Syntax natürlicher Sprachen

Vorlesung 9: Technische Aspekte und Parsing-Algorithmen

Martin Schmitt

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

12.01.2021

Arten von Parsing-Algorithmen

Top-Down

- Recursive Descent
- LL (Left-to-right Leftmost (derivation))
- LL(k)
- L(*)
- Earley

Bottom-Up

- Recursive Ascent
- GLR (Generalized Left-to-right Rightmost (derivation))
- Shift-Reduce
- CYK

Arten von Parsing-Algorithmen

Top-Down

- Recursive Descent
- LL (Left-to-right Leftmost (derivation))
- LL(k)
- L(*)
- Earley

Bottom-Up

- Recursive Ascent
- GLR (Generalized Left-to-right Rightmost (derivation))
- Shift-Reduce
- CYK

Themen der heutigen Vorlesung

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing

Nächstes Thema

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing

Recursive Descent Parser

Top-Down-Parsing (dt. *Abwärtsparsen*)

Parsing-Strategie, bei der man von der höchsten Ebene eines Syntaxbaums (Startsymbol der Grammatik) ausgeht und sich mithilfe der Ersetzungsregeln (Produktionsregeln) einer Grammatik bis zu den Terminalen (Lexemen) vorarbeitet.

Recursive Descent Parsing (dt. *rekursiver Abstieg*)

- Form von Top-Down-Parsing
- probiert jede anwendbare Regel aus
- benutzt *Backtracking* im Problemfall
- am intuitivsten „händisch“ zu programmieren
- kann je nach Grammatik zu exponentieller Laufzeit führen (oder sogar zu unendlich langer Laufzeit)

Recursive Descent Parser: Beispiel

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

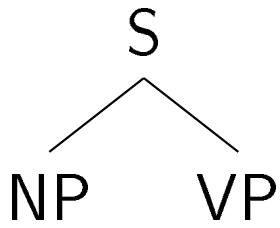
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

S

Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

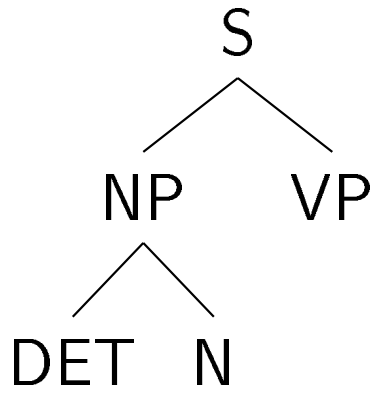
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

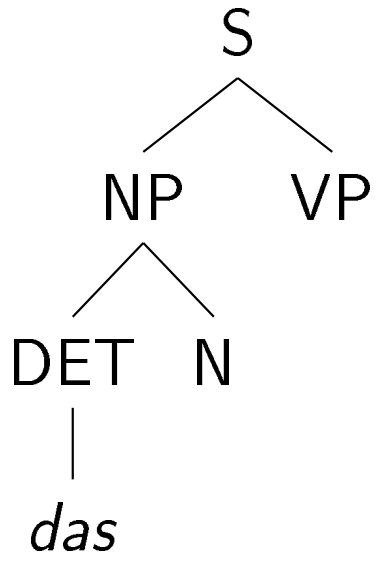
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

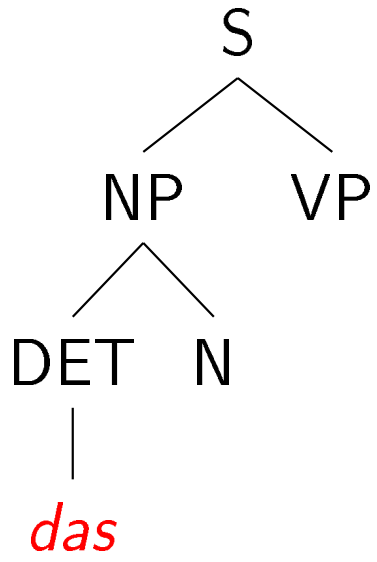
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

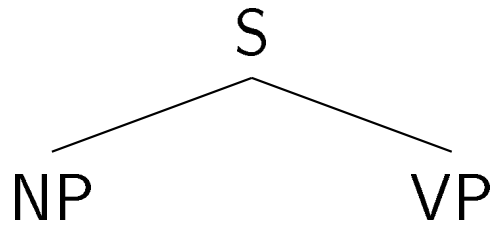
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Backtracking

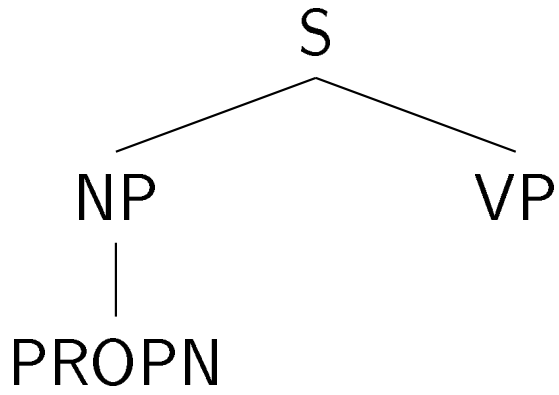
1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

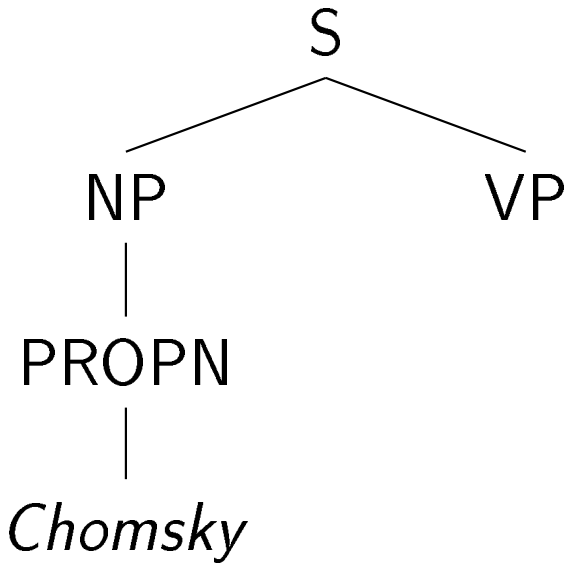
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

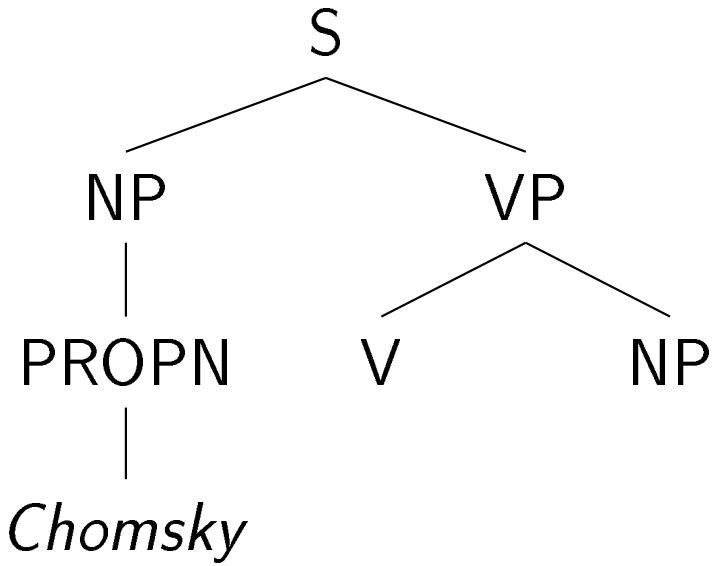
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

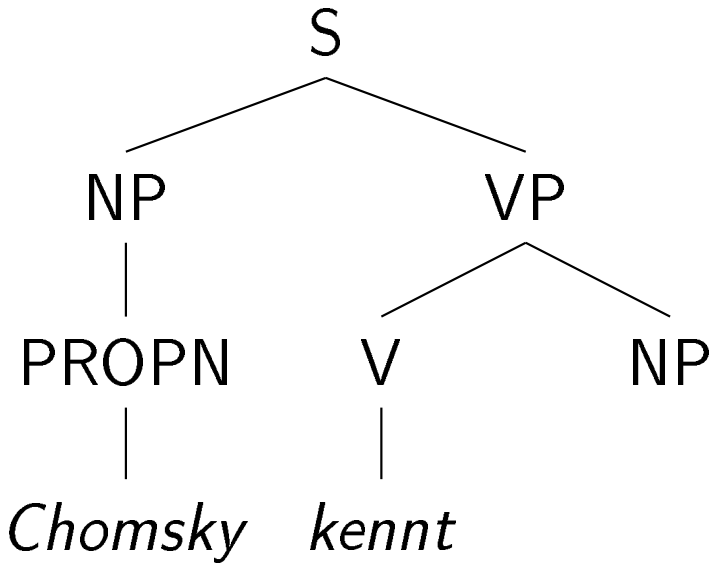
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

