Syntax natürlicher Sprachen

5: Dependenzgrammatik und Dependency Parsing

A. Wisiorek

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

11.11.2025

1. Dependenzstruktur

1.1. Motivation

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

Warum Dependenzstrukturen?

- **Konstituentenstruktur (PSG)** — beschreibt den **Aufbau** von Sätzen aus Wörtern als Hierarchie von Phrasen.
- **Dependenzstruktur (DG)** - beschreibt die **Beziehungen** zwischen den Wörtern eines Satzes.
- **Ziel:** Erfassen, **welche Wörter das Auftreten oder die Form anderer Wörter bestimmen - also *lizensieren*.**

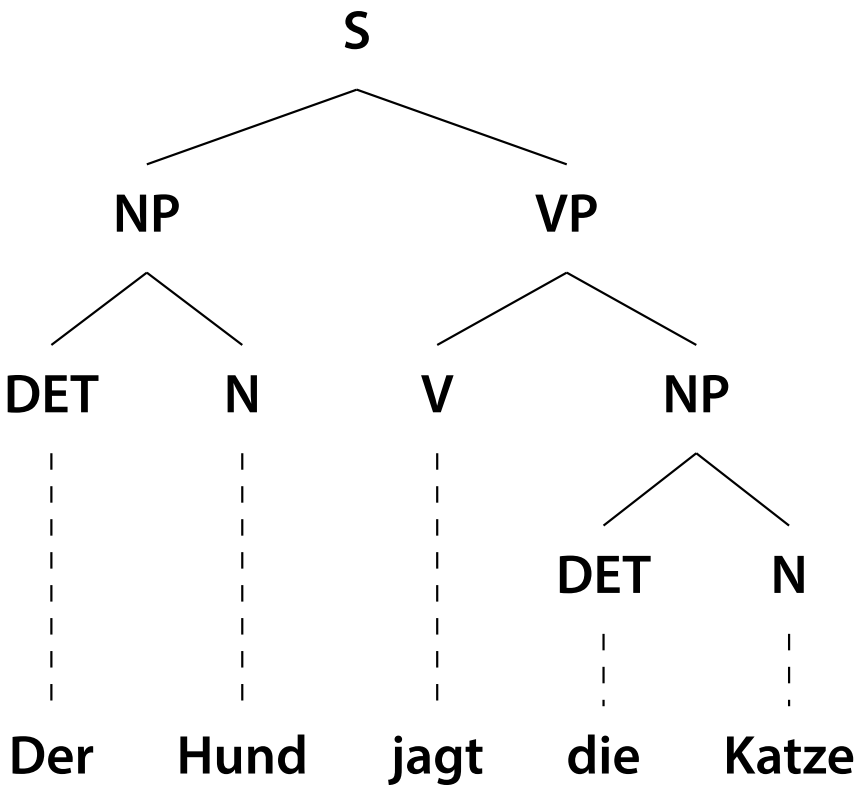
  → *Eine **Dependenz** ist eine gerichtete, gelabelte Kante zwischen **Kopf** und **Dependent**, die eine lokale **Lizenzierung/Selektion** ausdrückt.*

  → ***Universal Dependencies (UD)** bieten ein **sprachübergreifendes, standardisiertes Set von Dependenzrelationen und Labels**, das für zahlreiche Sprachen und NLP-Anwendungen verwendet wird.*

- **Dependenzgrammatik:** Eine formale, relationale Beschreibung von Syntax, die direkt mit semantischer Rollenstruktur korreliert und in CL praktisch nutzbar ist.

Konstituenten- vs Dependenzstruktur

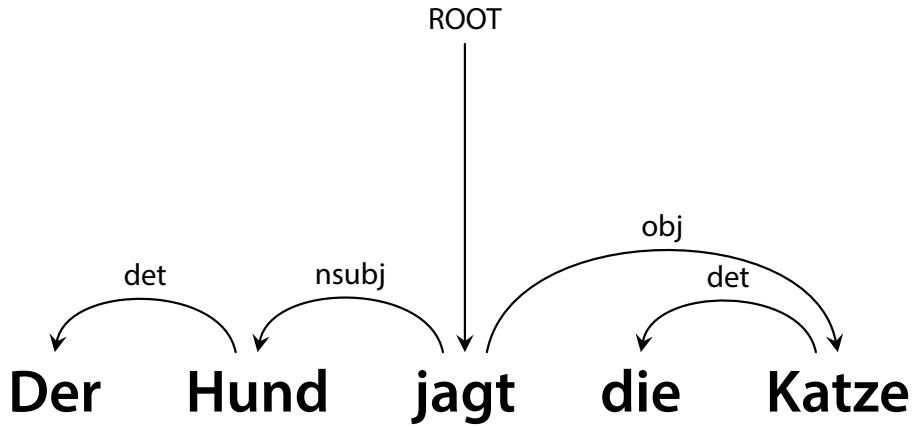
Konstituentenstruktur (PSG):



Baustein-Perspektive:

Welche Einheiten bilden Phrasen?

Dependenzstruktur (DG):



Beziehungs-Perspektive:

Wer hängt von wem ab?

Relevanz in Computerlinguistik

- direkter Zugriff auf **semantische Relationen** („wer tut was wem“)
  → *DG liefert direkt semantische Tripel: (Hund, jagt, Katze)*
- **sprachübergreifend** anwendbar (z. B. UD-Korpora)
  → *Nicht alle Sprachen bilden klare, phrasale Kategorien wie NP oder VP.*
- Grundlage für **Informationsextraktion, semantisches Parsing, maschinelle Übersetzung**
  → *Viele Anwendungen arbeiten intern auf Token-Ebene, nicht auf Phrasenebene.*

1.2. Eigenschaften der Dependenzstruktur

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

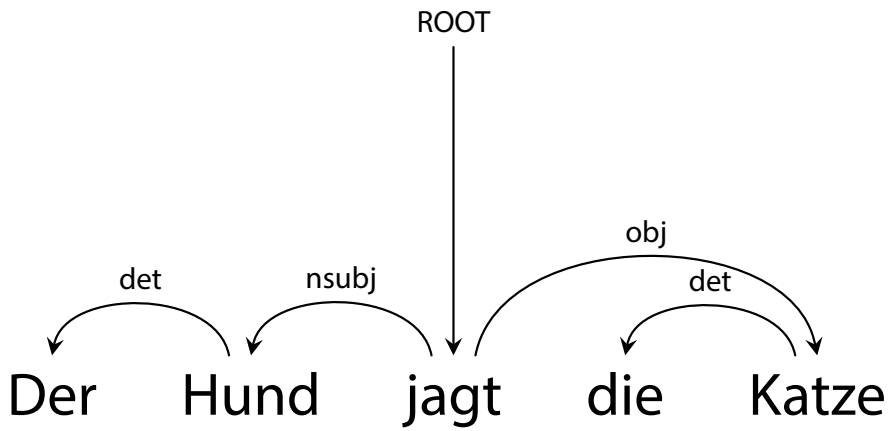
Was ist eine Dependenzstruktur?

- In der **Dependenzgrammatik (DG)** wird Syntax als ein Netz von **Abhängigkeitsrelationen zwischen Wörtern** beschrieben – nicht als Phrasenhierarchie.

- Eine **Dependenz** ist eine **gerichtete, asymmetrische Relation** zwischen zwei Wörtern:
  - **Kopf** = das lizenzierende Wort (bestimmt Vorkommen oder Form des anderen)
  - **Dependent** = das lizenzierte Wort (abhängig vom Kopf)

- Eine **Dependenzstruktur** ist die **Gesamtmenge solcher Relationen** in einem Satz.
- **Formale Sicht:** ein gerichteter Baum / Graph
  $\rightarrow G = \langle M, R \rangle$ *mit M = Menge der Wörter, R = Dependenzrelationen (geordnete Paare).*

Beispiel: Eine einfache Dependenzstruktur

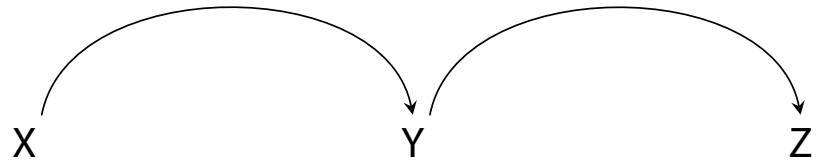


Interpretation

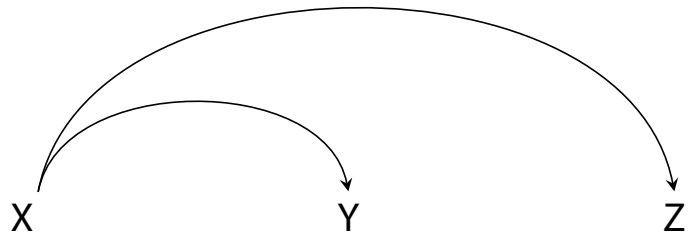
- **„jagt“** ist der **Kopf** des Satzes – das Verb als **Wurzelknoten**.
- **„Hund“** und **„Katze“** sind abhängige Argumente des Verbs.
- **„Der“** und **„die“** hängen als **Determinierer** von den Nomen ab.
- Die Kanten sind **gerichtet** und **gelabelt** (`nsubj`, `obj`, `det`).

