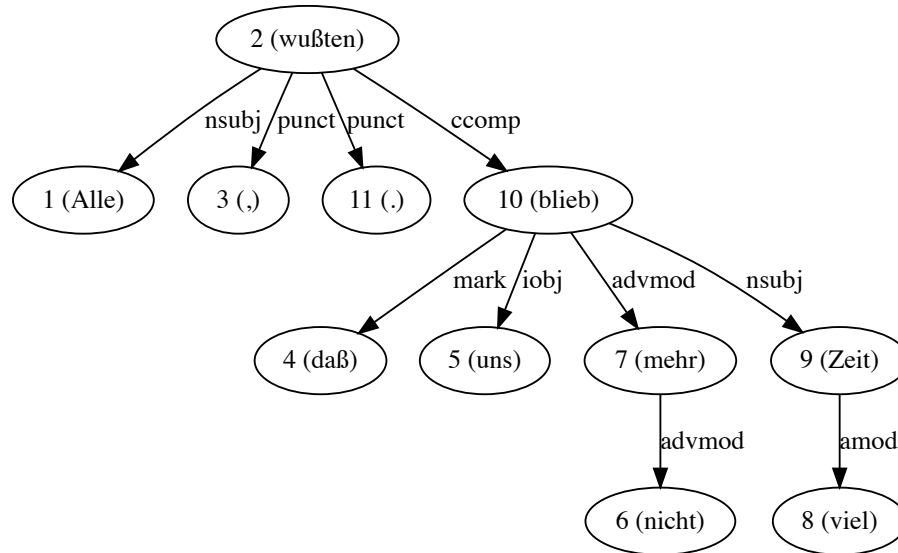


Abbildung 6: Visualisierung UD-German-Korpus (CoNLL) mit graphviz

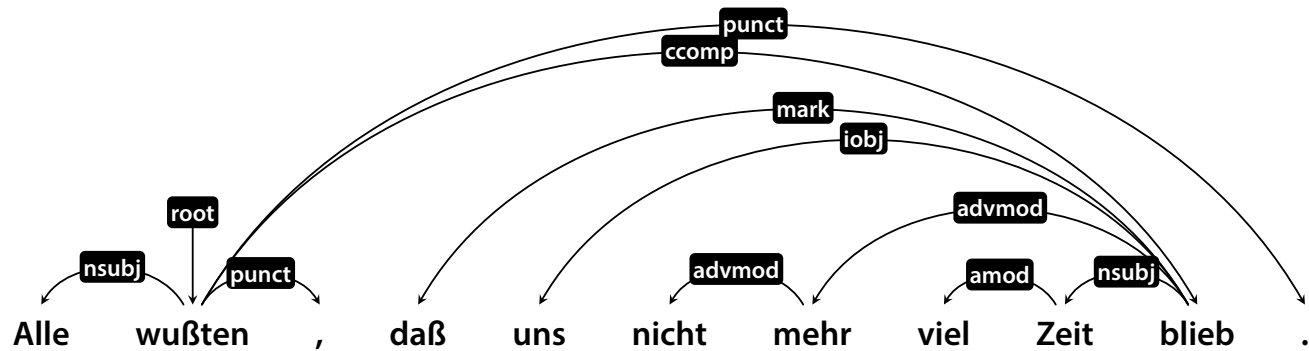


Abbildung 7: Satz aus Dependency-Treebank UD-german im LateX-tikz-dependency-Format