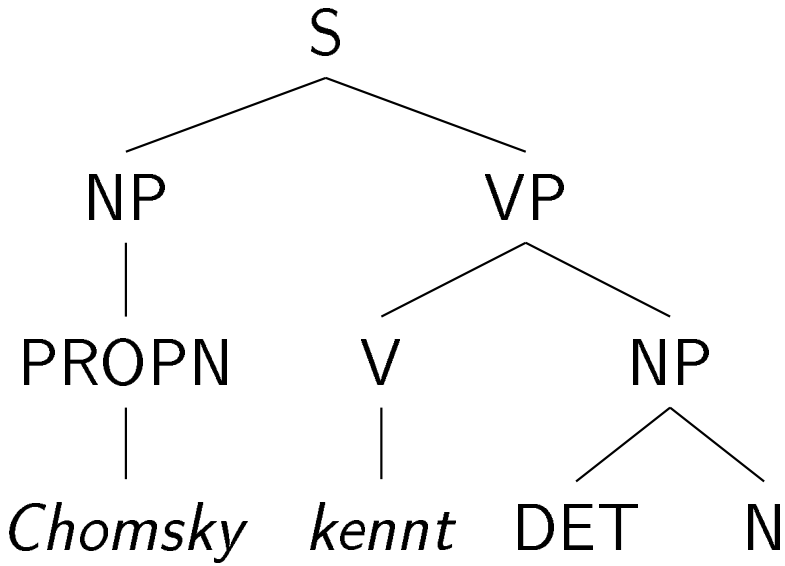
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

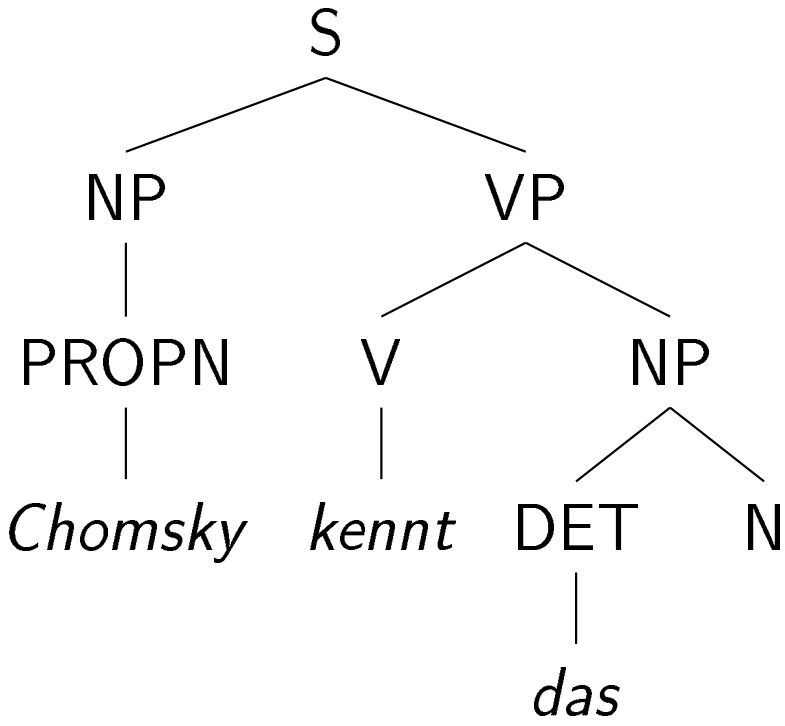
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

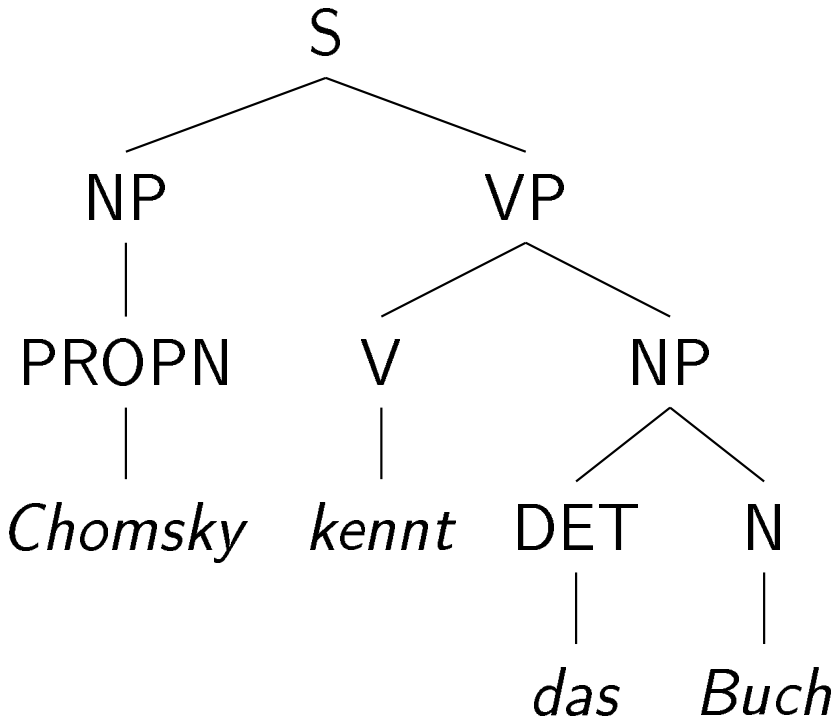
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

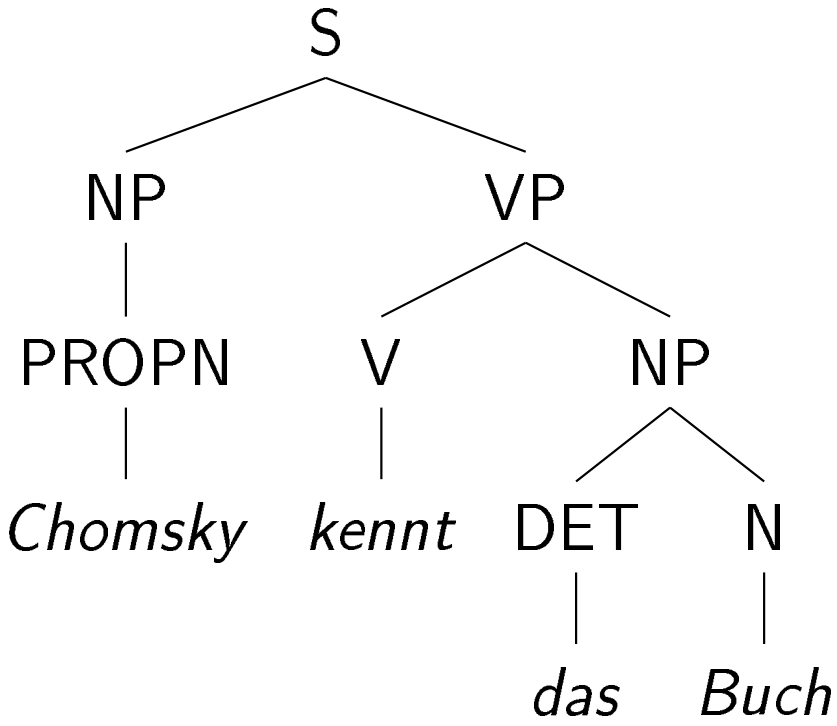
1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Recursive Descent Parser: Beispiel

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

✓

Recursive Descent Parsing: Anmerkungen

Probleme

- Es kann zu jeder Zeit für ein Nichtterminal viele verschiedene Ersetzungsregeln geben.
- Im schlimmsten Fall müssen alle diese Regeln ausprobiert werden (exponentieller Blow-up).
- Viele Teilstrukturen werden erzeugt, obwohl sie nie erfolgreich sein können.
  - $\rightarrow$ Bsp.: Eingabesatz enthält gar nicht die passenden Wörter.