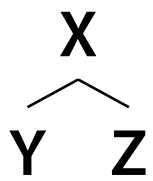
Eigenschaften der Dependenzstruktur

- **binär** – jede Relation verbindet genau zwei Wörter (Kopf–Dependent)
- **asymmetrisch** – wenn Y von X abhängt, ist X nicht von Y abhängig
- **genau *ein* Kopf pro Dependent** – aber mehrere Dependenten pro Kopf möglich
- **genau *ein* Wurzelknoten** (typischerweise das Verb) - alle anderen Wörter hängen direkt oder indirekt davon ab.
- **unmittelbare vs. mittelbare Abhängigkeit** – direkte vs. vermittelte Relationen



- **Abstraktion von Lineare Ordnung** - Dependenzstruktur ist unabhängig von der Wortstellung, daher kann die Struktur später auch nicht-projektiv sein.





Darstellung durch Dependenzbaum (Stemma)

Die Richtung der Dependenz wird durch die hierarchische Baumstruktur angezeigt (Parent–Child-Relation statt Pfeilrichtung), also implizit durch die vertikale Anordnung.

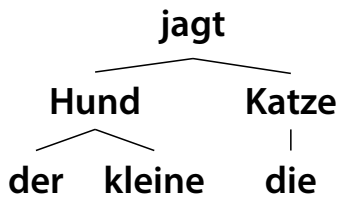


Abbildung: Einfacher Dependenzbaum

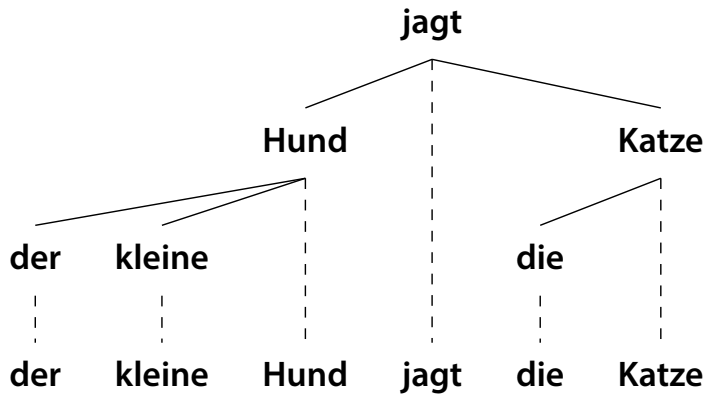
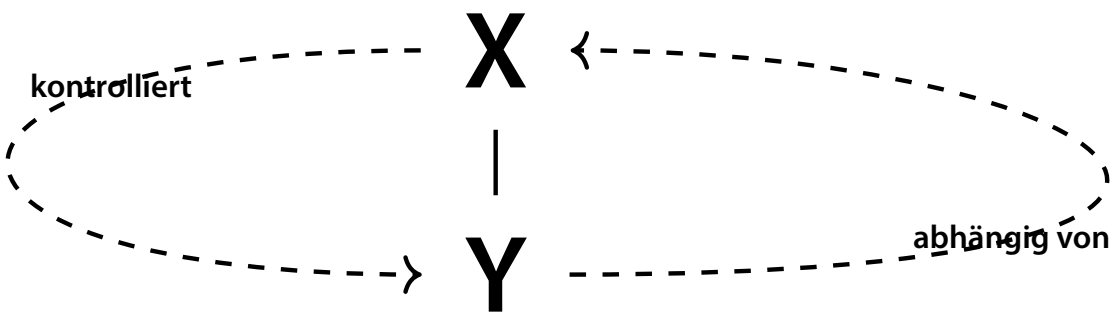


Abbildung: Dependenzbaum mit Berücksichtigung der linearen Ordnung

Richtung der Dependenzrelation

- In einer Dependenzstruktur ist die Relation **gerichtet**: sie verläuft vom **Kopf** zum **Dependenten**.
- Der Pfeil steht für die **Lizenzierungsrichtung**: der Kopf **lizenziert** oder **bestimmt** das Auftreten oder die Form des Dependents.
- **Dependenz** und **Kontrolle** sind zwei Perspektiven auf dieselbe Relation:
  - **Dependenz:** das abhängige Element (Dependent) hängt vom Kopf ab.
  - **Kontrolle:** der Kopf steuert oder bestimmt seinen Dependenten.
- Formal wird diese Relation als Pfeil **vom Kopf zum Dependenten** notiert.



1.3. Konstituenten- vs. Dependenzstruktur

Konstituentenstruktur vs. Dependenzstruktur

- **Konstituentenstruktur (PSG):**
  - beschreibt den **Aufbau** eines Satzes aus hierarchisch geordneten Phrasen (Bausteinperspektive).
  - Strukturinformation steckt in den **Knoten** (Kategorien: NP, VP, PP, …).
- **Dependenzstruktur (DG):**
  - beschreibt die **Beziehungen** zwischen den **Wörtern** (Beziehungsperspektive).
  - Strukturinformation steckt in den **Kanten** (Relationen: Subjekt, Objekt, Determinierer …).
- **DG** ist für Computerlinguistik besonders geeignet:
  $\rightarrow$ *flachere Strukturen, unmittelbarer Zugriff auf Wortrelationen, sprachübergreifend (UD).*

Konstituentenstruktur = Phrasenstrukturgrammatik (PSG)

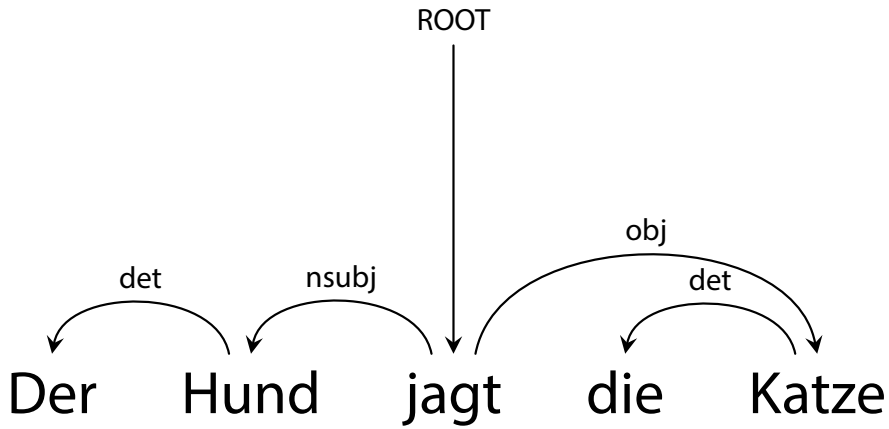
- Analyse des **Aufbaus der Satzstruktur durch Zergliederung** in Konstituenten
- Zusammensetzung von Wörtern zu **syntaktischen Einheiten**
- **Subjekt-Prädikat**-Grundstruktur
- **Strukturinformation in Knoten** (Kategorien des strukturellen Aufbaus)
- **phrasale Knoten**

Dependenzstruktur = relationale Wortgrammatik

- Analyse **Satzstruktur 'von innen heraus'** (vom Verb ausgehend)
- **Beziehung zwischen Wörtern**
- **Subjekt und Objekt gleichrangige Argumente des Verbs** (beide valenzgefordert)
- **Strukturinformation in Kanten** (relationale Kategorien)
  $\rightarrow$ ***Label einer Kante = syntaktische Funktion des Dependenten!***
- keine phrasalen Knoten, **flachere Struktur** als PSG

Übersicht Dependenzstruktur

- **Elemente der Struktur (Knoten)** $\rightarrow$ *Wörter*
- **Relationen der Struktur (Kanten)** $\rightarrow$ *Dependenzrelationen (z. B. Subjekt, Objekt)*
- **syntaktische Kategorien** $\rightarrow$ *gerichtete Kanten = Dependenzrelationen*
- **Kategorientyp** $\rightarrow$ *funktional / relational*
- **Strukturinformationen in Kanten des Syntaxbaums (funktionale Kategorien)**



