

# Programmierkurs Python II

## Vorlesung 7: Parsing II

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

Sommersemester 2011



## Übersicht

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

# 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  $\Rightarrow$  Suche & Backtracking
  - Identische Teilergebnisse werden u.U. mehrfach berechnet
  - $\Rightarrow$  Laufzeit exponentiell in der Eingabelänge (worst case)
- Lokale Ambiguität: Welche Regel bzw. Operation muss angewendet werden?

3

# Beispielgrammatik

$S \rightarrow NP VP$	$DET \rightarrow an$
$NP \rightarrow DET N$	$DET \rightarrow the$
$NP \rightarrow POSS N$	$POSS \rightarrow his$
$NP \rightarrow NP PP$	$N \rightarrow elephant$
$PP \rightarrow P NP$	$N \rightarrow pajamas$
$VP \rightarrow V NP$	$N \rightarrow boy$
$VP \rightarrow VP PP$	$V \rightarrow shot$
	$P \rightarrow in$

4

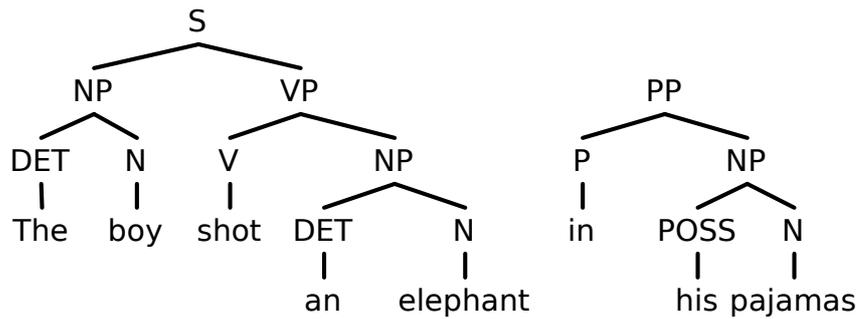
# The boy shot an elephant in ...

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

⇒\* ⟨[NP VP], [in his pajamas]⟩

⇒ ⟨[S], [in his pajamas]⟩

⇒\* ⟨[S PP], []⟩ ⇒ Backtracking



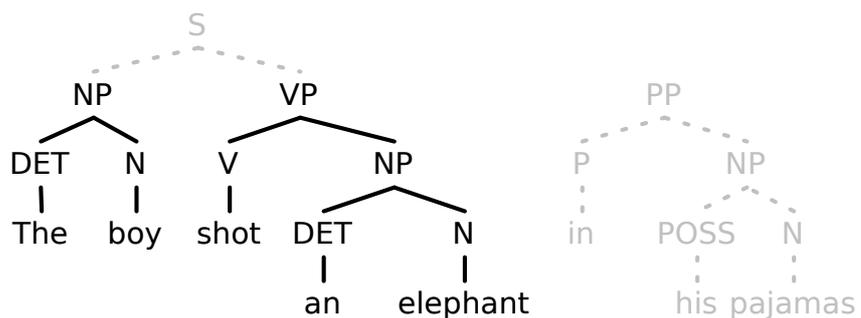
(Shift-Reduce)

5

# The boy shot an elephant in ...

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

⇒\* ⟨[NP VP], [in his pajamas]⟩



(Shift-Reduce)

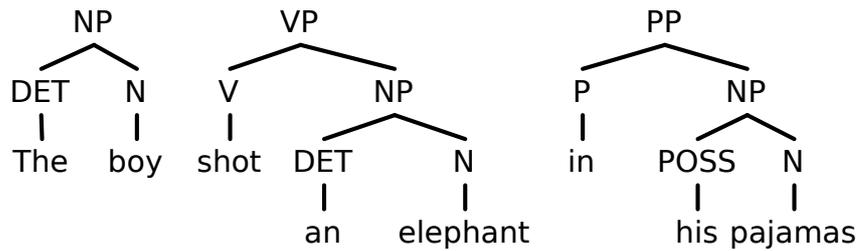
6

# The boy shot an elephant in ...

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

⇒\* ⟨[NP VP], [in his pajamas]⟩

⇒\* ⟨[NP VP PP], []⟩



(Shift-Reduce)