Gefahr der Endlosschleife

- *Links-rekursive* Produktionsregeln führen (bei naiver Ausführung) zu unendlicher Laufzeit!
- Beispiel: ADJP $\rightarrow$ ADJP ADJ

Recursive Descent Parsing: Anmerkungen

Probleme

- Es kann zu jeder Zeit für ein Nichtterminal viele verschiedene Ersetzungsregeln geben.
- Im schlimmsten Fall müssen alle diese Regeln ausprobiert werden (exponentieller Blow-up).
- Viele Teilstrukturen werden erzeugt, obwohl sie nie erfolgreich sein können.
  - $\rightarrow$ Bsp.: Eingabesatz enthält gar nicht die passenden Wörter.

Gefahr der Endlosschleife

- *Links-rekursive* Produktionsregeln führen (bei naiver Ausführung) zu unendlicher Laufzeit!
- Beispiel: ADJP $\rightarrow$ ADJP ADJ

Recursive Descent Parsing: Anmerkungen

Probleme

- Es kann zu jeder Zeit für ein Nichtterminal viele verschiedene Ersetzungsregeln geben.
- Im schlimmsten Fall müssen alle diese Regeln ausprobiert werden (exponentieller Blow-up).
- Viele Teilstrukturen werden erzeugt, obwohl sie nie erfolgreich sein können.
  - → Bsp.: Eingabesatz enthält gar nicht die passenden Wörter.

Gefahr der Endlosschleife

- *Links-rekursive* Produktionsregeln führen (bei naiver Ausführung) zu unendlicher Laufzeit!
- Beispiel: ADJP → ADJP ADJ

Recursive Descent Parsing: Anmerkungen

Probleme

- Es kann zu jeder Zeit für ein Nichtterminal viele verschiedene Ersetzungsregeln geben.
- Im schlimmsten Fall müssen alle diese Regeln ausprobiert werden (exponentieller Blow-up).
- Viele Teilstrukturen werden erzeugt, obwohl sie nie erfolgreich sein können.
  - $\rightarrow$ Bsp.: Eingabesatz enthält gar nicht die passenden Wörter.

Gefahr der Endlosschleife

- *Links-rekursive* Produktionsregeln führen (bei naiver Ausführung) zu unendlicher Laufzeit!
- Beispiel: ADJP $\rightarrow$ ADJP ADJ

Recursive Descent Parsing: Anmerkungen

Probleme

- Es kann zu jeder Zeit für ein Nichtterminal viele verschiedene Ersetzungsregeln geben.
- Im schlimmsten Fall müssen alle diese Regeln ausprobiert werden (exponentieller Blow-up).
- Viele Teilstrukturen werden erzeugt, obwohl sie nie erfolgreich sein können.
  - $\rightarrow$ Bsp.: Eingabesatz enthält gar nicht die passenden Wörter.

Gefahr der Endlosschleife

- *Links-rekursive* Produktionsregeln führen (bei naiver Ausführung) zu unendlicher Laufzeit!
- Beispiel: ADJP $\rightarrow$ ADJP ADJ

Recursive Descent Parsing: Anmerkungen

Probleme

- Es kann zu jeder Zeit für ein Nichtterminal viele verschiedene Ersetzungsregeln geben.
- Im schlimmsten Fall müssen alle diese Regeln ausprobiert werden (exponentieller Blow-up).
- Viele Teilstrukturen werden erzeugt, obwohl sie nie erfolgreich sein können.
  - $\rightarrow$ Bsp.: Eingabesatz enthält gar nicht die passenden Wörter.

Gefahr der Endlosschleife

- *Links-rekursive* Produktionsregeln führen (bei naiver Ausführung) zu unendlicher Laufzeit!
- Beispiel: ADJP $\rightarrow$ ADJP ADJ

Nächstes Thema

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing

Shift Reduce Parsing

Bottom-Up-Parsing (dt. *Aufwärtsparsen*)

Parsing-Strategie, bei der man von den kleinsten vorgefundenen Einheiten (Token, Lexeme, Terminale) ausgeht und versucht, diese nach und nach zu größeren syntaktischen Strukturen zu verbinden, bis man beim Startsymbol der Grammatik angelangt ist.

Shift Reduce Parsing (dt. *Verschieben – Zurückführen*)

- Form von Bottom-Up-Parsing (*datengeleitetes Parsing*)
- gebraucht die Datenstruktur *Stack* (dt. *Stapel*)
- verschiebt Token auf den Stapel, um sie auf Grammatikregeln zurückzuführen

Shift Reduce Parser: Beispiel

Grammatik | Stapel | Ableitungsbaum

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Chomsky kennt das Buch

Grammatik | Stapel | Ableitungsbaum

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Chomsky kennt das Buch

Shift!

Grammatik	Stapel	Ableitungsbaum

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Chomsky

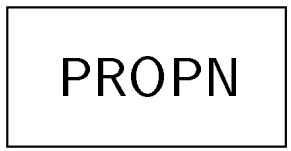
Chomsky

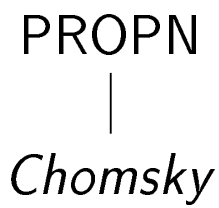
Chomsky kennt das Buch

Reduce!

Grammatik	Stapel	Ableitungsbaum

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



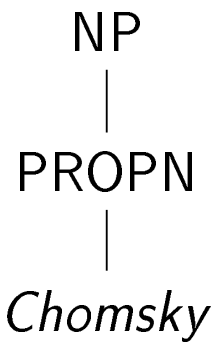


Chomsky kennt das Buch

Reduce!

Grammatik | Stapel | Ableitungsbaum

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Shift!

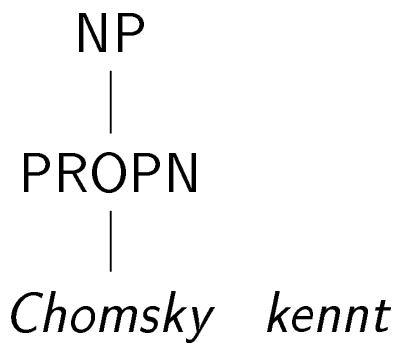
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

kennt
NP

Ableitungsbaum



Chomsky kennt das Buch

Reduce!

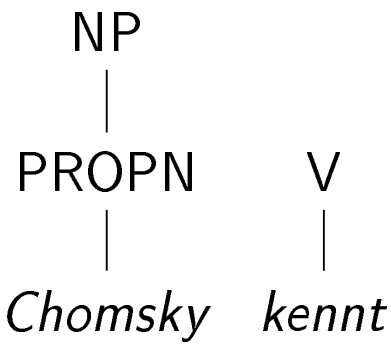
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

V
NP

Ableitungsbaum



Chomsky kennt das Buch

Shift!

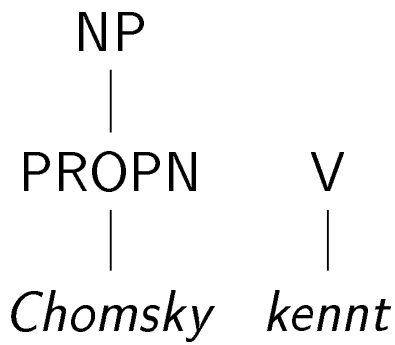
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

das
V
NP

Ableitungsbaum



das

Chomsky kennt das Buch

Reduce!

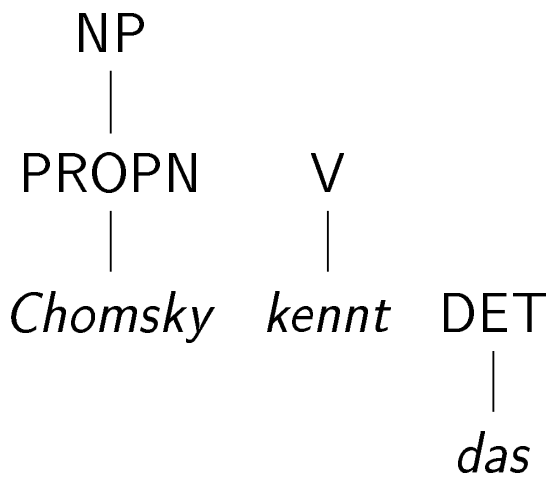
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

DET
V
NP

Ableitungsbaum



Chomsky kennt das Buch

Shift!

