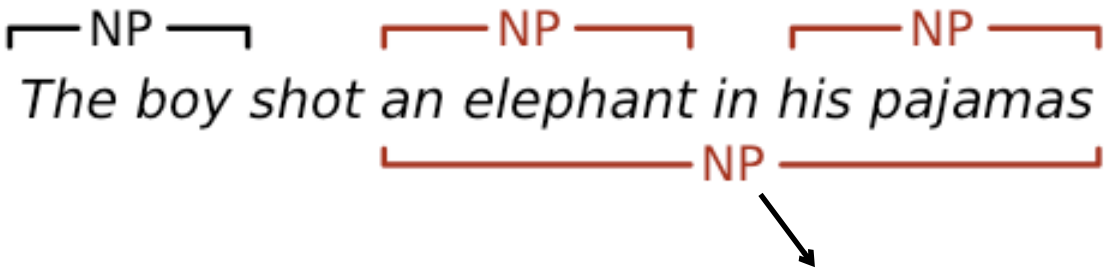
Dynamische Programmierung

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



diese Teile des Satzes können jeweils als NP-Konstituenten analysiert werden, unabhängig davon, wie der finale Parse-Baum aussieht.

One morning I shot an elephant in my pajamas.
How he got into my pajamas I don't know.

Chart-Parsing

- Charts sind kompakte Repräsentation aller (lokal) möglichen Teilkonstituenten der Eingabekette.
- 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, …
  - generieren alle möglichen Parse-Bäume bei Mehrdeutigkeiten

Cocke-Younger-Kasami (CYK) Algorithmus

- Chart-basierter Bottom-up-Parser, nutzt Prinzip der dynamischen Programmierung.
- Entwickelt von John Cocke, Tadao Kasami & Daniel Younger
- Idee:
  - finde alle Konstituenten mit Länge 1
  - finde alle Konstituenten mit Lange 2
  - finde alle Konstitutenten mit Länge 3
  - ...
- 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 $\varepsilon \in L$)
    - hier nehmen wir an, dass $\varepsilon \notin L$

CYK: Chart als Matrix

Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

	I 0	shot 1	an 2	elephant 3	in 4	my 5	pajamas 6
0	N						
1		V					
2			DET				
3				N			
4					P		
5						POSS	
6							N

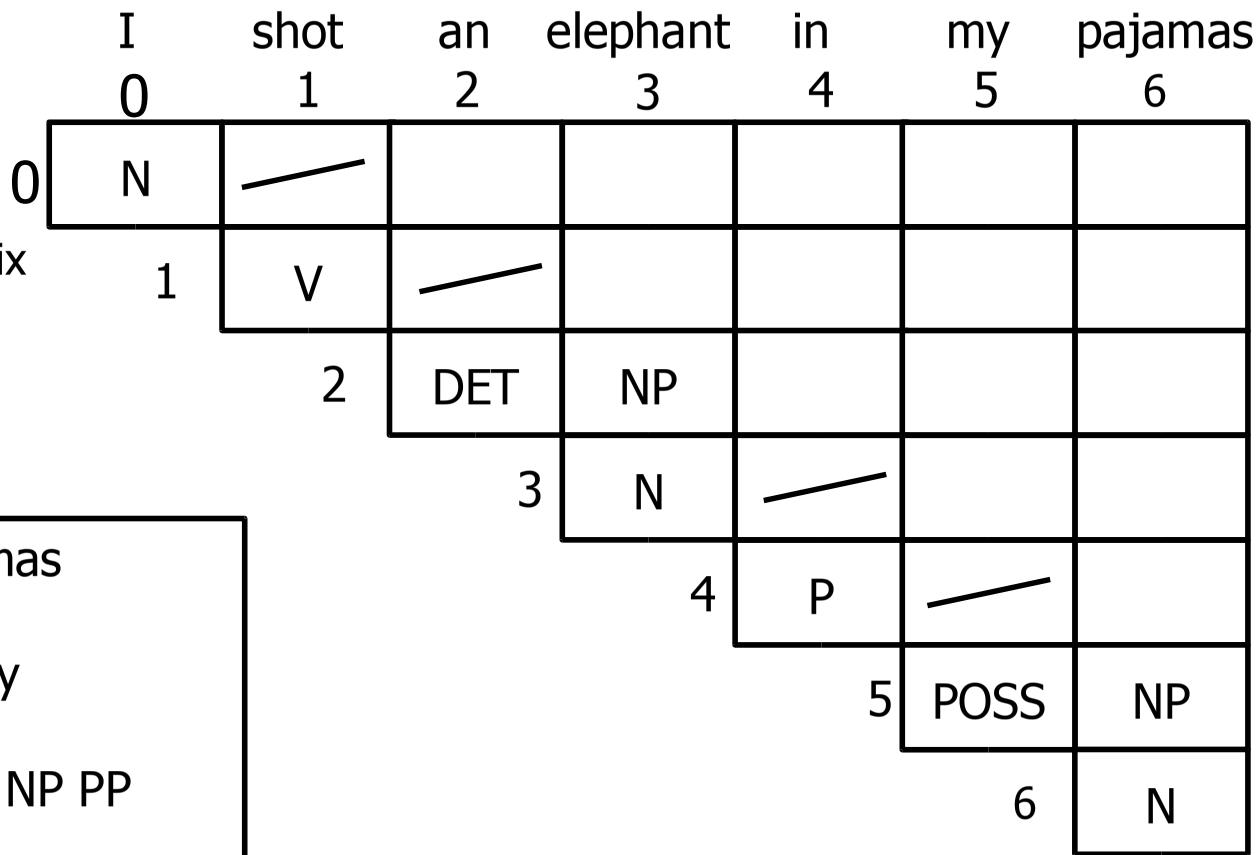
N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 1: Diagonale ausfüllen
→ Terminale zu Nichtterminalen
→ Eintrag für Wort i in Zelle [i,i]
→ benutze Lexikon

