Programmierkurs Python II

Vorlesung 7: Parsing II

Michaela Regneri & Stefan Thater
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

Sommersemester 2012

Übersicht

- Kurze Wiederholung:
  - Probleme mit elementaren Parsing-Strategien
- Der CYK-Algorithmus:
  - Parsing als dynamische Programmierung
  - Charts als kompakte Repräsentatation von Teilergebnissen
  - Der Algorithmus: Erkenner & Parser

Probleme

- Die elementaren Parsing-Strategien (top-down, bottom-up) sind nicht auf allgemeine Grammatiken anwendbar
  - keine Tilgungsregeln, keine zyklischen Kettenregeln (BU)
  - keine (links-) rekursiven Regeln (TD)
- Lokale Ambiguität ⇒ Suche & Backtracking
  - Identische Teilergebnisse werden u.U. mehrfach berechnet
  - ⇒ Laufzeit exponentiell in der Eingabelänge (worst case)
- Lokale Ambiguität = Welche Regel bzw. Operation muss angewendet werden?

Beispielgrammatik

S → NP VP	DET → *an*
NP → DET N	DET → *the*
NP → POSS N	POSS → his
NP → NP PP	N → *elephant*
PP → P NP	N → *pajamas*
VP → V NP	N → *boy*
VP → VP PP	V → *shot*
	P → *in*

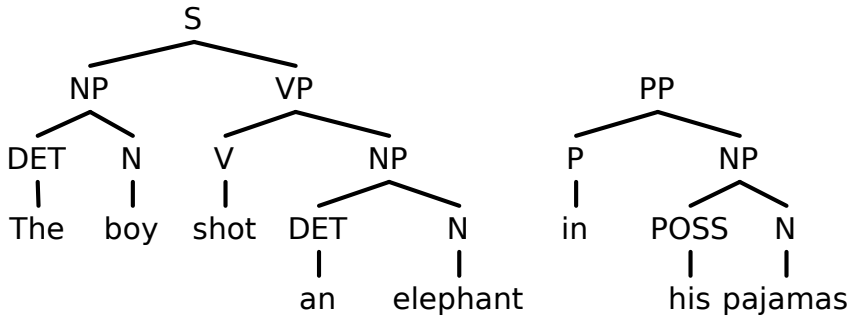
The boy shot an elephant in …

⟨[], [the boy shot an elephant in his pajamas]⟩

⇒* ⟨[NP VP], [in his pajamas]⟩

⇒ ⟨[S], [in his pajamas]⟩

⇒* ⟨[S PP], []⟩ ⇒ Backtracking

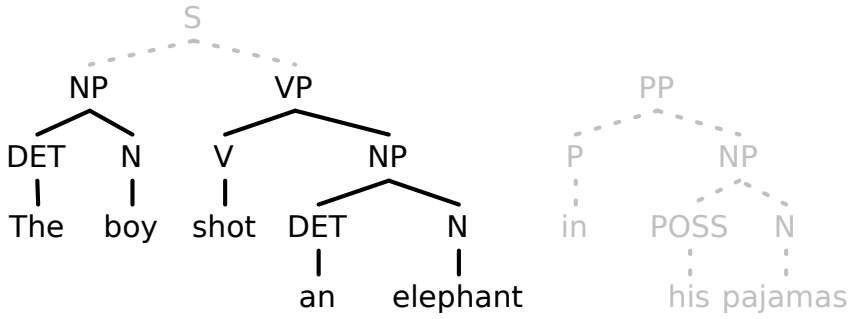


(Shift-Reduce)