Dependenz- vs Konstituentenstruktur: Beispielsatz

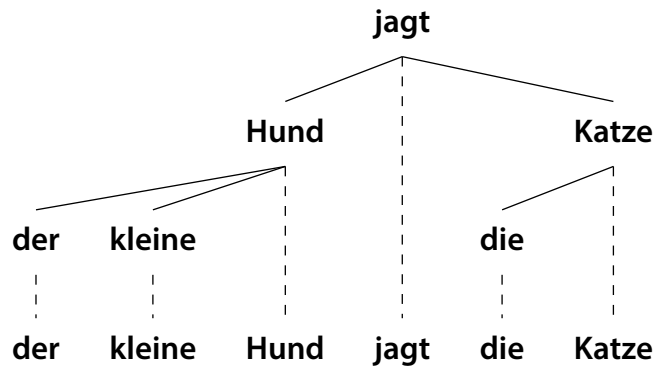


Abbildung: Dependenzstruktur mit projizierter Wortfolge

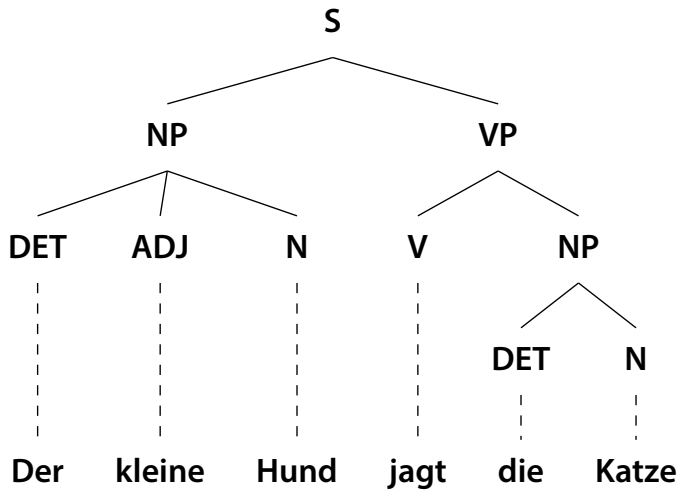


Abbildung: Konstituentenstruktur

Beziehung zwischen Dependenz- und Konstituentenstruktur

- **Dependenz in der Konstituentenstruktur (PSG)**
  - Implizite Dependenzanalyse in den **Phrasenkategorien** über das **Kopfprinzip**.
    $\rightarrow$ *Der Phrasenkopf regiert seine Schwesterknoten (z. B. N in NP).*
- **Konstituenz in der Dependenzstruktur (DG)**
  - Implizite Konstituentenstruktur durch **Teilbäume** der Dependenzstruktur.
    $\rightarrow$ *Jeder Teilbaum kann als Konstituente interpretiert werden.*
  - Nicht alle Konstituenten der PSG werden explizit repräsentiert (z. B. keine eigenständige **VP**).

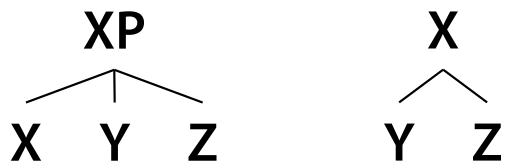


Abbildung: Konstituenten- und Dependenzstrukturschema (X ist Kopf)

Komplement und Adjunkt in PSG und DG

Komplemente bzw. **Adjunkte** der X-Bar-Theorie, die über ihre **Strukturposition** (kopfnahe vs. adjointe Position) definiert sind, entsprechen **Ergänzungen** und **Angaben/Attributen** in der Dependenztheorie.

- In der **X-Bar-Theorie (PSG)**:
  - **Komplement** = Schwester des Kopfes (Kopfnahe Komplement-Position)
  - **Adjunkt** = höher adjointe Phrase (rekursiv angehängte Adjunkt-Position)
- In der **Dependenzgrammatik (DG)**:
  - **Komplement (Ergänzung)** = **valenzgeforderter Dependent** des Kopfes $\rightarrow$ *Lizenzierung über Valenz* $\rightarrow$ *obligatorisch oder fakultativ*
  - **Adjunkt** (**Angabe** / **Attribut**) = **nicht-valenzgeforderter Dependent** $\rightarrow$ *Lizenzierung optional, Modifikator*
- Die Unterscheidung bleibt also dieselbe — aber sie wird **nicht über Position**, sondern über **Lizenzierung** beschrieben.

Vorteile von Dependenzmodellen

- **Relationale Informationen direkt** vorhanden statt indirekt über Position in Strukturbaum
  $\rightarrow$ *Verwendung z. B. für* ***Informationsextraktion und semantisches Parsing***
- **Wortgrammatik** = direkte Modellierung von Relation zwischen Wörtern
  $\rightarrow$ ***keine Lexikalisierung notwendig***
- **Dependenzgrammatik als Wortgrammatik**
  $\Rightarrow$ **reduziert *sparse data*-Problem bei Parameterabschätzung**

1.5. Transformation Konstituenten- in Dependenzstruktur

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

Von der Phrasenstruktur zur Dependenzstruktur

- Die Umwandlung einer PSG-Analyse in eine DG-Analyse erfolgt über sogenannte **Head-Finding-Rules**.
- Dabei wird für jede Phrase bestimmt, welches Element ihr **Kopf** ist.
- **Beispielhafte Regeln:**
  - head(NP) = N
  - head(VP) = V
  - head(PP) = NP *(gemäß UD)* oder P (traditionell)
  - head(S) = VP
- Nach Identifikation der Köpfe:
  1. Die Köpfe der Ko-Konstituenten werden Dependenten ihres Phrasenkopfs.
  2. Jede Dependenz erhält ein Funktionslabel (z. B. `nsubj`, `obj`, `det`).

Ergebnis

- Aus einem hierarchischen Phrasenbaum wird ein gerichteter, gelabelter **Dependenzbaum**.
- **PSG $\rightarrow$ DG = Positionsinformation $\rightarrow$ Relationeninformation.**