CYK: Chart als Matrix

Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]



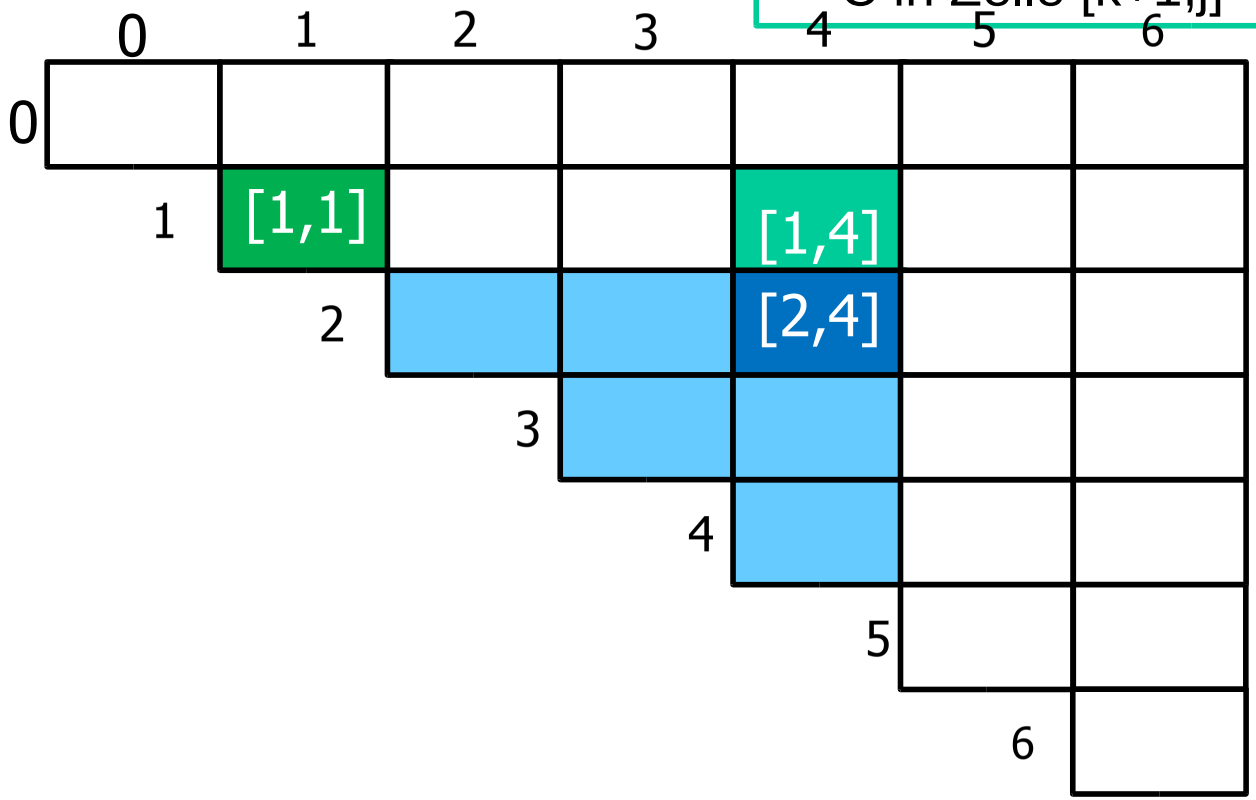
N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 2: Diagonale darüber ausfüllen
→ benutze Regeln der Form A → BC
→ trage A in Zelle [i-1,i] ein wenn B in Zelle [i-1,i-1] und C in Zelle [i,i]