The boy shot an elephant in …

⟨[], [the boy shot an elephant in his pajamas]⟩

⇒* ⟨[NP VP], [in his pajamas]⟩



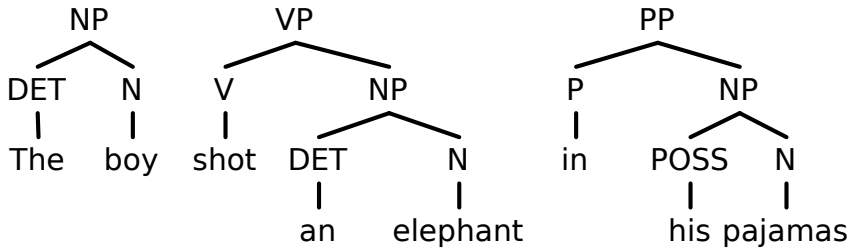
(Shift-Reduce)

The boy shot an elephant in …

⟨[], [the boy shot an elephant in his pajamas]⟩

⇒* ⟨[NP VP], [in his pajamas]⟩

⇒* ⟨[NP VP PP], []⟩



(Shift-Reduce)

The boy shot an elephant in …

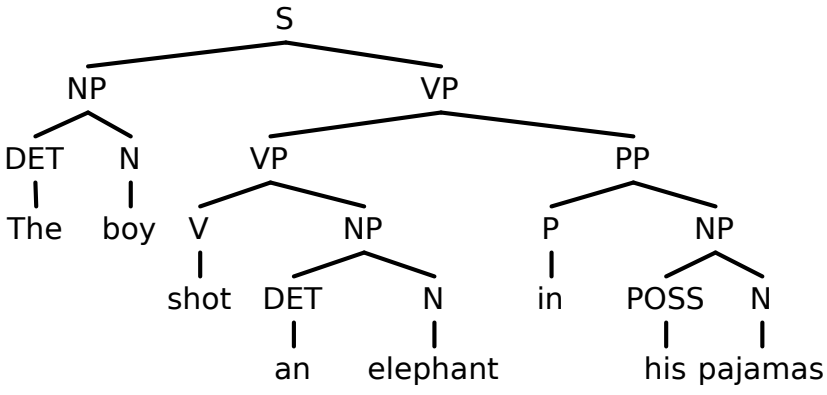
⟨[], [the boy shot an elephant in his pajamas]⟩

⇒* ⟨[NP VP], [in his pajamas]⟩

⇒* ⟨[NP VP PP], []⟩

⇒* ⟨[NP VP], []⟩

⇒* ⟨[S], []⟩



(Shift-Reduce)

Dynamische Programmierung

- Kontextfreie Grammatiken: die Anwendbarkeit einer Regel in einer Ableitung ist unabhängig vom Kontext.

- Chart-Parsing: Speichere bereits analysierte Teilergebnisse (Konstituenten) in einer „Chart.“
- Charts sind kompakte Repräsentation aller (lokal) möglichen Teilkonstituenten der Eingabekette.

Chart-Parsing

- Chart-Parsing: speichere bereits analysierte Teilergebnisse (Konstituenten) in einer „Chart.“
  - Charts aka. „well-formed substring table“
- Charts können enthalten:
  - Konstituenten, die bereits gefunden wurden
  - Hypothesen darüber, welche Konstituenten gefunden werden können (z.B. Earley-Algorithmus).
- Verschiedene Chart-Parser:
  - CYK, Earley, Bottom-up chart parser, …

Charts als Matrix

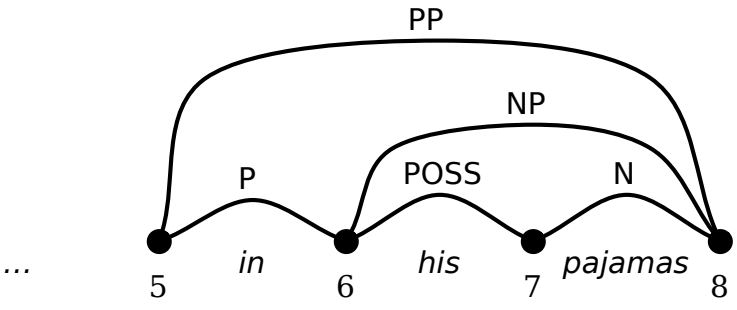
1	DET						
2	NP	N					
3	∅	∅	V				
4	∅	∅	∅	DET			
5	S	∅	VP	NP	N		
6	∅	∅	∅	∅	∅	P	
7	∅	∅	∅	∅	∅	∅	POSS
8	S	∅	VP	NP	∅	PP	NP	N
	0	1	2	3	4	5	6	7