von Phrasenstrukturbaum zu Dependenzbaum

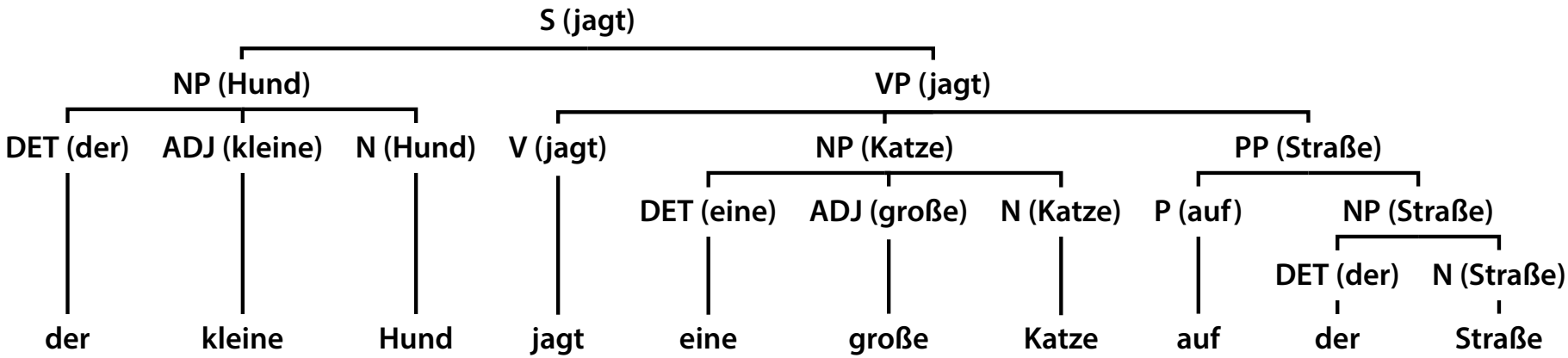


Abbildung: Phrasenstruktur mit Kopfannotation

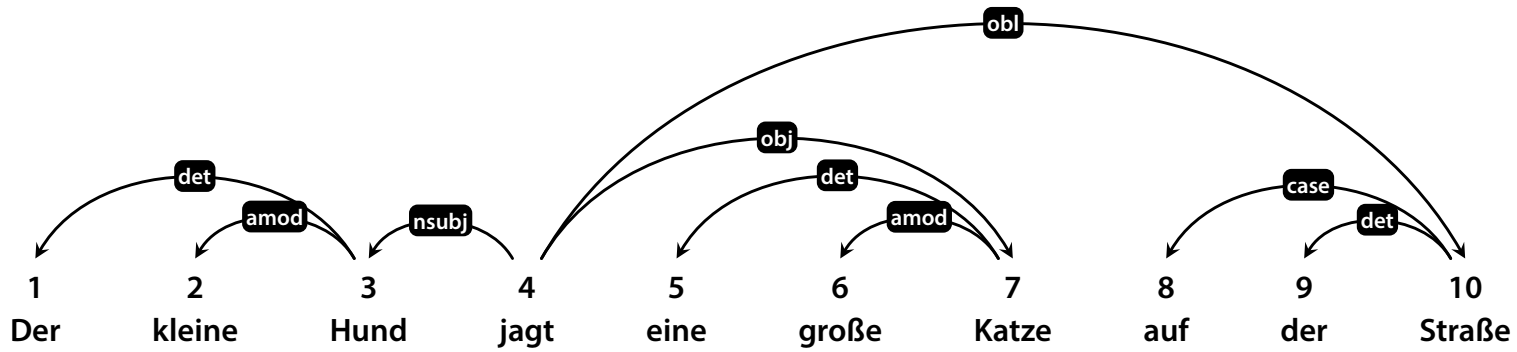
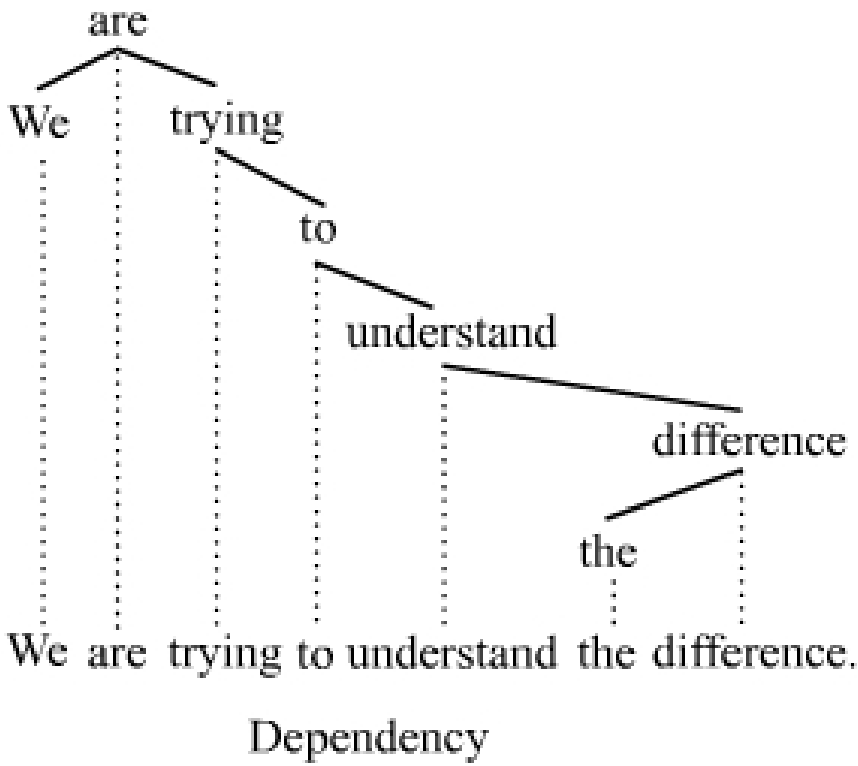
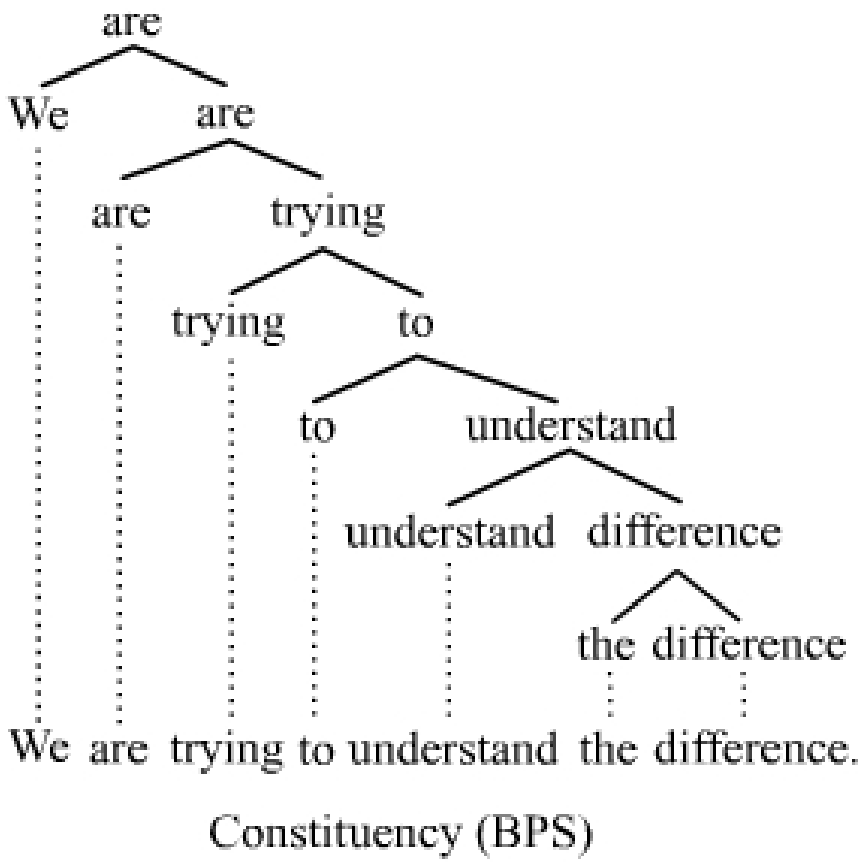


Abbildung: daraus abgeleitete Dependenzstruktur

Vergleich Syntaxbäume mit Kopfangaben



Dependency



Constituency (BPS)

Abbildung: Geordneter Dependenzbaum - Konstituentenbaum (von Tjo3ya - eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17517283)

2. Dependenzgrammatik

2.1. Eigenschaften der Dependenzgrammatik

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

Was ist eine Dependenzgrammatik?

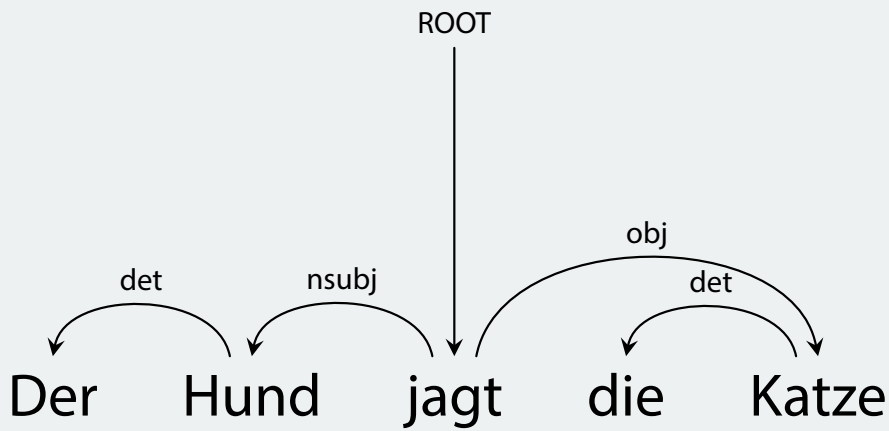
- Die **Dependenzgrammatik (DG)** ist ein formales Modell der Satzstruktur, das von **Wortabhängigkeiten** (Dependenzrelationen) statt von **Phrasenregeln** ausgeht.
- **Grundidee:** Syntax = Menge von **gerichteten, gelabelten Kanten** zwischen Wörtern.
- Eine DG beschreibt:
  - welche Wörter (Köpfe) welche anderen Wörter (Dependents) lizensieren,
  - welche **Relationstypen** (Labels) dabei auftreten (gelabelte DG),
  - und welche Strukturen dadurch in Sätzen erlaubt sind.
    $\rightarrow$ *HINWEIS: In NLTK nur ungelabelte DGs möglich!*
- Ziel: **Formale Repräsentation** der Dependenzstruktur, aus der sich die Wohlgeformtheit von Sätzen ableiten lässt.

Dependenzregeln als Grammatik

- Analog zu Phrasenstrukturregeln kann auch eine Dependenzgrammatik als **Menge von Regeln** formuliert werden, die Kopf–Dependent-Relationen beschreiben.

Beispiel: einfache Dependenzregeln

- `jagt` $\rightarrow$ `Hund` (Subjekt)
- `jagt` $\rightarrow$ `Katze` (Objekt)
- `Hund` $\rightarrow$ `der` (Det.)
- `Katze` $\rightarrow$ `die` (Det.)



- Jede Regel beschreibt eine **zulässige Abhängigkeit** in der Sprache. Die Gesamtheit dieser Regeln lizenziert alle wohlgeformten Strukturen.
- **DG als Baum:** Alle Wörter eines Satzes sind über Kanten mit dem Verb als Wurzel verbunden.