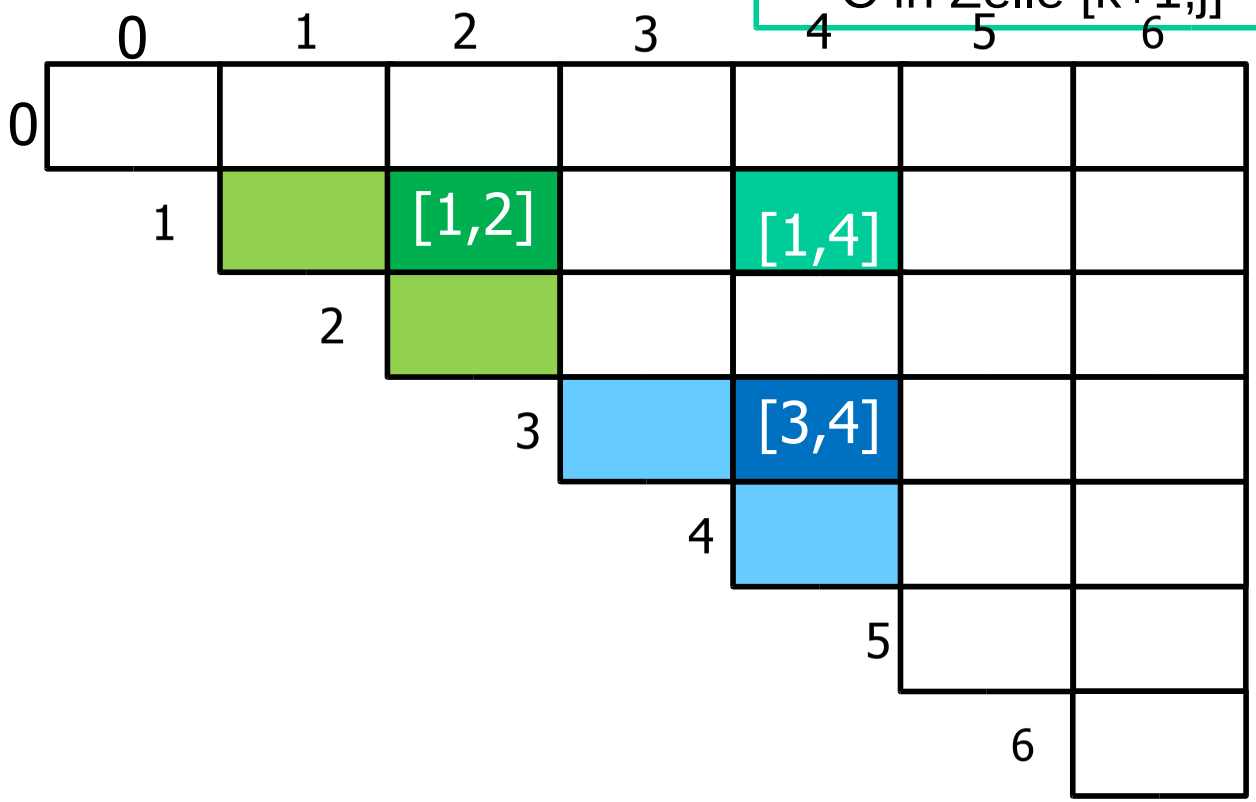
CYK Algorithmus: Beispiel

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$



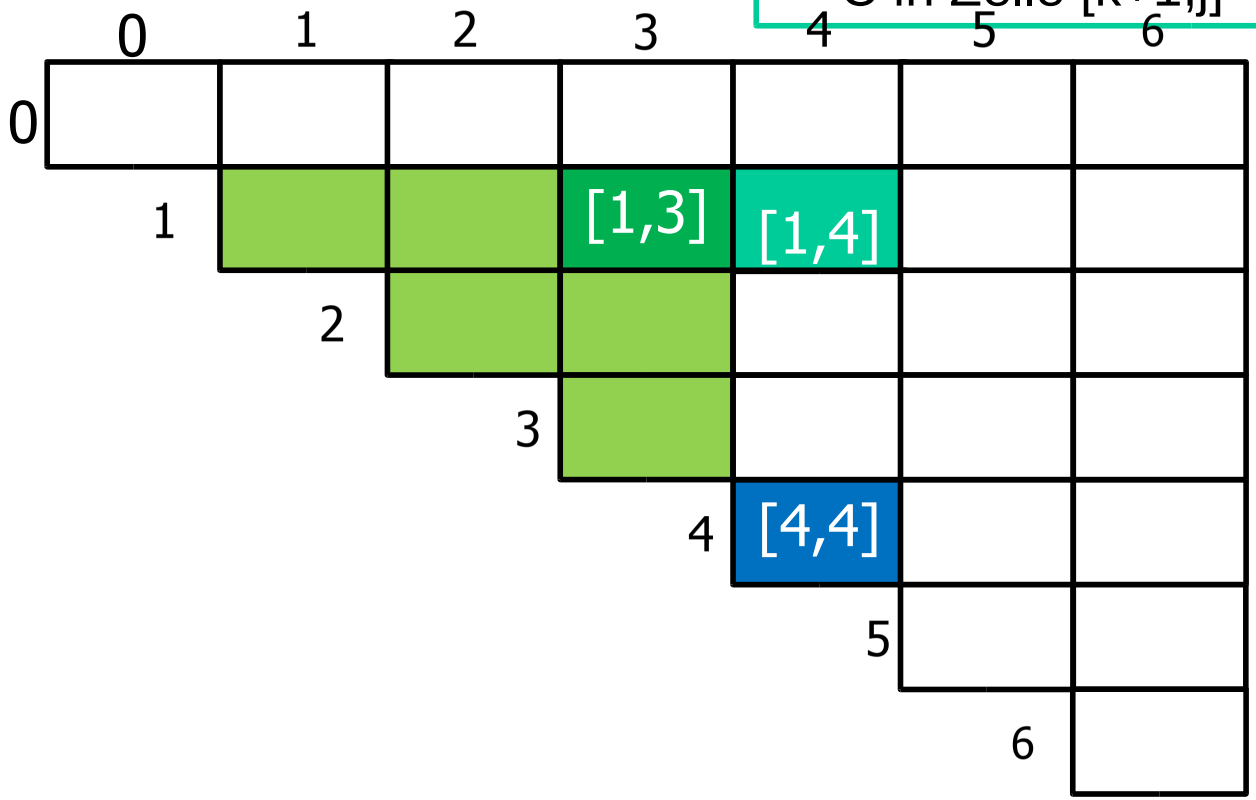
CYK Algorithmus: Beispiel

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$



CYK Algorithmus: Beispiel

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$



CYK: Algorithmus

	I (0)	shot (1)	an (2)	elephant (3)	in (4)	my (5)	pajamas (6)
0	N						
1		V					
2			DET	NP			
3				N			
4					P		
5						POSS	NP
6							N