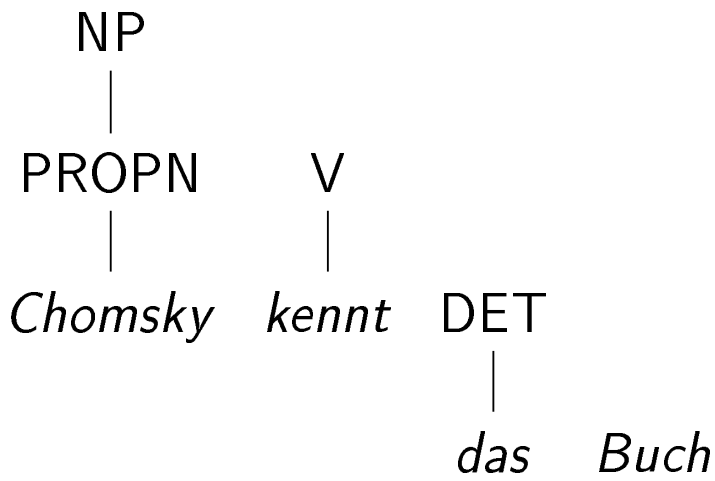
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

Buch
DET
V
NP

Ableitungsbaum



Chomsky kennt das Buch

Reduce!

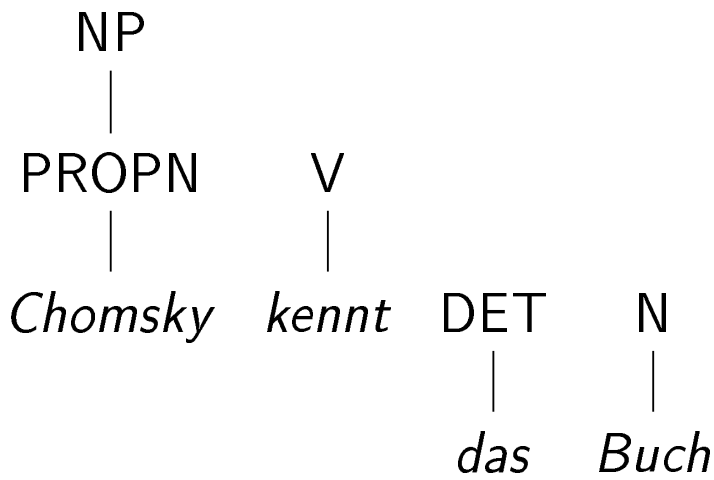
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

N
DET
V
NP

Ableitungsbaum



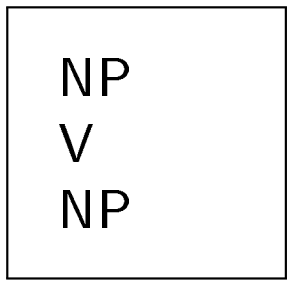
Chomsky kennt das Buch

Reduce!

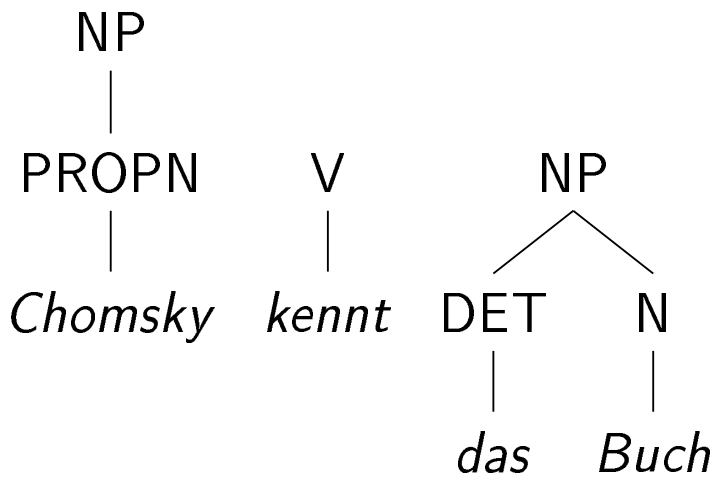
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel



Ableitungsbaum



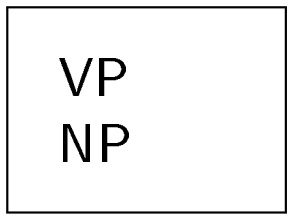
Chomsky kennt das Buch

Reduce!

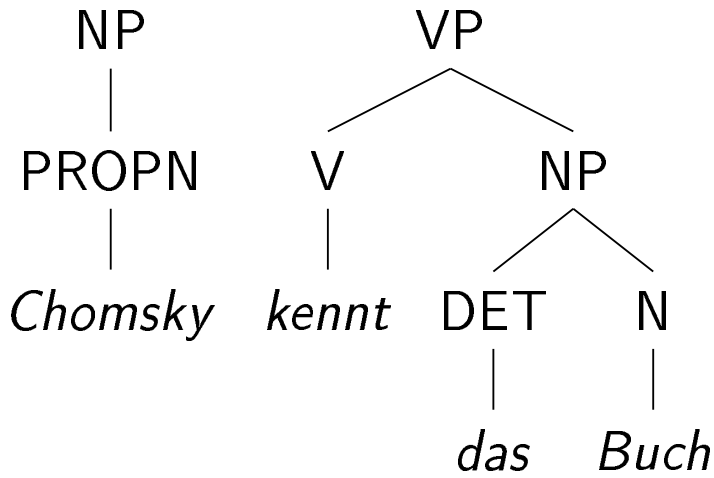
Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel



Ableitungsbaum



Chomsky kennt das Buch

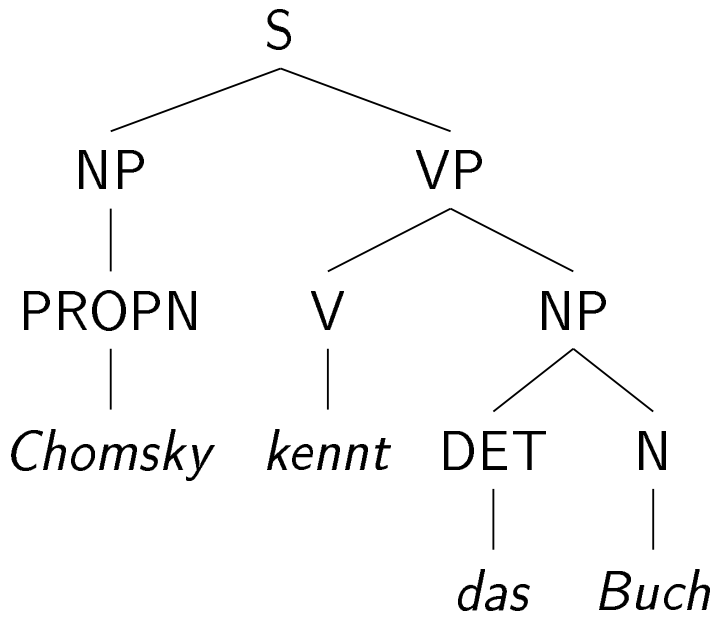
Reduce!

Grammatik

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Stapel

Ableitungsbaum

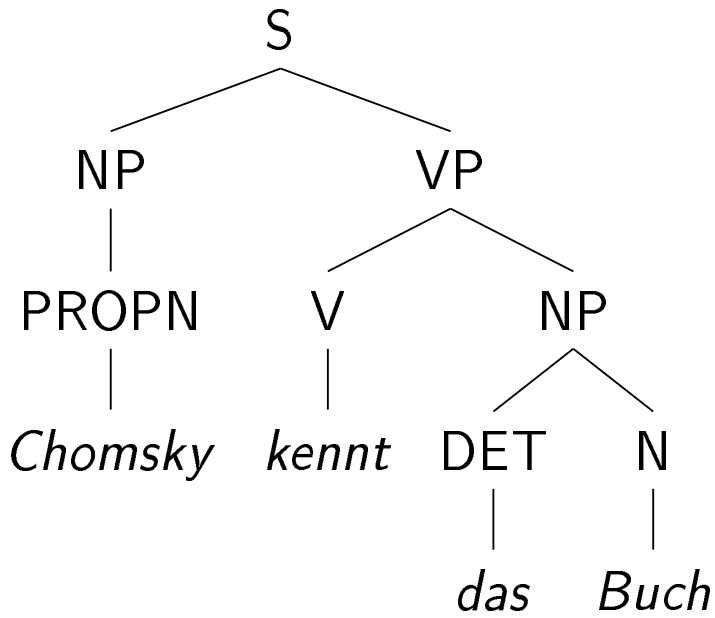


Chomsky kennt das Buch

✓

Grammatik	Stapel	Ableitungsbaum

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt



Chomsky kennt das Buch

Shift Reduce Parsing: Anmerkungen

Vorteile

- arbeitet abhängig von der Eingabe
- ist daher effizienter als ein Top-Down-Parser

Probleme

- erzeugt auch Teilstrukturen, die zu keinem Ergebnis führen
- benötigt also im Allgemeinen auch Backtracking

$\rightarrow$ potentiell exponentielle Laufzeit

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Top-Down vs. Bottom-Up

Top-Down

- startet die Analyse beim Startsymbol
- alterniert zwischen Regelanwendung (*Predict*) und Abgleich mit der Eingabe (*Scan*)
- geht besser mit POS Ambiguitäten um
- baut Strukturen öfter als benötigt
- verbringt viel Zeit mit unmöglichen Ableitungen

Bottom-Up

- startet die Analyse beim Beginn der Eingabe
- alterniert zwischen Einlesen der Eingabe (*Shift*) und „Rückwärtsanwendung“ der Regeln (*Reduce*)
- muss alle lexikalische Ambiguitäten berücksichtigen
- baut benötigte Strukturen nur einmal
- verbringt viel Zeit mit unnötigen Strukturen

Nächstes Thema

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.