DG als gerichteter Graph

- Eine Dependenzstruktur kann formal als **gerichteter Baum / Graph** beschrieben werden (siehe oben):

  $\rightarrow G = \langle M, R \rangle$ *mit*
  M = Menge der Wörter (Knoten)
  R = Menge der Dependenzrelationen (gerichtete, gelabelte Kanten = geordnete Paare).

- **Formale DG = Menge von geordneten Paaren** $(x, y) \in R$, die definieren, welche Dependenzen erlaubt sind.
- **Wohlgeformtheit:** Ein Satz ist genau dann wohlgeformt, wenn er durch diese Regeln eine gültige Dependenzstruktur aufweist.
- **Eigenschaften gültiger Dependenzstruktur (siehe oben):**
  - genau ein **Wurzelknoten (ROOT)**: hängt von keinem anderen ab
  - jedes Wort hat **maximal einen Kopf** (Ein-Kopf-Beschränkung)
  - Kanten können gelabelt sein (Subjekt, Objekt, Determinierer, …)

Phrasenstrukturgrammatik vs. Dependenzgrammatik

Phrasenstrukturgrammatik (CFG)

- beschreibt **hierarchischen Aufbau** von Sätzen aus Phrasen
- Regeln kombinieren **Kategorien** (Nichtterminale)
- Strukturinformation steckt in den **Knoten**
- typische Regel:

$$S \rightarrow NP\ VP$$
$$VP \rightarrow V\ NP$$

- Repräsentation: **Phrasenbaum**

Dependenzgrammatik (DG)

- beschreibt **Relationen zwischen Wörtern**
- Regeln kombinieren **lexikalische Elemente**
- Strukturinformation steckt in den **Kanten**
- typische Regel:

$$jagt \rightarrow Hund$$
$$jagt \rightarrow Katze$$

- Repräsentation: **Dependenzbaum**

Dependenzgrammatik in NLTK

```
1  sent= 'der Mann schenkt der Frau das Buch'
2
3  grammar = nltk.DependencyGrammar.fromstring("""
4  'gibt' -> 'Mann' | 'Frau' | 'Buch'
5  'schenkt' -> 'Mann' | 'Frau' | 'Buch'
6  'Mann' -> 'der'
7  'Frau' -> 'der' | 'die'
8  'Buch' -> 'das'
9  """)
10
11 parser = nltk.ProjectiveDependencyParser(grammar)
12 for tree in parser.parse(sent.split()):
13     print(tree)
14
15 #(schenkt (Mann der) (Frau der) (Buch das))
```

2.2. Ambiguität und Projektivität

Ambiguität in der Dependenzgrammatik

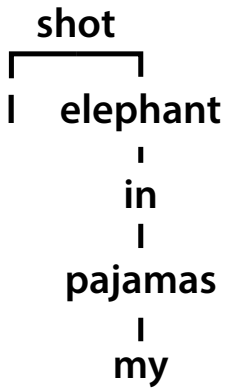
- Auch in Dependenzstrukturen kann es **mehrdeutige Analysen** geben: Ein Satz kann mehrere gültige Dependenzbäume haben.
- **Ursache:** ein Wort oder eine Phrase kann sich **mehreren Köpfen anschließen**.
- **Klassischer Fall:** Präpositionalphrasen-Ambiguität (PP-Attachment)

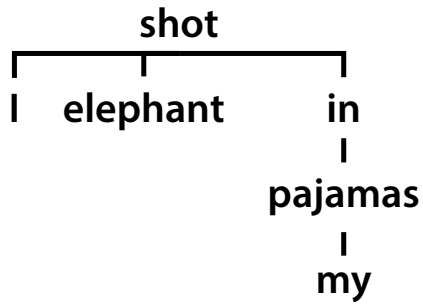
Er sah den Mann mit dem Fernglas.

- Mögliche Interpretationen:
  1. **Instrumental:** „Er sah (den Mann) **mit dem Fernglas**."
  2. **Attributiv:** „Er sah (**den Mann mit dem Fernglas**)."
- In der DG zeigt sich Ambiguität durch **unterschiedliche Kantenstrukturen.**

PP-Attachment-Ambiguität in Dependenzgrammatik

```
1 sent= 'I shot an elephant in my pajamas'
2
3 grammar = nltk.DependencyGrammar.fromstring("""
4 'shot' -> 'I' | 'elephant' | 'in'
5 'elephant' -> 'an' | 'in'
6 'in' -> 'pajamas'
7 'pajamas' -> 'my'
8 """)
```





Projektivität und Nicht-Projektivität

- **Projektive Struktur:** Alle Kanten können gezeichnet werden, ohne sich zu kreuzen. Zwischen Kopf und Dependent liegt nur Material, das vom Kopf abhängt.
- **Nicht-projektive Struktur:** Eine oder mehrere Kanten **überkreuzen sich** $\rightarrow$ zeigt Diskontinuität.
- **Ursachen:**
  - freie Wortstellung (z. B. im Deutschen)
  - Relativsätze, Topikalisierung, Extraposition

Projektivität und Nichtprojektivität

- **projektive Struktur:** nur **projektive** Kanten, d. h. es gibt einen Pfad vom Kopf der Relation **zu jedem Wort** zwischen Kopf und Dependent
- **nicht-projektive Struktur = Überschneidung von Kanten**
  $\rightarrow$ *c) und e) in folgendem Beispiel enthalten eine nicht-projektive Kante:*

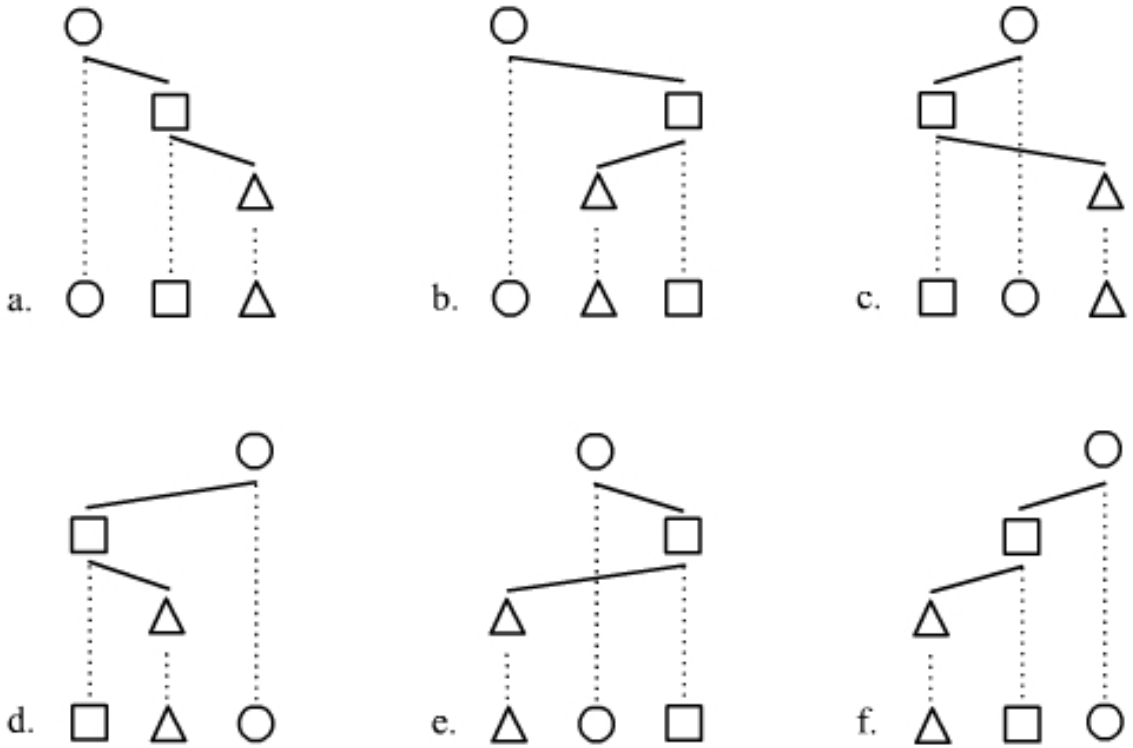
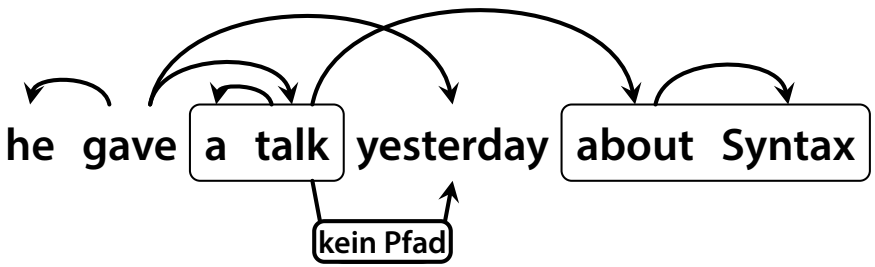