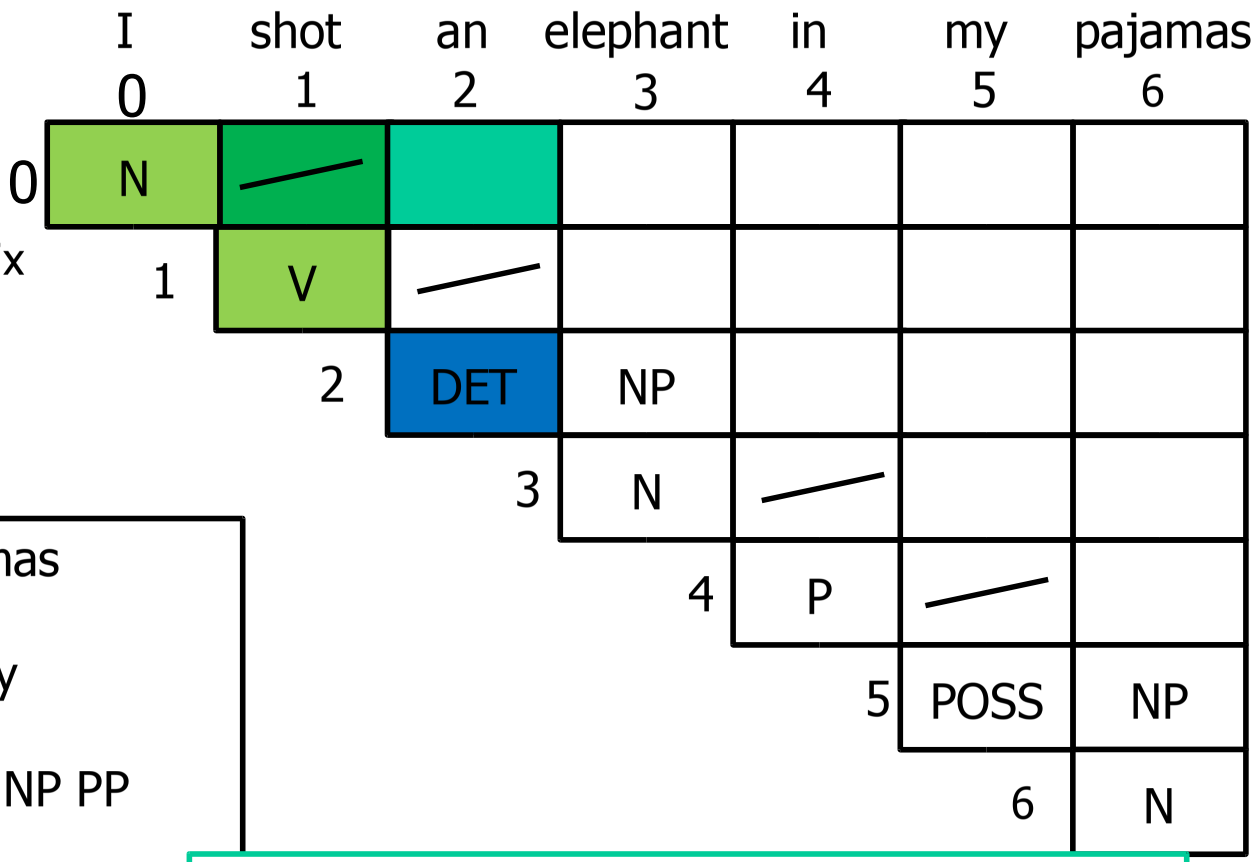
Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$

CYK: Algorithmus



Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$

CYK: Algorithmus

Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

	I (0)	shot (1)	an (2)	elephant (3)	in (4)	my (5)	pajamas (6)
0	N						
1		V					
2			DET	NP			
3				N			
4					P		
5						POSS	NP
6							N

N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$

CYK: Algorithmus

Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

	I 0	shot 1	an 2	elephant 3	in 4	my 5	pajamas 6
0	N						
1		V		VP			
2			DET	NP			
3				N			
4					P		
5						POSS	NP
6							N

N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$

CYK: Algorithmus

	I (0)	shot (1)	an (2)	elephant (3)	in (4)	my (5)	pajamas (6)
0	N						
1		V		VP			
2			DET	NP			
3				N			
4					P		
5						POSS	NP
6							N