7

# The boy shot an elephant in ...

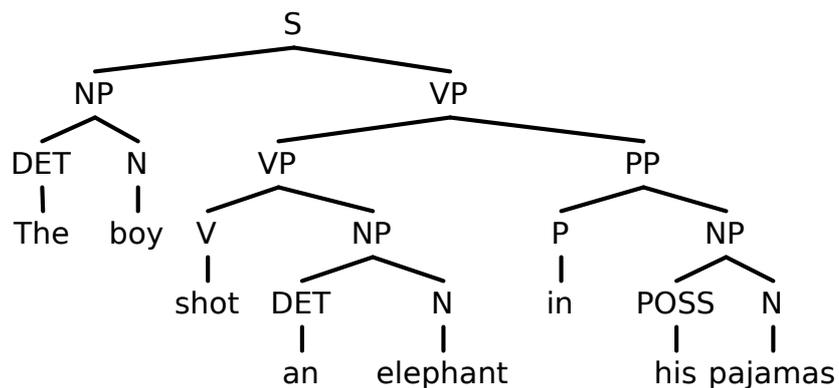
⟨[], [the boy shot an elephant in his pajamas]⟩

⇒\* ⟨[NP VP], [in his pajamas]⟩

⇒\* ⟨[NP VP PP], []⟩

⇒\* ⟨[NP VP], []⟩

⇒\* ⟨[S], []⟩

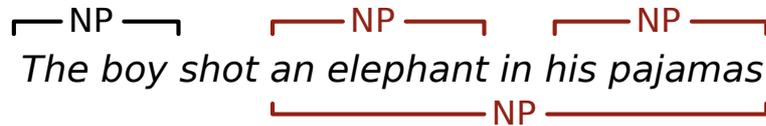


(Shift-Reduce)

8

# 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.

9

# 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 (⇒ Earley).
- **Verschiedene Chart-Parser:**
  - CYK, Earley, Bottom-up chart parser, ...

10

# Charts als Matrix

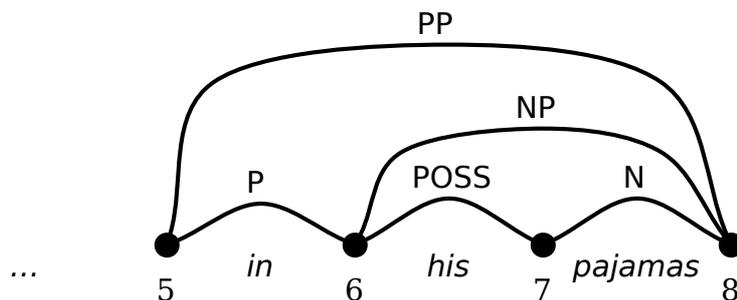
1	DET							
2	NP	N						
3	∅	∅	V					
4	∅	∅	∅	DET				
5	S	∅	VP	NP	N			
6	∅	∅	∅	∅	∅	P		
7	∅	∅	∅	∅	∅	∅	POSS	
8	S	∅	VP	NP	∅	PP	NP	N

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

$\begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \textit{The} & \textit{boy} & \textit{shot} & \textit{an} & \textit{elephant} & \textit{in} & \textit{his} & \textit{pajamas} \end{matrix}$

