# CKY und Earley Parser

Katja Markert (mit einigen Folien von Yannick Versley)
Institut für Computerlinguistik

# Parsing

- Bisher: ***kontextfreie Grammatiken (CFG)***
- **Jetzt: Parsing:** Wie kann – gegeben eine CFG – für einen Satz *automatisch*
  - das *Erkennungsproblem* gelöst werden?
  - die *syntaktische Analyse* des Satzes durchgeführt werden?

  **Eingabe**: ein Satz und eine formal definierte CFG

  **Ausgabe**:

  *ein oder mehrere Phrasenstrukturbäume* falls grammatisch

  *False* falls ungrammatisch

- **Später:** Wie gehen wir mit *Ambiguitäten* der syntaktischen Analyse um?

# Warum Parsing?

-- Grammar checking

– als **Zwischenstufe für „tiefe“ semantische Verarbeitung**
z.B. Frage-Antwort-Systeme, Informationsextraktion

Hier genügt es oftmals zu ermitteln, wer das Subjekt/Objekt eines Ereignisverbs ist, oder wann ein Ereignis stattgefunden hat:

*[$_S$ [$_{NP}$ Who] [$_{VP}$ won [$_{NP}$ the Nobel Prize [$_{PP}$ for Physics]] in 2007]?*

*[$_S$ [$_{NP}$ France's Albert Fert and German Peter Gruenberg] [$_{VP}$ win [$_{NP}$ the 2007 Nobel Prize [$_{PP}$ in Physics]] it was announced Tuesday Oct. 9, 2007*

# Grundlegende Parsingalgorithmen

- Naive Methoden
  - Top-down Parsing (eine Version: Recursive Descent Parsing)
  - Bottom-up Parsing
- Probleme: Effizienz (und Ambiguitäten)
- Chart Parsing (dynamische Programmierung)
  - CKY Cocke-Kasami-Younger (bottom-up)
  - Earley (top-down)

- Daneben gibt es viele Varianten für Optimierungen.

# Grammatik und Parsebaum

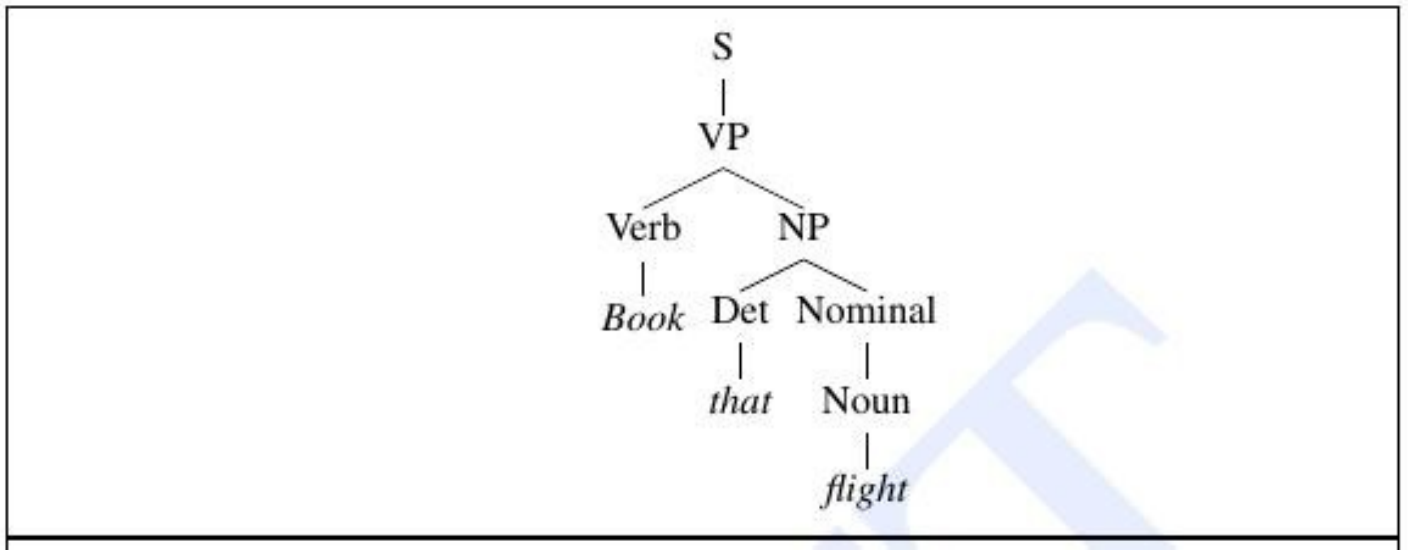
| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 12.1** The $\mathscr{L}_1$ miniature English grammar and lexicon.