Earley Parsing

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.
- Zwischenergebnisse werden in Datenstruktur *(Chart)* gespeichert

Earley Parsing

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.
- Zwischenergebnisse werden in Datenstruktur *(Chart)* gespeichert

Earley Parsing

- Top-Down-Parser (ohne Backtracking)

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.
- Zwischenergebnisse werden in Datenstruktur *(Chart)* gespeichert

Earley Parsing

- Top-Down-Parser (ohne Backtracking)
- Algorithmus kann eigentlich nur Grammatikalität entscheiden.

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.
- Zwischenergebnisse werden in Datenstruktur *(Chart)* gespeichert

Earley Parsing

- Top-Down-Parser (ohne Backtracking)
- Algorithmus kann eigentlich nur Grammatikalität entscheiden.

→ Zur Baumerstellung müssen zusätzliche Verweise gespeichert werden.

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.
- Zwischenergebnisse werden in Datenstruktur *(Chart)* gespeichert

Earley Parsing

- Top-Down-Parser (ohne Backtracking)
- Algorithmus kann eigentlich nur Grammatikalität entscheiden.

→ Zur Baumerstellung müssen zusätzliche Verweise gespeichert werden.

- Komplexität: $\mathcal{O}(n^3)$

Earley Parser

Chart Parsing

- *Dynamische Programmierung* vermeidet doppelte Berechnungen.
- Zwischenergebnisse werden in Datenstruktur *(Chart)* gespeichert

Earley Parsing

- Top-Down-Parser (ohne Backtracking)
- Algorithmus kann eigentlich nur Grammatikalität entscheiden.

→ Zur Baumerstellung müssen zusätzliche Verweise gespeichert werden.

- Komplexität: $\mathcal{O}(n^3)$
- funktioniert nur mit ε-freien Grammatiken!

ε-Eliminierung

ε-Regel

- Regel der Form $A \rightarrow \varepsilon$ (Nichtterminal A wird gelöscht)
- Im `nltk`-Format `A ->` (z. B. für optionale Elemente)

Eliminierungsalgorithmus

1. Wähle ein Nichtterminal A mit einer ε-Regel
2. Entferne die ε-Regel
3. Für jede Regel p mit A auf der rechten Seite:
   dupliziere die Regel für jede mögliche Kombination mit/ohne A
4. Falls es immer noch ε-Regeln gibt, gehe zurück zu Schritt 1.

ε-Eliminierung

ε-Regel

- Regel der Form $A \rightarrow \varepsilon$ (Nichtterminal A wird gelöscht)
- Im `nltk`-Format `A ->` (z. B. für optionale Elemente)

Eliminierungsalgorithmus

1. Wähle ein Nichtterminal A mit einer ε-Regel
2. Entferne die ε-Regel
3. Für jede Regel p mit A auf der rechten Seite:
   dupliziere die Regel für jede mögliche Kombination mit/ohne A
   ($2^{\text{„Anzahl der Vorkommen von } A \text{ in } p\text{“}}$ neue Regeln)
4. Falls es immer noch ε-Regeln gibt, gehe zurück zu Schritt 1.

ε-Eliminierung

Beispiel (Leeres Subjekt bei Imperativ)

```
1 | S  → NP VP
2 | NP → ε
3 | NP → DET N
4 | VP → V
5 | V  → "schlaf" | "schläft"
6 | DET → "der"
7 | N  → "Hund"
```

Zur Vermeidung von Übergenerierung fehlen noch entsprechende Bedingungen! (s. Hausaufgabe)

ε-Eliminierung

Beispiel (nach Eliminierung)

```
1 | S  → NP VP
2 | S  → VP
3 | NP → DET N
4 | VP → V
5 | V  → "schlaf" | "schläft"
6 | DET → "der"
7 | N  → "Hund"
```

Zur Vermeidung von Übergenerierung fehlen noch entsprechende Bedingungen! (s. Hausaufgabe)

Earley Algorithmus I

Gegeben

Eingabesequenz $s = s_1, \ldots, s_n$; Grammatik $G = (T, N, P, S)$

Datenstrukturen

- Position := Tokengrenze
  (z. B. zwischen s_1 und s_2 etc.)
- Zu jeder Pos. Menge Q von *Zuständen*
- Zustand := $(X \to \alpha \cdot \beta,\ i)$
  bestehend aus
  - der aktuellen Produktionsregel $X \to \alpha\beta \in P$,
  - der aktuellen Position in dieser Regel (der Punkt $\cdot$),
  - der Ursprungsposition i in der Eingabe, an der das Abgleichen dieser Regel begann.

Earley Algorithmus II

Operationen

P Prediction (dt. *Voraussage*)
falls $(A \rightarrow \ldots \cdot B \ \ldots, \ j) \in Q_i$ mit $B \in N$, dann für jede Regel $B \rightarrow \alpha \in P$:

setze $(B \rightarrow \cdot \ \alpha, \ i) \in Q_i$

S Scanning (dt. *Überprüfung*)
falls $(A \rightarrow \ldots \cdot a \ \ldots, \ j) \in Q_i$ mit $a \in T$ und $a = s_{i+1}$, dann

setze $(A \rightarrow \ldots \ a \ \cdot \ \ldots, \ j) \in Q_{i+1}$

C Completion (dt. *Vervollständigung*)
falls $(A \rightarrow \ldots \ \cdot, \ j) \in Q_i$, dann für alle Zustände $(B \rightarrow \ldots \cdot A \ \ldots, \ k) \in Q_j$:

setze $(B \rightarrow \ldots \ A \ \cdot \ \ldots, \ k) \in Q_i$

Earley Algorithmus III

Algorithm

1. Initialisiere Q_0 mit dem Zustand $(S' \rightarrow \cdot\ S, 0)$ mit S' frisches nichtterminales Symbol
2. Führe je nach Situation eine der drei Operationen (P, S, C) aus, bis keine weiteren Zustände mehr hinzugefügt werden können.
3. Wiederhole Schritt 2 bis keine neuen Zustände mehr hinzugefügt werden können.
4. Akzeptiere die Eingabesequenz s genau dann, wenn $(S' \rightarrow S\ \cdot, 0) \in Q_{|s|}$

$\Longrightarrow$ Beispiel auf der nächsten Folie

Grammatik:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Pos.	Zustände
Q_0	(S' $\rightarrow \cdot$ S, 0)
Q_1	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

1. S → NP VP
2. NP → DET N
3. NP → PROPN
4. VP → V NP
5. DET → das
6. N → Buch
7. PROPN → Chomsky
8. V → kennt

Pos.	Zustände
Q_0	(S' → · S, 0)
	(S → · NP VP, 0)
Q_1	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Pos.	Zustände
Q_0	(S' $\rightarrow \cdot$ S, 0) (S $\rightarrow \cdot$ NP VP, 0) (NP $\rightarrow \cdot$ DET N, 0)
Q_1	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Pos.	Zustände
Q_0	(S' $\rightarrow \cdot$ S, 0)
	(S $\rightarrow \cdot$ NP VP, 0)
	(NP $\rightarrow \cdot$ DET N, 0)
	(NP $\rightarrow \cdot$ PROPN, 0)
Q_1	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Pos.	Zustände
Q_0	(S' $\rightarrow \cdot$ S, 0)
	(S $\rightarrow \cdot$ NP VP, 0)
	(NP $\rightarrow \cdot$ DET N, 0)
	(NP $\rightarrow \cdot$ PROPN, 0)
	(DET $\rightarrow \cdot$ das, 0)
Q_1	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Pos.	Zustände
Q_0	(S' $\rightarrow \cdot$ S, 0)
	(S $\rightarrow \cdot$ NP VP, 0)
	(NP $\rightarrow \cdot$ DET N, 0)
	(NP $\rightarrow \cdot$ PROPN, 0)
	(DET $\rightarrow \cdot$ das, 0)
	(PROPN $\rightarrow \cdot$ Chomsky, 0)
Q_1	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ DET N
3. NP $\rightarrow$ PROPN
4. VP $\rightarrow$ V NP
5. DET $\rightarrow$ das
6. N $\rightarrow$ Buch
7. PROPN $\rightarrow$ Chomsky
8. V $\rightarrow$ kennt