Abbildung:
`https://en.wikipedia.org/wiki/Discontinuity_(linguistics)`

Beispiel: nicht-projektive Struktur



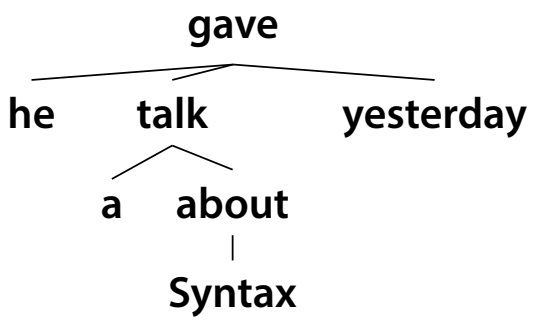


Abbildung: Dependenzanalyse diskontinuierlicher = nicht-projektiver Struktur (mit und ohne Berücksichtigung linearer Ordnung)

- **linguistisch: nicht-projektive Strukturen** entstehen durch **diskontinuierliche Elemente**
  $\rightarrow$ ***freie Wortstellung und long distance dependencies***

Warum Projektivität wichtig ist

- **Sprachlich:**
  - Projektive Strukturen = kontinuierliche Satzgliederung
  - Nicht-projektive Strukturen = diskontinuierliche Elemente
- **Formale Relevanz:**
  - Projektive Bäume sind **kontextfrei modellierbar** (≈ CFG-äquivalent)
  - Nicht-projektive Strukturen erfordern **komplexere Modelle**
- **CL-Bedeutung:**
  - Viele Parsing-Algorithmen (z. B. übergangsbasierte) akzeptieren nur projektive Bäume
  - Moderne neuronale Parser (z.B. graph-basierte) können auch nicht-projektive Strukturen erzeugen

- **Fazit:** Projektivität ist das Bindeglied zwischen linguistischer Theorie (Strukturkontinuität) und Parsing-Komplexität.

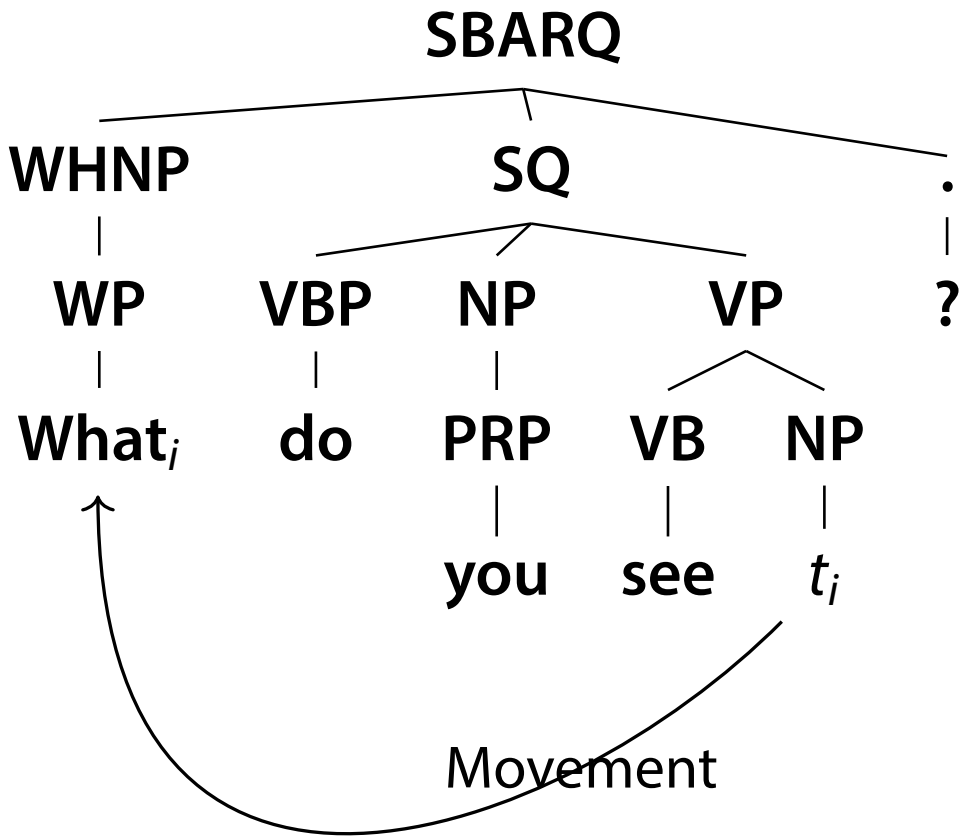
Vorteil Dependenzgrammatik

- 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*
- bei **Ableitung Dependenzgrammatik** von PSG-Treebanks durch *head-finding-rules* ergeben sich **automatisch projektive** Strukturen

Exkurs: Diskontinuierliche Konstituenten

- auch: ***long distance dependencies***
- Heraustrennung von Teilkonstituenten einer Konstituente
- Problem für Baumdarstellung:
  $\rightarrow$ ***Überkreuzung*** = *nicht-projektiv*
- Lösung: **leere Knoten** (*empty nodes*: `0`, ϵ, `t`, `NONE`)
  $\rightarrow$ *trace (**Spur**): Konzept der Transformationsgrammatik*
- **Transformationsgrammatik**:
  $\rightarrow$ *Annahme: **Tiefen- und Oberflächenstruktur***
  $\rightarrow$ *abstrakte vs. beobachtbare Form von Sätzen*
  $\rightarrow$ *z. B.: Annahme deutsche Tiefenstruktur der VP: OV (den Hund sehen)*
  $\rightarrow$ ***Transformationsregelanwendung** zur Erzeugung der Oberflächenstruktur: **läßt Spur zurück***
- im Englischen relativ begrenzt: z.B. Topikalisierung, Extraposition, **Wh-fronting**

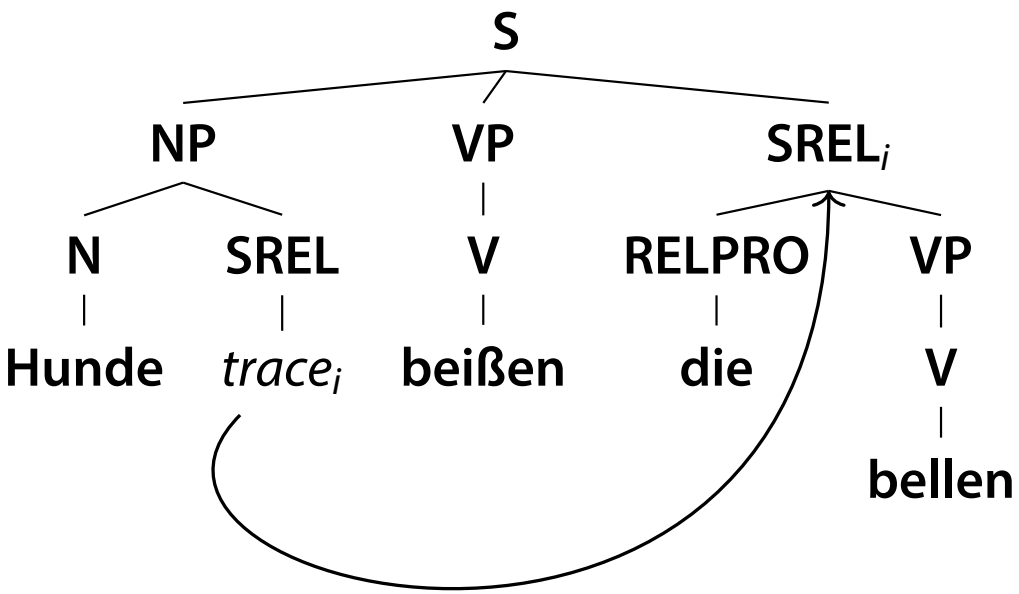
Analyse *long distance dependencies* mit Spur (t)



Nicht-projektive Strukturen: CFG

- diskontinuierliche Strukturen entstehen durch Herausbewegung von Phrasen/Konstituenten

- mit CFGs: nicht direkt modellierbar (überkreuzende Kanten)
- nur mit *traces* (Leerstelle + Movement/Transformationsregeln)