Beispiegrammatk L1 aus Jurafsky und Martin, 3rd edition

*Ziel des Parsingalgorithmus*: Finde ***alle*** Bäume mit Startsymbol S, die den gesamten Eingabestring überspannen

# Grammatik und Parsebaum




Beispiel aus Jurafsky und Martin, 2nd edition

*Ziel des Parsingalgorithmus*: Finde ***alle*** Bäume mit Startsymbol S, die den gesamten Eingabestring überspannen

# Parsingstrategien

- Zwei Informationsquellen beschränken die Suche
  - Eingabesatz:
    Jeder Parsebaum muss die Wörter des Eingabesatzes als Blätter (Terminalsymbole) enthalten
  - Startsymbol der Grammatik:
    Jeder Parsebaum muss Startsymbol als Wurzelknoten besitzen

==> zwei grundlegende Parsingstrategien

- **Top-down Suche** (zielgetriebene Suche)
- **Bottom-up Suche** (datengetriebene Suche)

# Top-Down Parsing

- Ein *Top-down Parser: ausgehend vom Wurzelknoten* (Startsymbol S) Expansion von Grammatikregeln
- Die Suche ist *erfolgreich*, wenn die *Blätter des Parsebaums* genau mit den *Wörtern des Eingabestrings* übereinstimmen
- Algorithmus (recursive descent)
  1. Startannahme: Der Baum beginnt mit dem Wurzelknoten X = S
  2. Finde *alle* Regeln $R_X$ mit Symbol X auf der linken Regelseite
  3. Für jede Regel $R_X$
     3.1. setze die Kategorien der rechten Regelseite als Tochterknoten $D_x$ von X ein
     3.2. für jeden Tochterknoten $D_x$: suche Bäume mit Wurzelknoten X = $D_x$

Recursive Descent: top-down, left-right, erschöpfende Suche mit Backtracking

# Recursive Descent am Beispiel “Book that flight” (top-down, left-right,depth first, backtracking)

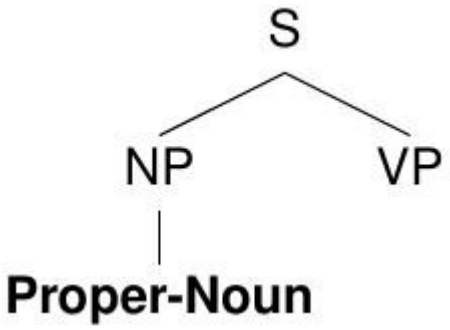

| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 12.1** The $\mathscr{L}_1$ miniature English grammar and lexicon.

Backtracking nötig, da POS mismatch

book

# Recursive Descent am Beispiel “Book that flight”



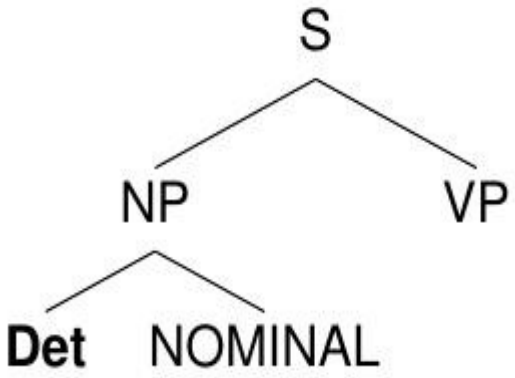

| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 12.1** The $\mathscr{L}_1$ miniature English grammar and lexicon.

Backtracking nötig, da POS mismatch

book

# Recursive Descent am Beispiel “Book that flight”



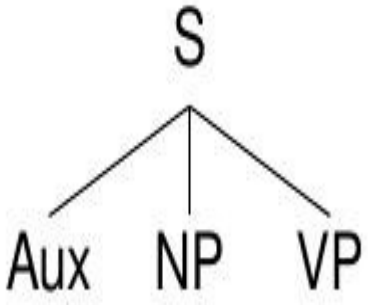

| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 12.1** The $\mathscr{L}_1$ miniature English grammar and lexicon.

Backtracking nötig, da POS mismatch

book

# Recursive Descent am Beispiel “Book that flight”




| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 12.1** The $\mathscr{L}_1$ miniature English grammar and lexicon.