Pos.	Zustände
Q_0	(S' $\rightarrow \cdot$ S, 0)
	(S $\rightarrow \cdot$ NP VP, 0)
	(NP $\rightarrow \cdot$ DET N, 0)
	(NP $\rightarrow \cdot$ PROPN, 0)
	(DET $\rightarrow \cdot$ das, 0)
	(PROPN $\rightarrow \cdot$ Chomsky, 0)
Q_1	(PROPN $\rightarrow$ Chomsky $\cdot$, 0)

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(4) VP $\rightarrow$ V NP

(8) V $\rightarrow$ kennt

Q_0:

(S' $\rightarrow \cdot$ S, 0)

(S $\rightarrow \cdot$ NP VP, 0)

(NP $\rightarrow \cdot$ DET N, 0)

(NP $\rightarrow \cdot$ PROPN, 0)

(DET $\rightarrow \cdot$ das, 0)

(PROPN $\rightarrow \cdot$ Chomsky, 0)

Pos.	Zustände
Q_1	(PROPN $\rightarrow$ Chomsky $\cdot$, 0)
Q_2	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(4) $\text{VP} \rightarrow \text{V NP}$

(8) $\text{V} \rightarrow \text{kennt}$

Q_0:

$(\text{S'} \rightarrow \cdot\ \text{S}, 0)$

$(\text{S} \rightarrow \cdot\ \text{NP VP}, 0)$

$(\text{NP} \rightarrow \cdot\ \text{DET N}, 0)$

$(\text{NP} \rightarrow \cdot\ \text{PROPN}, 0)$

$(\text{DET} \rightarrow \cdot\ \text{das}, 0)$

$(\text{PROPN} \rightarrow \cdot\ \text{Chomsky}, 0)$

Pos.	Zustände
Q_1	$(\text{PROPN} \rightarrow \text{Chomsky}\ \cdot, 0)$ $(\text{NP} \rightarrow \text{PROPN}\ \cdot, 0)$
Q_2	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(4) VP $\rightarrow$ V NP

(8) V $\rightarrow$ kennt

Q_0:

(S' $\rightarrow \cdot$ S, 0)

(S $\rightarrow \cdot$ NP VP, 0)

(NP $\rightarrow \cdot$ DET N, 0)

(NP $\rightarrow \cdot$ PROPN, 0)

(DET $\rightarrow \cdot$ das, 0)

(PROPN $\rightarrow \cdot$ Chomsky, 0)

Pos.	Zustände
Q_1	(PROPN $\rightarrow$ Chomsky $\cdot$, 0) (NP $\rightarrow$ PROPN $\cdot$, 0) (S $\rightarrow$ NP $\cdot$ VP, 0)
Q_2	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(4) VP → V NP

(8) V → kennt

Q_0:

(S' → · S, 0)

(S → · NP VP, 0)

(NP → · DET N, 0)

(NP → · PROPN, 0)

(DET → · das, 0)

(PROPN → · Chomsky, 0)

Pos.	Zustände
Q_1	(PROPN → Chomsky ·, 0) (NP → PROPN ·, 0) (S → NP · VP, 0) (VP → · V NP, 1)
Q_2	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(4) $\text{VP} \rightarrow \text{V NP}$

(8) $\text{V} \rightarrow \text{kennt}$

Q_0:

$(\text{S'} \rightarrow \cdot\ \text{S}, 0)$

$(\text{S} \rightarrow \cdot\ \text{NP VP}, 0)$

$(\text{NP} \rightarrow \cdot\ \text{DET N}, 0)$

$(\text{NP} \rightarrow \cdot\ \text{PROPN}, 0)$

$(\text{DET} \rightarrow \cdot\ \text{das}, 0)$

$(\text{PROPN} \rightarrow \cdot\ \text{Chomsky}, 0)$

Pos.	Zustände
Q_1	$(\text{PROPN} \rightarrow \text{Chomsky}\ \cdot, 0)$ $(\text{NP} \rightarrow \text{PROPN}\ \cdot, 0)$ $(\text{S} \rightarrow \text{NP}\ \cdot\ \text{VP}, 0)$ $(\text{VP} \rightarrow \cdot\ \text{V NP}, 1)$ $(\text{V} \rightarrow \cdot\ \text{kennt}, 1)$
Q_2	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(4) $\text{VP} \rightarrow \text{V NP}$

(8) $\text{V} \rightarrow \text{kennt}$

Q_0:

$(\text{S'} \rightarrow \cdot\ \text{S}, 0)$

$(\text{S} \rightarrow \cdot\ \text{NP VP}, 0)$

$(\text{NP} \rightarrow \cdot\ \text{DET N}, 0)$

$(\text{NP} \rightarrow \cdot\ \text{PROPN}, 0)$

$(\text{DET} \rightarrow \cdot\ \text{das}, 0)$

$(\text{PROPN} \rightarrow \cdot\ \text{Chomsky}, 0)$

Pos.	Zustände
Q_1	$(\text{PROPN} \rightarrow \text{Chomsky}\ \cdot, 0)$ $(\text{NP} \rightarrow \text{PROPN}\ \cdot, 0)$ $(\text{S} \rightarrow \text{NP}\ \cdot\ \text{VP}, 0)$ $(\text{VP} \rightarrow \cdot\ \text{V NP}, 1)$ $(\text{V} \rightarrow \cdot\ \text{kennt}, 1)$
Q_2	$(\text{V} \rightarrow \text{kennt}\ \cdot, 1)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) NP $\rightarrow$ DET N

(3) NP $\rightarrow$ PROPN

(5) DET $\rightarrow$ das

(7) PROPN $\rightarrow$ Chomsky

Q_1:

(PROPN $\rightarrow$ Chomsky $\cdot$, 0)

(NP $\rightarrow$ PROPN $\cdot$, 0)

(S $\rightarrow$ NP $\cdot$ VP, 0)

(VP $\rightarrow \cdot$ V NP, 1)

(V $\rightarrow \cdot$ kennt, 1)

Pos.	Zustände
Q_2	(V $\rightarrow$ kennt $\cdot$, 1)
Q_3	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) NP → DET N

(3) NP → PROPN

(5) DET → das

(7) PROPN → Chomsky

Q_1:

(PROPN → Chomsky ·, 0)

(NP → PROPN ·, 0)

(S → NP · VP, 0)

(VP → · V NP, 1)

(V → · kennt, 1)

Pos.	Zustände
Q_2	(V → kennt ·, 1) (VP → V · NP, 1)
Q_3	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) NP → DET N

(3) NP → PROPN

(5) DET → das

(7) PROPN → Chomsky

Q_1:

(PROPN → Chomsky ·, 0)

(NP → PROPN ·, 0)

(S → NP · VP, 0)

(VP → · V NP, 1)

(V → · kennt, 1)

Pos.	Zustände
Q_2	(V → kennt ·, 1) (VP → V · NP, 1) (NP → · DET N, 2)
Q_3	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) NP $\rightarrow$ DET N

(3) NP $\rightarrow$ PROPN

(5) DET $\rightarrow$ das

(7) PROPN $\rightarrow$ Chomsky

Q_1:

(PROPN $\rightarrow$ Chomsky $\cdot$, 0)

(NP $\rightarrow$ PROPN $\cdot$, 0)

(S $\rightarrow$ NP $\cdot$ VP, 0)

(VP $\rightarrow$ $\cdot$ V NP, 1)

(V $\rightarrow$ $\cdot$ kennt, 1)