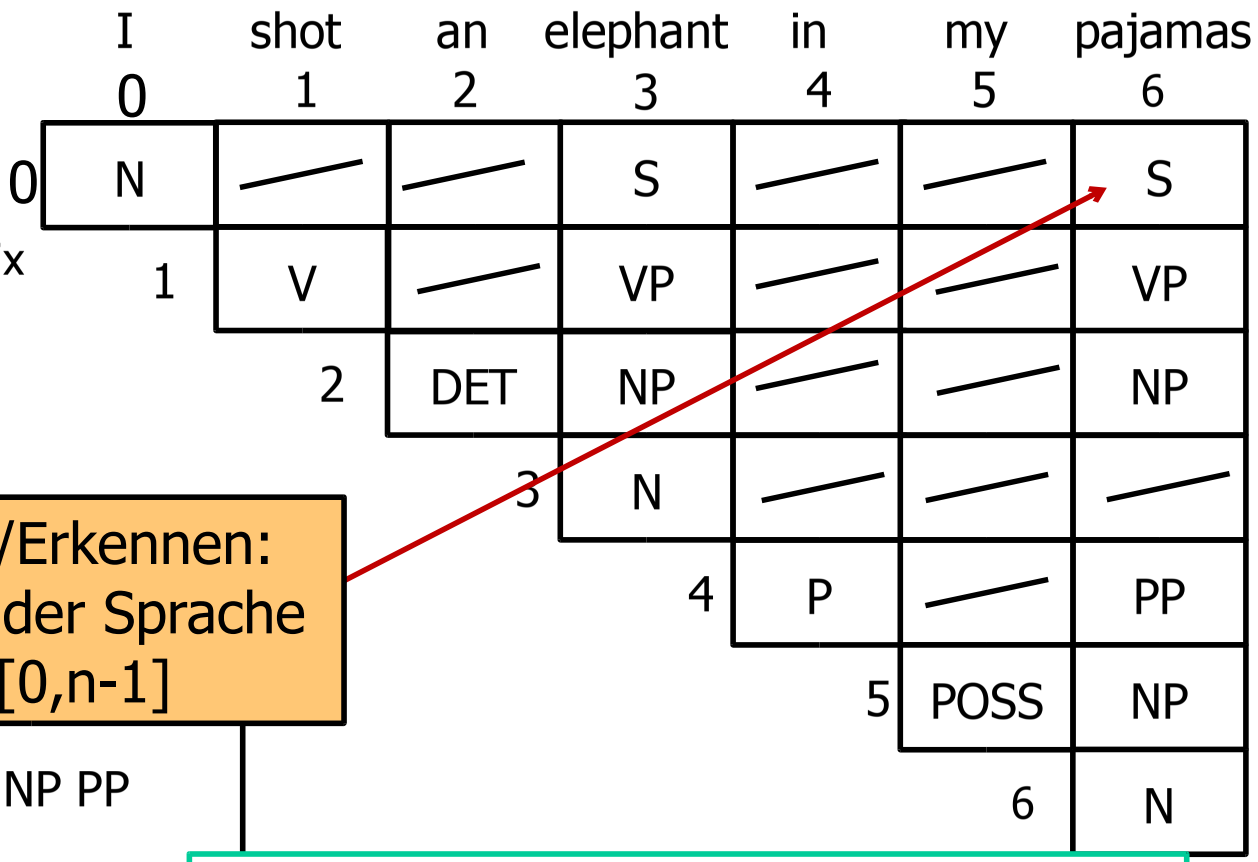
Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

N → I | elephant | pajamas
V → shot P → in
DET → an POSS → my

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$

CYK: Algorithmus



Datenstruktur
Satzlänge n=7 → 7x7 Matrix
→ wir nutzen die
obere Dreiecksmatrix
→ Notation: [Zeile, Spalte]