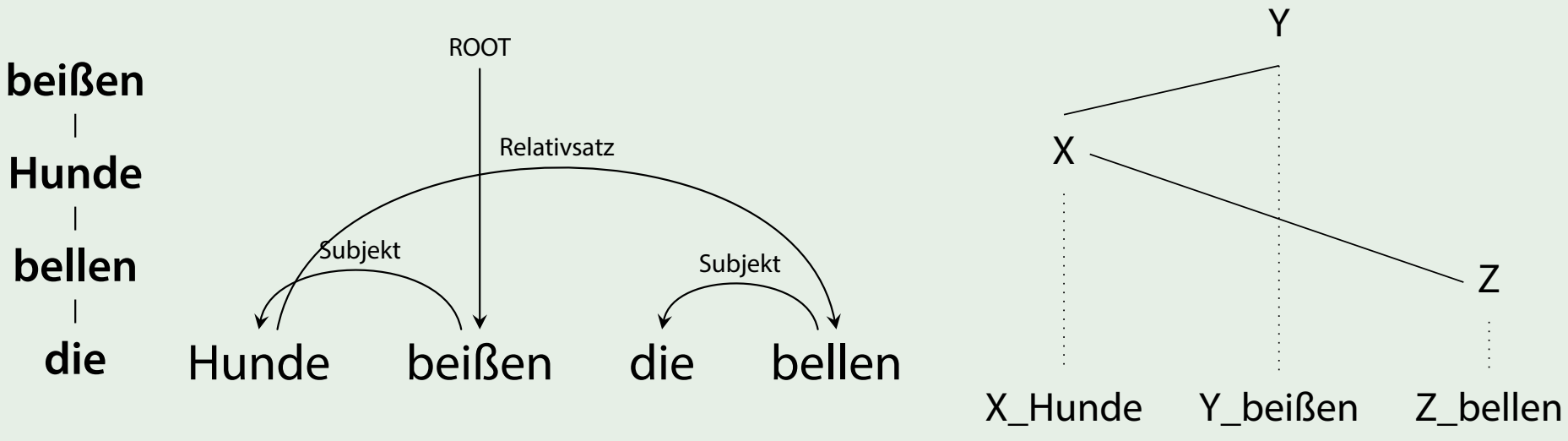


Nicht-projektive Strukturen: Dependenzgrammatik

- Dependenzgrammatik: direkte Modellierung nicht-projektiver Strukturen, da unabhängig von linearer Anordnung

Beispiel Relativsatz

- Dependent eines Wortes folgt nach dessen Kopf, vgl c) oben



- aber: bestimmte Dependency Parsingalgorithmen können nicht-projektive Strukturen nicht verarbeiten!

3. Dependency Parsing

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

3.1. Dependenzbasierte Modelle

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

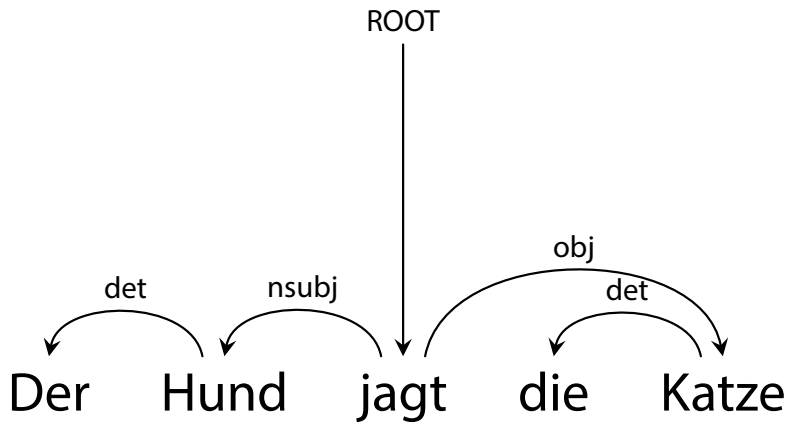
Was ist Dependency Parsing?

- **Ziel:** automatische Erzeugung der **Dependenzstruktur** eines Satzes.
- Eingabe: Wortfolge (z. B. Tokenisierung, POS-Tags, Lemmata)
- Ausgabe: gerichteter, gelabelter Baum mit Kopf–Dependent-Relationen.

EINGABE:

Der Hund jagt die Katze.

AUSGABE:



- Parser berechnet diese Struktur auf Basis einer **Grammatik oder eines trainierten Modells**.

Übergangsbasierte Parser

- Idee: Satz wird schrittweise in eine Dependenzstruktur überführt
- Parser verwaltet drei Datenstrukturen:
  - **Stack** – enthält bereits verarbeitete Wörter
  - **Buffer** – enthält verbleibende Wörter
  - **Relationenmenge** – enthält erzeugte Kanten
- Operationen:
  - `SHIFT`: nächstes Wort vom Buffer auf den Stack legen
  - `LEFTARC`: oberstes Stack-Element wird Dependent des nächsten
  - `RIGHTARC`: oberstes Stack-Element wird Kopf des nächsten

- Modell wählt die passende Operation mit einem **Klassifikator** (z. B. neuronales Netz).
- Beispielsysteme: **MaltParser (Nivre)**, **spaCy**, **Stanford Transition Parser**.

Graphbasierte Parser

- Satz $\rightarrow$ vollständiger Graph aller möglichen Kopf–Dependent-Paare.
- Jede potenzielle Kante erhält ein Gewicht $w(x, y)$, das ihre Wahrscheinlichkeit ausdrückt.
- Parser sucht den **maximal gewichteten Baum** (Maximum Spanning Tree, MST).

- Vorteile:
  - Berücksichtigt **globale Struktur** des Satzes
  - Kann **nicht-projektive Strukturen** verarbeiten
- Nachteile:
  - höhere Rechenkomplexität
  - keine inkrementelle Verarbeitung

- Typische Implementationen: **MSTParser**, **biaffine neural parsers** (z. B. Dozat & Manning 2017).

Evaluation und Bedeutung

- **Evaluation:**
  - **UAS (Unlabeled Attachment Score):** Kopf richtig?
  - **LAS (Labeled Attachment Score):** Kopf und Label richtig?
  - berechnet auf Basis goldannotierter Treebanks (z. B. UD)
- **Anwendungsgebiete:**
  - semantisches Parsing, Relationsextraktion, maschinelle Übersetzung
  - syntaktische Features für neuronale Sprachmodelle
- Moderne Systeme erreichen über 95 % UAS auf Standardsprachen.

3.2. Übergangsbasierter Shift-Reduce-Dependency-Parser

1. Dependenzstruktur
   - Motivation
   - Eigenschaften der Dependenzstruktur
   - Konstituenten- vs. Dependenzstruktur
   - Transformation Konstituenten- in Dependenzstruktur
2. Dependenzgrammatik
   - Eigenschaften der Dependenzgrammatik
   - Ambiguität und Projektivität
3. Dependency Parsing
   - Dependenzbasierte Modelle
   - Übergangsbasierter Shift-Reduce-Dependency-Parser
4. Dependency-Treebanks

Übergangsbasiertes Dependenz-Parsing

Grundprinzip

- Parsing erfolgt als **sequenzieller Aufbau** der Dependenzstruktur.
- Parser arbeitet als **Shift–Reduce-System** mit einer internen Konfiguration aus:
  - **Stack** – bereits verarbeitete Wörter
  - **Buffer** – verbleibende Wörter
  - **Relationenmenge** – bisher erzeugte Abhängigkeiten
- In jedem Schritt wählt der Parser den nächsten **Übergang** (`SHIFT`, `LEFTARC`, `RIGHTARC`) **über einen Klassifikator**.
- **Klassifikator:** bildet aktuelle Konfigurationen auf mögliche Übergänge ab.
- **Training:** erfolgt auf Basis annotierter **Dependency-Treebanks**.

Shift–Reduce-Schritte

SHIFT

- Das nächste Wort aus dem **Buffer** wird auf den **Stack** gelegt.
- Der Stack wird mit einem speziellen `ROOT`-Knoten **initialisiert**.
- Der Parsing-Prozess ist **abgeschlossen**, wenn der Buffer leer ist und nur noch `ROOT` auf dem Stack liegt.

REDUCE

- Statt einer Ersetzung durch ein Nonterminal (wie bei CFG-Parsing):
  $\Rightarrow$ **Hinzufügen einer Dependenzrelation** zwischen den obersten Stack-Elementen (Kopf–Dependent)
  $\Rightarrow$ **Entfernen des Dependents** vom Stack

Hinweis

Die **REDUCE**-Entscheidung erfolgt auf Basis einer **Grammatik oder eines trainierten Modells** (z. B. Syntaxbäume als **Oracle** für das Training).