Pos.	Zustände
Q_2	(V $\rightarrow$ kennt $\cdot$, 1) (VP $\rightarrow$ V $\cdot$ NP, 1) (NP $\rightarrow$ $\cdot$ DET N, 2) (NP $\rightarrow$ $\cdot$ PROPN, 2)
Q_3	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) NP $\rightarrow$ DET N

(3) NP $\rightarrow$ PROPN

(5) DET $\rightarrow$ das

(7) PROPN $\rightarrow$ Chomsky

Q_1:

(PROPN $\rightarrow$ Chomsky $\cdot$, 0)

(NP $\rightarrow$ PROPN $\cdot$, 0)

(S $\rightarrow$ NP $\cdot$ VP, 0)

(VP $\rightarrow$ $\cdot$ V NP, 1)

(V $\rightarrow$ $\cdot$ kennt, 1)

Pos.	Zustände
Q_2	(V $\rightarrow$ kennt $\cdot$, 1) (VP $\rightarrow$ V $\cdot$ NP, 1) (NP $\rightarrow$ $\cdot$ DET N, 2) (NP $\rightarrow$ $\cdot$ PROPN, 2) (DET $\rightarrow$ $\cdot$ das, 2)
Q_3	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) NP $\rightarrow$ DET N

(3) NP $\rightarrow$ PROPN

(5) DET $\rightarrow$ das

(7) PROPN $\rightarrow$ Chomsky

Q_1:

(PROPN $\rightarrow$ Chomsky $\cdot$, 0)

(NP $\rightarrow$ PROPN $\cdot$, 0)

(S $\rightarrow$ NP $\cdot$ VP, 0)

(VP $\rightarrow$ $\cdot$ V NP, 1)

(V $\rightarrow$ $\cdot$ kennt, 1)

Pos.	Zustände
Q_2	(V $\rightarrow$ kennt $\cdot$, 1) (VP $\rightarrow$ V $\cdot$ NP, 1) (NP $\rightarrow$ $\cdot$ DET N, 2) (NP $\rightarrow$ $\cdot$ PROPN, 2) (DET $\rightarrow$ $\cdot$ das, 2) (PROPN $\rightarrow$ $\cdot$ Chomsky, 2)
Q_3	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(2) $\text{NP} \rightarrow \text{DET N}$

(3) $\text{NP} \rightarrow \text{PROPN}$

(5) $\text{DET} \rightarrow \text{das}$

(7) $\text{PROPN} \rightarrow \text{Chomsky}$

Q_1:

$(\text{PROPN} \rightarrow \text{Chomsky} \cdot, 0)$

$(\text{NP} \rightarrow \text{PROPN} \cdot, 0)$

$(\text{S} \rightarrow \text{NP} \cdot \text{VP}, 0)$

$(\text{VP} \rightarrow \cdot \text{V NP}, 1)$

$(\text{V} \rightarrow \cdot \text{kennt}, 1)$

Pos.	Zustände
Q_2	$(\text{V} \rightarrow \text{kennt} \cdot, 1)$
	$(\text{VP} \rightarrow \text{V} \cdot \text{NP}, 1)$
	$(\text{NP} \rightarrow \cdot \text{DET N}, 2)$
	$(\text{NP} \rightarrow \cdot \text{PROPN}, 2)$
	$(\text{DET} \rightarrow \cdot \text{das}, 2)$
	$(\text{PROPN} \rightarrow \cdot \text{Chomsky}, 2)$
Q_3	$(\text{DET} \rightarrow \text{das} \cdot, 2)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$
Q_4	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$
Q_4	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) N → Buch

Q_0:

(S' → · S, 0)

Q_1:

(S → NP · VP, 0)

(VP → · V NP, 1)

Q_2:

(VP → V · NP, 1)

(NP → · DET N, 2)

(DET → · das, 2)

Pos.	Zustände
Q_3	(DET → das ·, 2) (NP → DET · N, 2) (N → · Buch, 3)
Q_4	

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$ $(N \rightarrow \cdot\ Buch, 3)$
Q_4	$(N \rightarrow Buch\ \cdot, 3)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$ $(N \rightarrow \cdot\ Buch, 3)$
Q_4	$(N \rightarrow Buch\ \cdot, 3)$ $(NP \rightarrow DET\ N\ \cdot, 2)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$ $(N \rightarrow \cdot\ Buch, 3)$
Q_4	$(N \rightarrow Buch\ \cdot, 3)$ $(NP \rightarrow DET\ N\ \cdot, 2)$ $(VP \rightarrow V\ NP\ \cdot, 1)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$ $(N \rightarrow \cdot\ Buch, 3)$
Q_4	$(N \rightarrow Buch\ \cdot, 3)$ $(NP \rightarrow DET\ N\ \cdot, 2)$ $(VP \rightarrow V\ NP\ \cdot, 1)$ $(S \rightarrow NP\ VP\ \cdot, 0)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$ $(N \rightarrow \cdot\ Buch, 3)$
Q_4	$(N \rightarrow Buch\ \cdot, 3)$ $(NP \rightarrow DET\ N\ \cdot, 2)$ $(VP \rightarrow V\ NP\ \cdot, 1)$ $(S \rightarrow NP\ VP\ \cdot, 0)$ $(S' \rightarrow S\ \cdot, 0)$

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Grammatik:

(6) $N \rightarrow$ Buch

Q_0:

$(S' \rightarrow \cdot\ S, 0)$

Q_1:

$(S \rightarrow NP\ \cdot\ VP, 0)$

$(VP \rightarrow \cdot\ V\ NP, 1)$

Q_2:

$(VP \rightarrow V\ \cdot\ NP, 1)$

$(NP \rightarrow \cdot\ DET\ N, 2)$

$(DET \rightarrow \cdot\ das, 2)$

Pos.	Zustände
Q_3	$(DET \rightarrow das\ \cdot, 2)$ $(NP \rightarrow DET\ \cdot\ N, 2)$ $(N \rightarrow \cdot\ Buch, 3)$
Q_4	$(N \rightarrow Buch\ \cdot, 3)$ $(NP \rightarrow DET\ N\ \cdot, 2)$ $(VP \rightarrow V\ NP\ \cdot, 1)$ $(S \rightarrow NP\ VP\ \cdot, 0)$ $(S' \rightarrow S\ \cdot, 0)$ ✓

$_0$ Chomsky $_1$ kennt $_2$ das $_3$ Buch $_4$

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

$\rightarrow$ Dafür wesentlich schneller

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

$\rightarrow$ Dafür wesentlich schneller

Komplexität

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

$\rightarrow$ Dafür wesentlich schneller

Komplexität

- Laufzeit in $\mathcal{O}(n^3)$ im schlimmsten Fall

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

$\rightarrow$ Dafür wesentlich schneller

Komplexität

- Laufzeit in $\mathcal{O}(n^3)$ im schlimmsten Fall
- Für unambige Grammatiken sogar $\mathcal{O}(n^2)$

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

$\rightarrow$ Dafür wesentlich schneller

Komplexität

- Laufzeit in $\mathcal{O}(n^3)$ im schlimmsten Fall
- Für unambige Grammatiken sogar $\mathcal{O}(n^2)$
- Für bestimmte Typen von Grammatiken (LR) sogar $\mathcal{O}(n)$

Earley Parser: Zusammenfassung

Top-Down-Parsing mit Extras

- Zwischenergebnisse werden in Datenstruktur (Chart) gespeichert ($\rightarrow$ Chart-Parsing, Dynamische Programmierung)
- Zustände werden mit Positionen in der Eingabesequenz abgeglichen (Elemente des Bottom-Up-Parsings)

$\rightarrow$ Komplizierter als *Recursive Descent* und *Shift Reduce*

$\rightarrow$ Dafür wesentlich schneller

Komplexität

- Laufzeit in $\mathcal{O}(n^3)$ im schlimmsten Fall
- Für unambige Grammatiken sogar $\mathcal{O}(n^2)$
- Für bestimmte Typen von Grammatiken (LR) sogar $\mathcal{O}(n)$
- Funktioniert am besten mit *links-rekursiven* Regeln

Nächstes Thema

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing

Earley Algorithmus mit Merkmalen

1. Möglichkeit

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

$\rightarrow$ Optimierungspotential

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

$\rightarrow$ Optimierungspotential

2. Möglichkeit

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

→ Optimierungspotential

2. Möglichkeit

- Merkmalstruktur zu jedem Earley-Zustand hinzufügen

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

→ Optimierungspotential

2. Möglichkeit

- Merkmalstruktur zu jedem Earley-Zustand hinzufügen
- Complete-Operation unifiziert die Merkmalstrukturen der beiden Zustände

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

→ Optimierungspotential

2. Möglichkeit

- Merkmalstruktur zu jedem Earley-Zustand hinzufügen
- Complete-Operation unifiziert die Merkmalstrukturen der beiden Zustände
- Predict-Operation fügt neuen Zustand nur hinzu, wenn er von keinem vorhandenen *subsumiert* wird

Earley Algorithmus mit Merkmalen

1. Möglichkeit

- Parsen wie bisher und am Ende versuchen, zu unifizieren
- Unschön: Zahl von möglichen Analysen wird nicht so früh wie möglich beschränkt

→ Optimierungspotential

2. Möglichkeit

- Merkmalstruktur zu jedem Earley-Zustand hinzufügen
- Complete-Operation unifiziert die Merkmalstrukturen der beiden Zustände
- Predict-Operation fügt neuen Zustand nur hinzu, wenn er von keinem vorhandenen *subsumiert* wird
- Nicht-destruktive Unifikation einsetzen! (Kopien machen!)