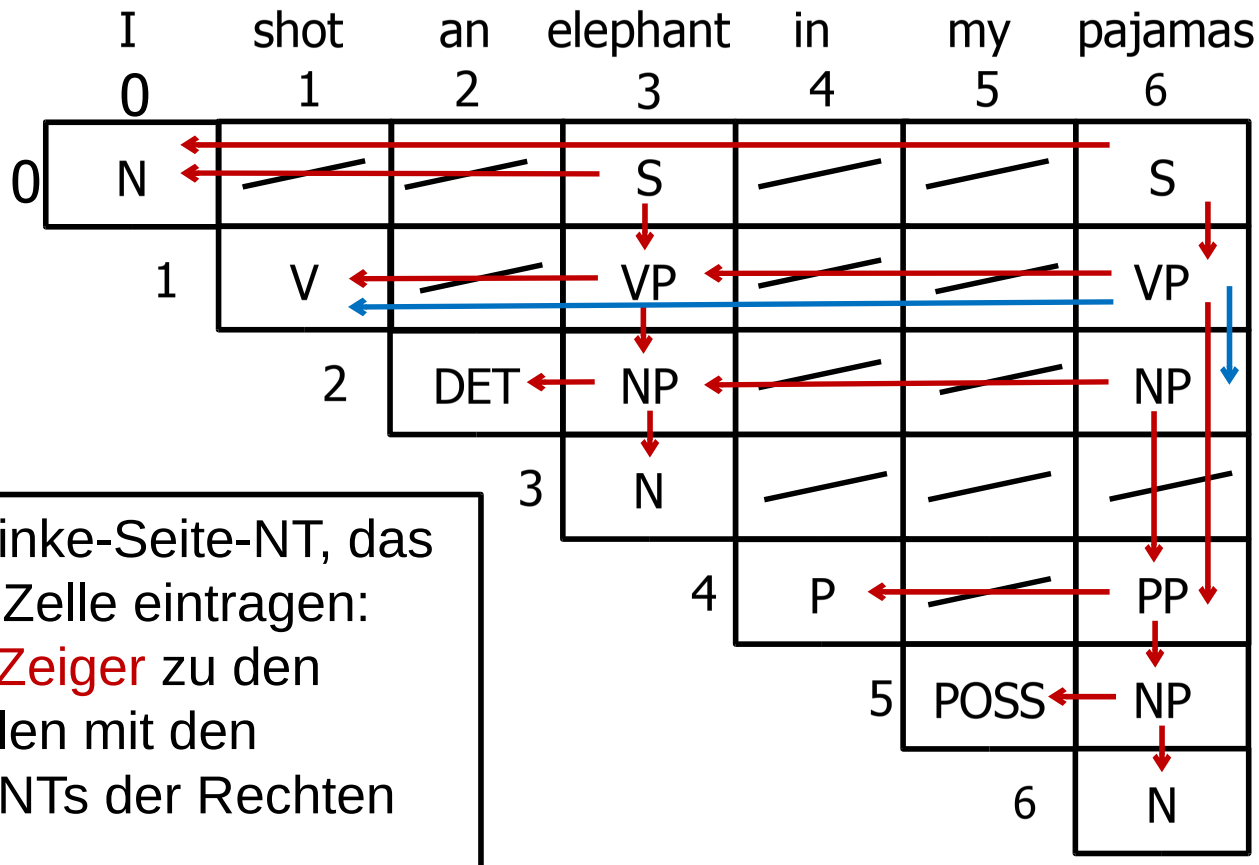
N → I | ele
V → shot
DET → an

NP → DET N | POSS N | NP PP
PP → P NP
S → N VP | NP VP
VP → V NP | VP PP

Recognize/Erkennen:
Satz ist in der Sprache
wenn S ∈ [0,n-1]

Schritt 2 bis Schritt n allgemein:
fülle jeweils die nächste Diagonale aus
trage A in Zelle [i,j] ein, falls
- es eine Regel A → B C gibt und
- B in Zelle [i,k] und
- C in Zelle [k+1,j] $i <= k < j$