# Charts als Graph

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



# 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  $S \rightarrow \varepsilon$

13

# 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] }  
  return S ∈ T[0, n]
```

14

# Eigenschaften

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

15

# Variante (Kettenregeln)

$$\text{Closure}(A) = \{ B \mid B \Rightarrow^* A \}$$

CYK( $G, w_1 \dots w_n$ ):

for  $i$  in  $1 \dots n$ :

$T[i-1, i] = \{ \text{Closure}(A) \mid A \rightarrow w_i \in R \}$

for  $j$  in  $i - 2 \dots 0$ :

$T[j, i] = \emptyset$

for  $k$  in  $j + 1 \dots i - 1$ :

$T[j, i] = T[j, i] \cup$

$\{ \text{Closure}(A) \mid A \rightarrow B C, B \in T[j, k], C \in T[k, i] \}$

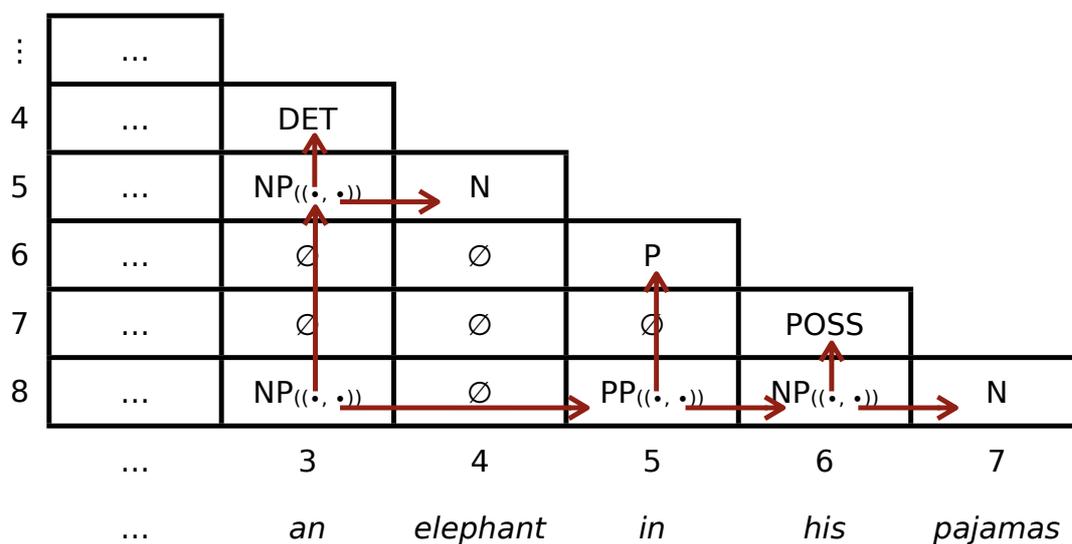
return  $S \in T[0, n]$

16

# 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)



# Grammatik $\Rightarrow$ CNF

## Linksbinarisierung(G):

while G enthält Regel  $A \rightarrow A_1 A_2 A_3 \dots A_k$ ,  $k \geq 3$   
entferne die Regel aus G  
neue Regel:  $\langle A_1, \dots, A_{k-1} \rangle \rightarrow A_1 \dots A_{k-1}$   
neue Regel:  $A \rightarrow \langle A_1, \dots, A_{k-1} \rangle A_k$

## Rechtsbinarisierung(G):

while G enthält Regel  $A \rightarrow A_1 A_2 A_3 \dots A_k$ ,  $k \geq 3$   
entferne die Regel aus G  
neue Regel:  $\langle A_2, \dots, A_k \rangle \rightarrow A_2 \dots A_k$   
neue Regel:  $A \rightarrow A_1 \langle A_2, \dots, A_k \rangle$

19

# Grammatik $\Rightarrow$ CNF

- **Anmerkung:** mit „CNF“ sind hier Grammatiken mit maximal zwei Nichtterminalen auf der rechten Regelseite gemeint (Kettenregeln sind also erlaubt).
  - $A \rightarrow w$
  - $A \rightarrow B$
  - $A \rightarrow B C$

20

# Implementierungsvarianten

- $T[i,j] = T[i,j] \cup \{ A \mid A \rightarrow B C, B \in T[i,k], C \in T[k,j] \}$ 
  - $\Rightarrow$  kann verschieden implementiert werden
- **Variante 1**
  - Iteriere über alle Regeln  $A \rightarrow B C$
  - Prüfe, ob  $B \in T[i,k]$  und  $C \in T[k,j]$
- **Variante 2**
  - Iteriere über alle  $B \in T[i,k]$
  - Iteriere über alle Regeln  $A \rightarrow B C$
  - Prüfe, ob  $C \in T[k, j]$

21

# Implementierungsvarianten

- $T[i,j] = T[i,j] \cup \{ A \mid A \rightarrow B C, B \in T[i,k], C \in T[k,j] \}$ 
  - $\Rightarrow$  kann verschieden implementiert werden
- **Variante 3**
  - Iteriere über alle  $C \in T[k,j]$