REDUCE-Operationen

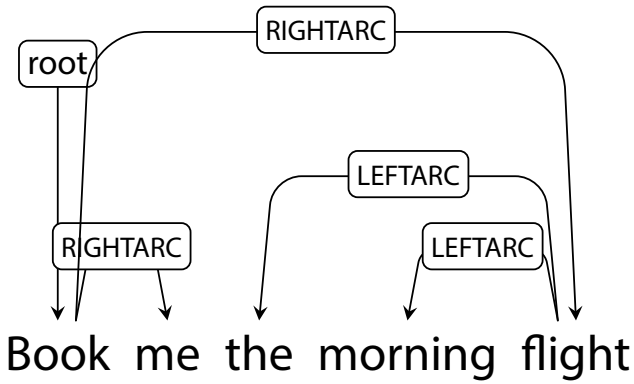
LEFTARC und RIGHTARC

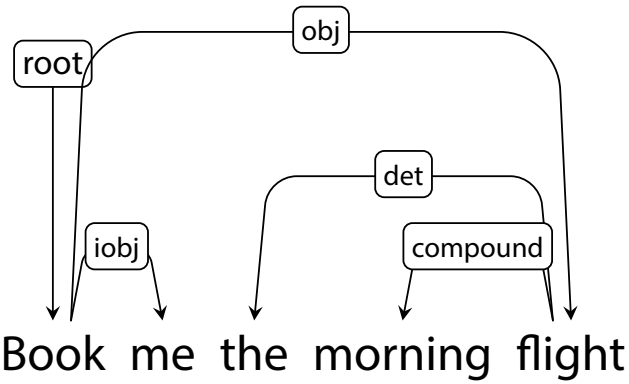
- Beim REDUCE-Schritt können zwei Arten von Relationen erzeugt werden – je nach Position des Kopfes:
  - LEFTARC – Kopf steht **rechts**: *the* $\leftarrow$ *flights*
  - RIGHTARC – Kopf steht **links**: *book* $\rightarrow$ *me*
- Jede Operation erzeugt eine gerichtete Kante (Head–Dependent) und entfernt den Dependent vom Stack.

Einschränkung bei RIGHTARC

- Eine RIGHTARC-Operation ist nur erlaubt, wenn der potentielle Dependent **nicht selbst noch Kopf einer offenen Relation** ist.
- Diese Einschränkung verhindert, dass ein Wort **zu früh vom Stack entfernt** wird.
- Im Gegensatz dazu ist LEFTARC immer möglich – sie erzeugt **nur projektive Strukturen** (vgl. nächste Folien).

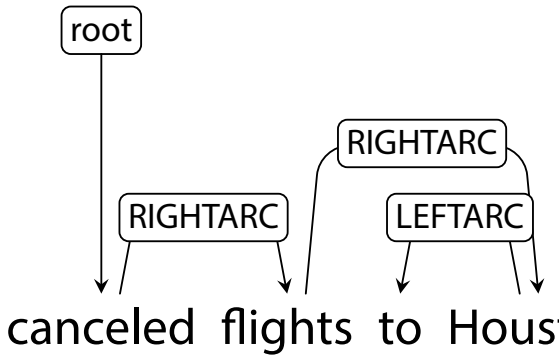
Beispiel: Übergangsbasiertes Shift–Reduce-Parsing

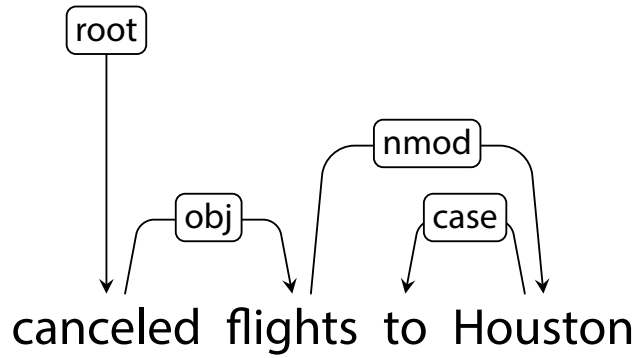




Step	Stack	Word List (Buffer)	Transition	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book $\rightarrow$ me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning,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

Beispiel: RIGHTARC-Einschränkung





	Stack	Word List (Buffer)	Transition	
	[root,canceled,flights]	[to, Houston]	**SHIFT oder RIGHTARC ?**	
mögliche Übergänge:				Relation Added
SHIFT	[root,canceled,flights,to]	[Houston]		-
RIGHTARC	[root,canceled]	[to, Houston]		(canceled → flights)

korrekter Übergang: `SHIFT`

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

4. Dependency-Treebanks

Dependency Treebanks: Datenbasis für Parsing

- Eine **Dependency Treebank** ist ein Korpus, in dem alle Sätze mit vollständigen Dependenzstrukturen annotiert sind.
- **Zweck:**
  - Trainings- und Testdaten für Parsing-Systeme
  - empirische Grundlage für Grammatik- und Sprachvergleich

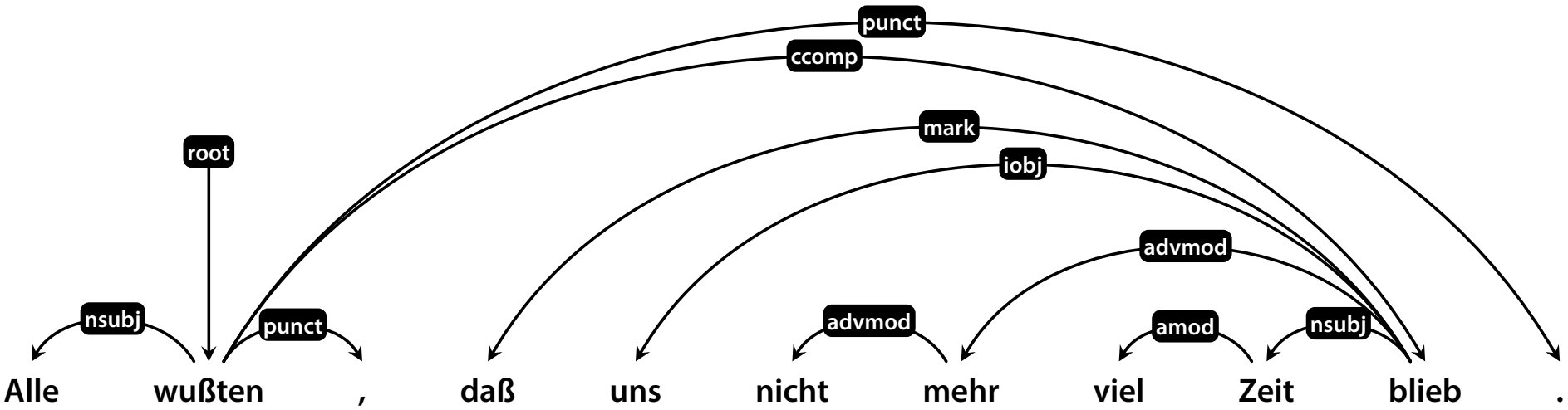
- **Wichtige Formate und Ressourcen:**
  - **UD-Treebanks (Universal Dependencies):** über 100 Sprachen, einheitliche Relationstags (`nsubj`, `obj`, `obl`, …)
  - **TIGER Dependency Bank:** deutsch, konvertierte Version des TIGER-Korpus
  - **CoNLL Shared Tasks:** Standard für Evaluation (CoNLL-U-Format)

- **Typisches Datenformat (CoNLL-U):**

ID	FORM	LEMMA	UPOS	…	HEAD	DEPREL
1	Der	der	DET	…	2	det
2	Hund	Hund	NOUN	…	3	nsubj
3	jagt	jagen	VERB	…	0	root

1	Alle	alle	PRON	PIS	Case=Nom..	2	nsubj	_	_
2	wußten	wissen	VERB	VVFIN	Number=Plur..	0	root	_	SpaceAfter=No
3	,	,	PUNCT	$,	_	2	punct	_	_
4	daß	daß	SCONJ	KOUS	_	10	mark	_	_
5	uns	wir	PRON	PPER	Case=Dat..	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..	9	amod	_	_
9	Zeit	Zeit	NOUN	NN	Case=Nom..	10	nsubj	_	_
10	blieb	bleiben	VERB	VVFIN	Number=Sing..	2	ccomp	_	SpaceAfter=No
11	.	.	PUNCT	$.	_	2	punct	_	_

Tabelle: Satz im CoNLL-Format (deutsches UD-Korpus)



Konversion konstituentenbasierter Treebanks in Dependenzformate

- Dependenzanalysen können auch **sekundär aus konstituentenbasierten Analysen** erzeugt werden.
- Dazu erfolgt eine **Transformation** von **kopfannotierten Konstituentenbäumen** in einen **Dependenzgraphen** (siehe Sitzung 5).
- Vorgehen:
  1. **Identifikation aller *Head–Dependent*-Relationen** mithilfe von *head-finding rules*.
  2. **Labeln der Relationen** anhand von handgeschriebenen Regeln:
     - Bestimmung des Relationstyps **über Strukturpositionen** (z. B. `NP` mit Mutterknoten `S` $\Rightarrow$ **subj**)
     - Bei der **Penn Treebank**: Nutzung **funktionaler Annotationen** in den Nichtterminalsymbolen (z. B. `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*