$A \in T[i, j]$ gdw. $A \Rightarrow^* w_{i+1} \ldots w_j$

0 *The* 1 boy 2 *shot* 3 *an* 4 *elephant* 5 *in* 6 *his* 7 *pajamas* 8

Charts als Graph

- Ecken repräsentieren Positionen in der Eingabekette
- Kanten zwischen zwei Knoten repräsentieren Teilketten der Eingabe
  - Kante $n_i \rightarrow n_j$ gdw. $A \Rightarrow^* w_{i+1} \ldots w_j$



CYK Algorithmus

- CYK (Cocke, Younger, Kasami) ist ein einfacher, chart-basierter bottom-up Parser.
- Die Grammatik muss in Chomsky-Normalform vorliegen:
  - $A \rightarrow w$ (w Terminalsymbol)
  - $A \rightarrow B\ C$ (B und C Nichtterminalsymbole)
  - $S \rightarrow \varepsilon$ (S Startsymbol, nur wenn $\varepsilon \in L$)
- Anmerkung: hier nehmen wir an, dass $\varepsilon \notin L$, die Grammatik enthält also keine Regel $S \rightarrow \varepsilon$

Grundlegende Idee

- Wenn
  - $B \Rightarrow^* w_i \ldots w_{j-1}$
  - $C \Rightarrow^* w_j \ldots w_{k-1}$
  - $A \rightarrow B\ C$
- Dann
  - $A \Rightarrow^* w_i \ldots w_{k-1}$

CYK (Erkenner, Pseudo-code)

```
CYK(G, w1 ... wn):
  for i in 1 ... n:
    T[i-1, i] = { A | A → wi ∈ R }
    for j in i - 2 ... 0:
      T[j, i] = ∅
      for k in j + 1 ... i - 1:
        T[j, i] = T[j, i] ∪
                  { A | A → B C, B ∈ T[j,k], C ∈ T[k, i] }
  if S ∈ T[0, n] then return True else return False
```


Eigenschaften

- Korrekt:
  Wenn $S \in T[0, n]$, dann $S \Rightarrow^* w_1 \ldots w_n$
- Vollständig:
  Wenn $S \Rightarrow^* w_1 \ldots w_n$, dann $S \in T[0, n]$
- Laufzeit:
  Polynomiell in der Eingabelänge: $O(n^3)$