  - Iteriere über alle Regeln  $A \rightarrow B C$
  - Prüfe, ob  $A \in T[i,k]$
- **Variante 4**
  - Iteriere über alle  $B \in T[i,k]$  und  $C \in T[k,j]$
  - Prüfe, ob es Regel  $A \rightarrow B C$  gibt

22

## Song &al. (2008)

- Experimente mit CYK & Wall Street Journal
- Laufzeit abhängig von Binarisierungsmethode
  - Rechtsbinarisierung  $\Rightarrow$  Variante 3 am effizientesten
  - Linksbinarisierung  $\Rightarrow$  Variante 2 am effizientesten

23

## CYK

- CYK ist eingeschränkt auf Grammatiken in CNF
  - keine echte Einschränkung, Grammatiken können in CNF überführt werden
- CYK ist ein Bottom-Up-Parser und erzeugt unter Umständen viele nicht benötigte Konstituenten
- *Mary gave the man a book*
  - $N \rightarrow \text{man}$
  - $V \rightarrow \text{man} \Rightarrow$  CYK leitet für „man a book“ eine VP ab

24

# Earleys Algorithmus

- Ein effizienter „aktiver“ Chart-Parser
  - Zeikomplexität  $O(n^3)$  in der Eingabelänge  $n$
- Für beliebige kontextfreie Grammatiken geeignet
  - Tilgungsregeln
  - Zyklische Kettenregeln
  - Links- und rechtsrekursive Regeln
- Mischung aus bottom-up und top-down
  - Analyserichtung links  $\rightarrow$  rechts

25

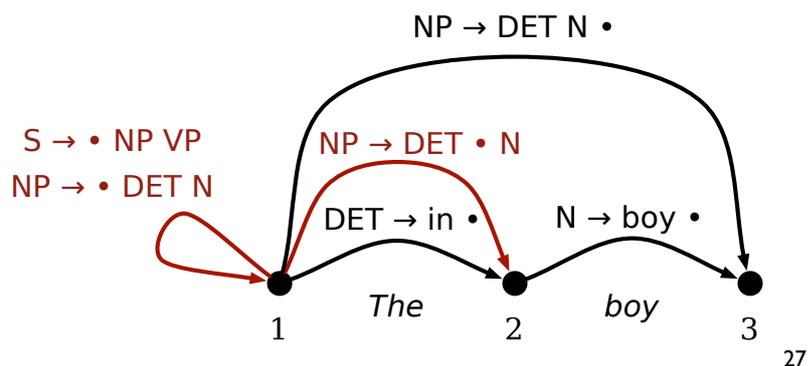
# Earleys Algorithmus

- Passive Chart-Parser speichern nur vollständige Konstituenten als Teilergebnis in der Chart.
- Aktive Chart-Parser speichern auch Hypothesen.
- $A \rightarrow \mathbf{B_1 \dots B_n} \cdot \mathbf{C_{n+1} \dots C_k}$ 
  - $\mathbf{B_1 \dots B_2}$  wurden bereits gefunden
  - $\mathbf{C_{n+1} \dots C_k}$  müssen noch gefunden werden

26

# Die Chart

- **Ecken** („Knoten“) entsprechen Positionen in der Eingabe
- **Kanten** entsprechen gefundenen Teilkonstituenten. Auch unvollständige Konstituenten werden gespeichert.
  - Aktive Kanten  $\langle A \rightarrow \alpha \cdot B \beta, i, j \rangle$
  - Passive Kanten  $\langle A \rightarrow \alpha \cdot, i, j \rangle$

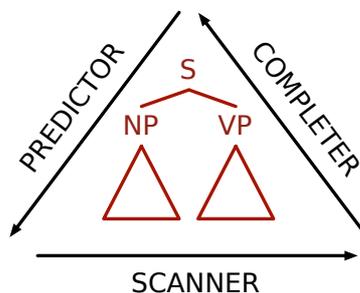


27

# Earleys Algorithmus

- **Drei Prozeduren**

- Predictor
- Scanner
- Completer

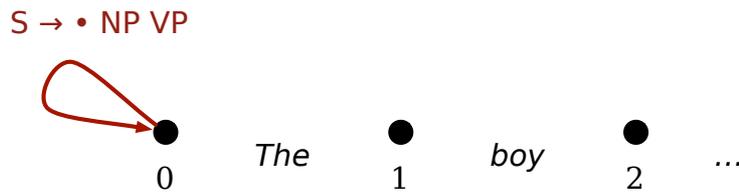


- Die drei Prozeduren werden für jeden Knoten in der Chart von links nach rechts aufgerufen.
- $\Rightarrow$  jeweils solange, bis keine neuen Kanten zur Chart hinzugefügt werden

28

# Initialisierung

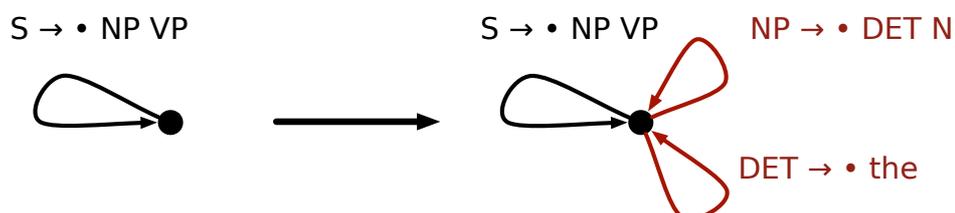
- Grammatik  $G = \langle V, \Sigma, R, S \rangle$