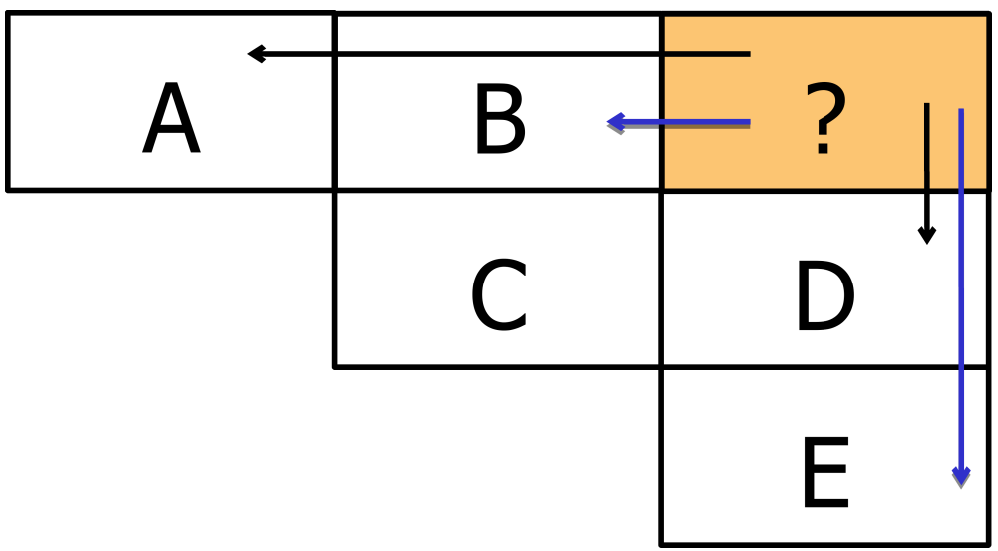
CYK: Algorithmus – Zeiger zu rechten Seiten



für jedes Linke-Seite-NT, das wir in eine Zelle eintragen: speichere Zeiger zu den beiden Zellen mit den jeweiligen NTs der Rechten Seite.
→ so können wir später die Parse-Bäume extrahieren, indem wir bei S in [0,n-1]

Bäume für dieses Beispiel:
an der Tafel

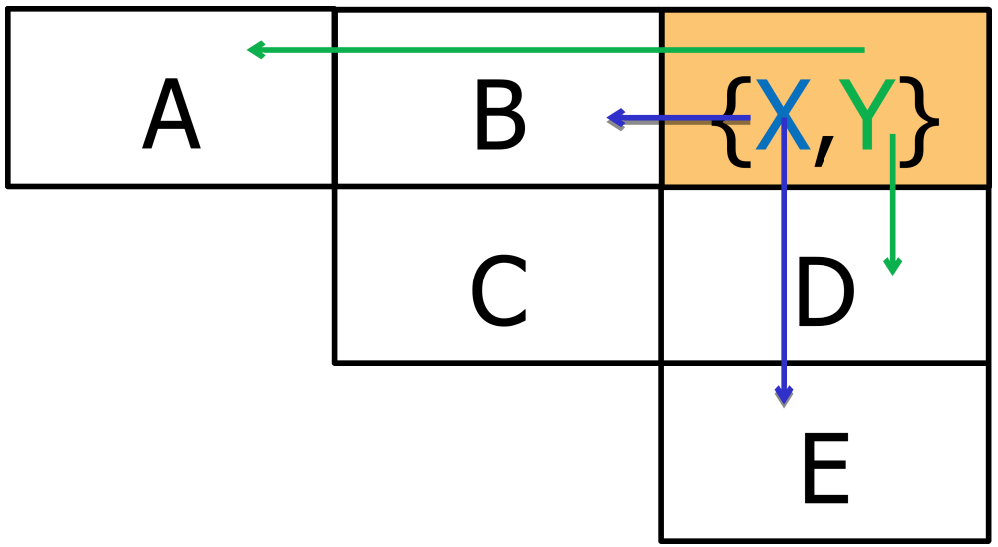
CYK: Mehrdeutigkeit in Zellen



Grammatik
X → BE
Y → AD

Eintrag für ?

CYK: Mehrdeutigkeit in Zellen



Grammatik
X → BE
Y → AD

Einträge in allen Zellen sind Mengen von Linke-Seite-Nichtterminalen (mit den jeweiligen Zeigern)

CYK Algorithmus: Eigenschaften

- Korrekt:
  Wenn $S \in [0, n-1]$, dann $S \Rightarrow^* w_1 \ldots w_n$
  - n Tokens in der Eingabe
- Vollständig:
  Wenn $S \Rightarrow^* w_1 \ldots w_n$, dann $S \in [0, n-1]$
- Laufzeit:
  Polynomiell in der Eingabelänge: $O(n^3)$