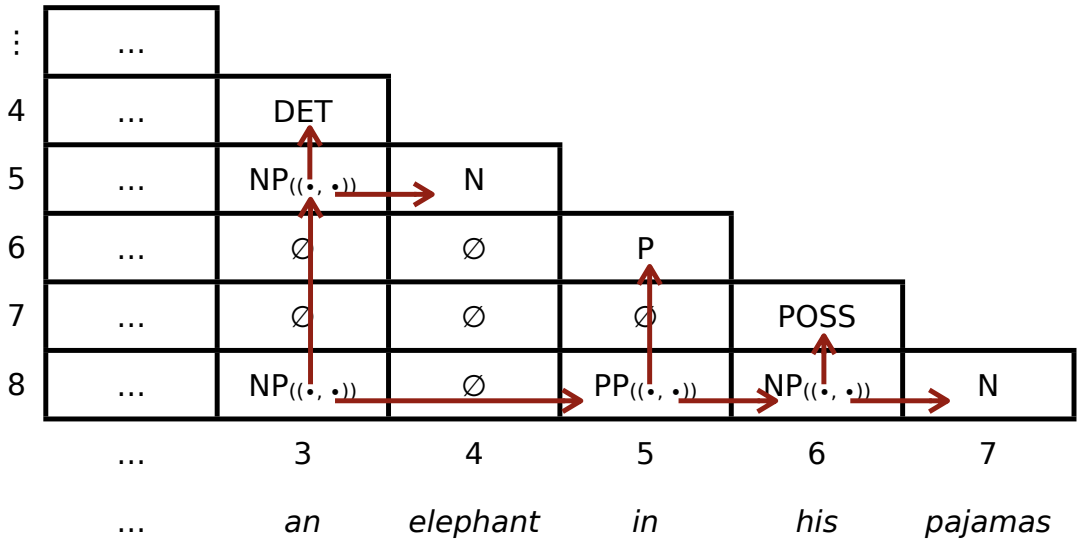
Korrekt & Vollständig

- Behauptung:
  - T[i, i+s] = { A | A ⇒* w_{i+1} ... w_{i+s} }, 0 ≤ i < n, 1 ≤ s ≤ n - i,
- Beweis durch Induktion über s [⇒ Tafel]

Erkenner → Parser

- Speichere zu jeder Kategorie A in der Chart eine Liste von Listen mit Verweisen auf Einträge in der Chart, die verwendet wurden, um A abzuleiten.
- Liste von Listen nötig, da Teilketten mehrere Ableitungsbäume haben können

CYK (Parser)



Chomsky Normalform

- Zu jeder kontextfreien Grammatik G gibt es eine äquivalente kontextfreie Grammatik G’ in Chomsky Normalform.
- Algorithmus:
  - Elimination von ε-Regeln
  - Elimination zu kurzer Regeln
  - Elimination zu langer Regeln

Elimination zu langer Regeln

```
Linksbinarisierung(G):
   while G enthält Regel A → A1 A2 A3 ... Ak, k ≥ 3
      entferne die Regel aus G
      neue Regel: ⟨A1, ...,Ak-1⟩ → A1 ... Ak-1
      neue Regel: A → ⟨A1, ...,Ak-1⟩ Ak

Rechtsbinarisierung(G):
   while G enthält Regel A → A1 A2 A3 ... Ak, k ≥ 3
      entferne die Regel aus G
      neue Regel: ⟨A2, ...,Ak⟩ → A2 ... Ak
      neue Regel: A → A1 ⟨A2, ...,Ak⟩
```


Implementierungsvarianten

- T[i,j] = T[i,j] ∪ { A | A → B C, B ∈ T[i,k], C ∈ T[k,j] }
  - ⇒ kann verschieden implementiert werden
- Variante 1
  - Iteriere über alle Regeln A → B C
  - Prüfe, ob B ∈ T[i,k] und C ∈ T[k,j]
- Variante 2
  - Iteriere über alle B ∈ T[i,k]
  - Iteriere über alle Regeln A → B C
  - Prüfe, ob C ∈ T[k, j]

Implementierungsvarianten

- T[i,j] = T[i,j] ∪ { A | A → B C, B ∈ T[i,k], C ∈ T[k,j] }
  - ⇒ kann verschieden implementiert werden
- Variante 3
  - Iteriere über alle C ∈ T[k,j]
  - Iteriere über alle Regeln A → B C
  - Prüfe, ob A ∈ T[i,k]
- Variante 4
  - Iteriere über alle B ∈ T[i,k] und C ∈ T[k,j]
  - Prüfe, ob es Regel A → B C gibt

Song &al. (2008)

- Ein paar konkrete Zahlen zur Grammatikgröße

	# of Symbols	# of Rules
Original	72	14,971
Right	10,654	25,553
Left	12,944	27,843
Head	11,798	26,697
Compact	3,644	18,543
Ours	8,407	23,306

Table 3: Grammar size of different binarizations

Song &al. (2008)

- Ein paar konkrete Laufzeitresultate:

Binarization	Constituents	Time (s)
Right	241,924,229	5,747
Left	193,238,759	3,474
Head	166,425,179	3,837
Compact	94,257,478	2,302
Ours	**52,206,466**	**2,182**

Table 4: Performance on test set

- Das Test-Set umfasst 90% der Sätze im Wall Street Journal mit maximal 40 Worten.