- Eingabe  $w_1 \dots w_n$
- Für jede Regel  $S \rightarrow \alpha$ 
  - Füge  $\langle S \rightarrow \cdot \alpha, 0, 0 \rangle$  zur Chart hinzu



29

# Predictor

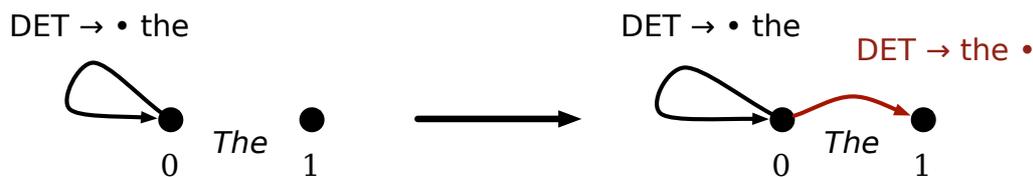
- Der Predictor sagt voraus, welche Grammatikregeln zum Ziel führen könnten (top-down).
- Für neue Kanten  $\langle A \rightarrow \alpha \cdot B \beta, i, j \rangle$ 
  - Für alle Regeln  $B \rightarrow \gamma \in \text{Regeln}$
  - Neue Kante:  $\langle B \rightarrow \cdot \gamma, j, j \rangle$



30

# Scanner

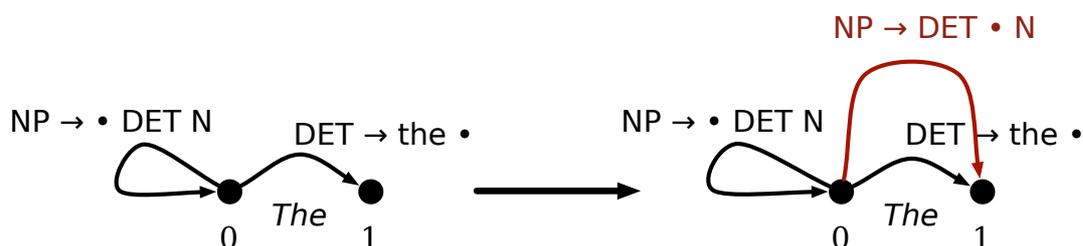
- Der Scanner konsumiert Terminalsymbole aus der Eingabekette.
- Für neue Kanten  $\langle A \rightarrow \alpha \cdot w_{j+1} \beta, i, j \rangle$ 
  - Neue Kante  $\langle A \rightarrow \alpha w_{j+1} \cdot \beta, i, j + 1 \rangle$



31

# Completer

- Der Completer vervollständigt aktive Kanten durch Kombination mit inaktiven Kanten (bottom-up).
- Für jede neue Kante  $\langle A \rightarrow \alpha \cdot, j, k \rangle$ 
  - Für jede alte Kante  $\langle B \rightarrow \gamma \cdot A \beta, i, j \rangle$ 
    - Neue Kante:  $\langle B \rightarrow \gamma A \cdot \beta, i, k \rangle$



32

# Übersicht

- **Scan**
  - $\langle A \rightarrow \alpha \cdot w_{j+1} \beta, i, j \rangle \Rightarrow \langle A \rightarrow \alpha w_{j+1} \cdot \beta, i, j + 1 \rangle$
- **Predict**
  - $\langle A \rightarrow \alpha \cdot B \beta, i, j \rangle, B \rightarrow \gamma \Rightarrow \langle B \rightarrow \cdot \gamma, j, j \rangle$
- **Complete**
  - $\langle A \rightarrow \alpha \cdot B \beta, i, j \rangle, \langle B \rightarrow \gamma \cdot, j, k \rangle \Rightarrow \langle A \rightarrow \alpha B \cdot \beta, i, k \rangle$

33

# Implementierung

- Kanten  $\langle A \rightarrow \alpha \cdot B \beta, i, j \rangle$  stellen wir dar als  $(A, [B, \dots], i, j)$ , d.h.  $\alpha$  wird nicht explizit repräsentiert.
- Der Algorithmus (Skizze)
  - Auf einer Agenda (Liste) werden durch Anwendung von Scan, Predict, Complete erzeugte Kanten (zwischen-) gespeichert
    - Die Agenda enthält initial die Kante  $(\text{START}, [S], 0, 0)$
  - In jedem Schritt entfernen wir eine Kante von der Agenda, wenden Scan, Predict oder Complete an und speichern die Kante abschließend in der Chart (Liste)
  - Durch Anwendung von Scan, Predict, Complete erzeugte Kanten werden auf der Agenda gespeichert

34

## Implementierung