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

- im Allgemeinen ist Unifikationsparsen „relativ teuer“

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

- im Allgemeinen ist Unifikationsparsen „relativ teuer“
- NP-vollständig in manchen Versionen

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

- im Allgemeinen ist Unifikationsparsen „relativ teuer“
- NP-vollständig in manchen Versionen
- mit sehr umfangreichen Constraints sogar Turing-vollständig (!)

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

- im Allgemeinen ist Unifikationsparsen „relativ teuer“
- NP-vollständig in manchen Versionen
- mit sehr umfangreichen Constraints sogar Turing-vollständig (!)
- Zahlreiche Varianten existieren

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

- im Allgemeinen ist Unifikationsparsen „relativ teuer“
- NP-vollständig in manchen Versionen
- mit sehr umfangreichen Constraints sogar Turing-vollständig (!)
- Zahlreiche Varianten existieren
  - Quasi-destruktive Unifikation (Hideto Tomabechi)

Merkmalbasiertes Parsing: Komplexität

Unterschied gegenüber ursprünglichem Earley Parser

- Zustandsmenge nach Zuständen durchsuchen, deren Merkmalstrukturen mit gegebener Merkmalstruktur unifizieren
- Häufiges Kopieren von Merkmalstrukturen (nicht-destruktive Unifikation)

Komplexität

- im Allgemeinen ist Unifikationsparsen „relativ teuer“
- NP-vollständig in manchen Versionen
- mit sehr umfangreichen Constraints sogar Turing-vollständig (!)
- Zahlreiche Varianten existieren
  - Quasi-destruktive Unifikation (Hideto Tomabechi)
  - Tractable HPSG (Gerald Penn)

Nächstes Thema

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing

Orakel aus Daten ableiten I

Orakel

Eine Funktion, die immer die richtige, nächste Operation liefert.

Orakel aus Daten ableiten I

Orakel

Eine Funktion, die immer die richtige, nächste Operation liefert.

Probabilistisches Modell als Orakel

Eine Wahrscheinlichkeitsverteilung über Operationen gegeben der aktuelle Zustand (des Stapels, des Lesebuffers, der bisherigen Analyse): $P(op \mid state)$

Orakel aus Daten ableiten II

Ableiten eines probabilistischen Modells aus Daten

- zu nutzende Features (Merkmale) des Parser-Zustandes festlegen

Orakel aus Daten ableiten II

Ableiten eines probabilistischen Modells aus Daten

- zu nutzende Features (Merkmale) des Parser-Zustandes festlegen
- gegeben ein Korpus aus Sätzen und Syntaxbäumen, erstelle für jeden Satz Paare von Features und korrekten Operationen ($\Rightarrow$ Trainingsdaten)

Orakel aus Daten ableiten II

Ableiten eines probabilistischen Modells aus Daten

- zu nutzende Features (Merkmale) des Parser-Zustandes festlegen
- gegeben ein Korpus aus Sätzen und Syntaxbäumen, erstelle für jeden Satz Paare von Features und korrekten Operationen ($\Rightarrow$ Trainingsdaten)
- Optimiere die Parameter eines statistischen Wahrscheinlichkeitsmodells dahingehend, dass es so oft wie möglich, die richtige Vorhersage macht.

Kodierung von Merkmalen

Binäre Merkmale

Ein binäres Merkmal ist entweder vorhanden (1) oder nicht (0).

Beispiele

- $f_1 \equiv \mathit{stack} = [\ldots|\text{in}|\text{München}]$
- $f_2 \equiv \mathit{stack} = [\ldots|\text{München}]$ und $\mathit{buffer} = [\text{gewesen}|\ldots]$

Merkmalsvektoren

- repräsentieren alle möglichen Merkmale als Folge von 0 und 1
- ermöglichen Methoden der linearen Algebra ($\Rightarrow$ Modell)
- Bsp.: $\vec{x} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$

Softmax-Regression

Einfaches lineares Modell

Sei Matrix $W \in \mathbb{R}^{|Op| \times |F|}$ mit Op Operationen und F Features.
Jede Zeile (Operation) gewichtet die Merkmale anders.
Wir berechnen Scores für jede Operation durch Multiplikation: $W\vec{x}$

Softmax-Regression

Einfaches lineares Modell

Sei Matrix $W \in \mathbb{R}^{|Op| \times |F|}$ mit Op Operationen und F Features.
Jede Zeile (Operation) gewichtet die Merkmale anders.
Wir berechnen Scores für jede Operation durch Multiplikation: $W\vec{x}$

Normalisierung durch Softmax

$$softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

Softmax-Regression

Einfaches lineares Modell

Sei Matrix $W \in \mathbb{R}^{|Op| \times |F|}$ mit Op Operationen und F Features.
Jede Zeile (Operation) gewichtet die Merkmale anders.
Wir berechnen Scores für jede Operation durch Multiplikation: $W\vec{x}$

Normalisierung durch Softmax

$$softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

- Werte liegen zwischen 0 und 1

Softmax-Regression

Einfaches lineares Modell

Sei Matrix $W \in \mathbb{R}^{|Op| \times |F|}$ mit Op Operationen und F Features.
Jede Zeile (Operation) gewichtet die Merkmale anders.
Wir berechnen Scores für jede Operation durch Multiplikation: $W\vec{x}$

Normalisierung durch Softmax

$$softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

- Werte liegen zwischen 0 und 1
- $\sum_j softmax(\vec{v})_j = 1$

Softmax-Regression

Einfaches lineares Modell

Sei Matrix $W \in \mathbb{R}^{|Op| \times |F|}$ mit Op Operationen und F Features.
Jede Zeile (Operation) gewichtet die Merkmale anders.
Wir berechnen Scores für jede Operation durch Multiplikation: $W\vec{x}$

Normalisierung durch Softmax

$$softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

- Werte liegen zwischen 0 und 1
- $\sum_j softmax(\vec{v})_j = 1$
- $softmax(\vec{v})_{op}$ ist die Wahrscheinlichkeit für Operation op

Training

Kosten-/Gütefunktion

Negative log likelihood

$$\mathcal{L}(W) = -\sum_{i=1}^{n} \log P(y^i|\vec{x}^i)$$

- Trainiert wird durch Minimierung der Kostenfunktion
- Maximierung von Güte entspricht der Minimierung von Kosten
- Der Unterschied ist ein Minuszeichen
- Zahlreiche Bibliotheken existieren, die die automatische Optimierung einer solchen Funktion ermöglichen: z. B. Pytorch, Tensorflow, Dynet.

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Statistik vs. Formale Grammatik

Statistik

- liefert immer ein Ergebnis
- findet häufig eine gute (wahrscheinliche) Lösung
- Annotatoren können sich auf echte linguistische Beispiele konzentrieren
- Sehr effiziente Berechnungen möglich

Formale Grammatik

- erkennt ungrammatische Sätze; ist aber auch naturgemäß unvollständig
- findet immer alle korrekte(n) Lösung(en)
- Generalisierung muss vom Grammatikschreiber eingeplant werden
- Exakte Algorithmen benötigen polynomiale Laufzeit

Rückblick auf heutige Themen

1. Top-Down-Parsing: Recursive Descent
2. Bottom-Up-Parsing: Shift Reduce
3. Chart Parsing: Earley Algorithmus
4. Parsing mit Merkmalstrukturen
   - Modifizierter Earley Parser
   - Komplexität
5. Ausblick: Statistisches Parsing