```
rules = [ ('S', ['NP', 'VP']),
          ('NP', ['DET', 'N']),
          ('VP', ['V', 'NP']),
          ('VP', ['V']),
          ('DET', ['der']),
          ('DET', ['das']),
          ('N', ['Student']),
          ('N', ['Buch']),
          ('V', ['liest']),
          ('V', ['arbeitet'])
        ]
```

35

## Implementierung

```
def earley(start, rules, words):
    length = len(words)
    chart = []
    agenda = [('START', [start], 0, 0)]
    while agenda:
        (lhs, rhs, i, j) = agenda.pop()
        # scan, predict, complete (nächste Folie)
        chart.append((lhs, rhs, i, j))
    return ('START', [], 0, length) in chart
```

36

# Implementierung

```
def earley(start, rules, words):  
    ...  
    while agenda:  
        (lhs, rhs, i, j) = agenda.pop()  
        if rhs:  
            if j < length and rhs[0] == words[j]: # scan  
                agenda.append((lhs, rhs[1:], i, j + 1))  
            else: # predict  
                for (_lhs, _rhs) in rules:  
                    if rhs[0] == _lhs:  
                        agenda.append((_lhs, _rhs, j, j))  
        else:  
            ...
```

37

# Implementierung

```
def earley(start, rules, words):  
    ...  
    while agenda:  
        (lhs, rhs, i, j) = agenda.pop()  
        if rhs:  
            ...  
        else: # complete  
            for (_lhs, _rhs, _i, k) in chart:  
                if k == i and _rhs and _rhs[0] == lhs:  
                    agenda.append((_lhs, _rhs[1:], _i, j))  
            chart.append((lhs, rhs, i, j))  
    return ('START', [], 0, length) in chart
```

38

# Variante

- Im hier vorgestellten Verfahren wird nicht zwischen Lexikon und „eentlichen“ Regeln unterschieden.
- Regeln wie „DET → the“ werden vom Predictor behandelt
  - ⇒ es können Terminalsymbole vorhergesagt werden, die gar nicht in der Eingabekette vorkommen (ineffizient)
- **Sinnvolle Variante:**
  - Regeln wie „DET → the“ vom Scanner verarbeiten lassen:
  - Für neue Kanten  $\langle A \rightarrow \alpha \cdot B \beta, i, j \rangle$ 
    - Neue Kante  $\langle A \rightarrow \alpha B \cdot \beta, i, j + 1 \rangle$  wenn  $B \rightarrow w_{i+1}$