Backtracking nötig, da POS mismatch

book

# Recursive Descent am Beispiel “Book that flight”

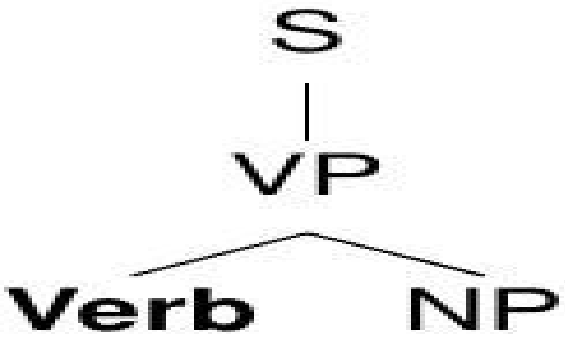

| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

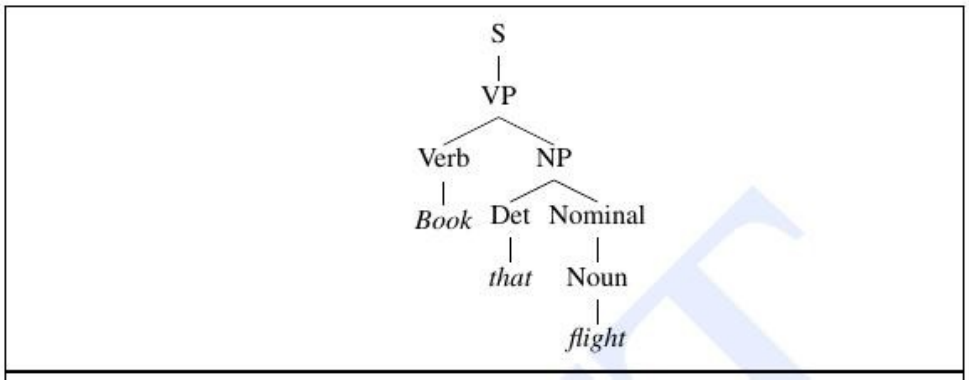
**Figure 12.1** The $\mathscr{L}_1$ miniature English grammar and lexicon.

Backtracking, da zuviel Input übrig

book

# Recursive Descent am Beispiel “Book that flight”




| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid the \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 12.1** The $\mathcal{L}_1$ miniature English grammar and lexicon.

Endlich (nach zwei weiteren Backtrackings an NP)

book

# Top-Down Parsing Invarianten

Wenn der (gefundene) Parsebaum die Expansion
A ->B C enthält

- Wenn B Teil des gefundenen (Teil-)Baumes ist,
  ist auch A Teil des gefundenen (Teil-)Baumes (**Top-Down**)
- Wenn C Teil des gefundenen (Teil-)Baumes ist,
  ist auch B Teil des gefundenen (Teil-)Baumes (**Left to right**)

Wenn der gefundene Parsebaum eine Expansion
A -> w (für ein Terminal w) enthält,
ist w Teil der Eingabe an dieser Stelle.

# Top-Down Parsing Terminierung

- Terminierung (oder: Abgleich mit der Realität)
  - Die *Blätter* des generierten Syntaxbaums werden mit den *Wörtern des Eingabestrings* abgeglichen.
  - Ist das aktuelle Eingabewort $w_i$ *identisch* mit dem Terminalsymbol $D_i$, ist die verwendete Regel ***erfolgreich.***

- Insgesamt:
  - Können die Blätter des konstruierten Parsebaums mit den Wörtern des Eingabestrings *vollständig abgeglichen* werden, ist der konstruierte Baum ein ***valider Parsebaum.***

# Recursive Descent Probleme: Rekursion

S → NP
NP → NP PP
NP → NNS
PP → IN NP
NNS → {men,women,tables,hats,feathers,birds}
IN → {with,for,against,of}

Parsen Sie damit Top-Down *men with hats*

*Probleme?*

*Lösungen?*

# Recursive Descent Probleme: Rekursion

Eine linksrekursive Grammatik hat Regeln der Form A → AX für mindestens ein A

Dies kann auch impliziter geschehen. Eine Grammatik, die die folgenden Regeln enthält, führt auch zu Linksrekursion.

A → BX
B → AY

Dann führt Recursive Descent zu einer unendlichen Schleife.

Linksrekursion kann durch geschicktes Schreiben von Grammatiken vermieden werden. Dies kann man auch automatisieren:

| | | |
|---|---|---|
| NP → NP PP | äquivalent zu | NP → NNS T |
| NP → NNS | | T → PP T |
| | | T → ε |

# Recursive Descent Probleme: Ambiguität

Wie behandelt der Top-Down-Parser ambige Sätze?

Lösungen?

# Recursive Descent Probleme: Effizienz

- Fehlschläge kann man stets erst dann feststellen, wenn die Blätter des Baums mit der Eingabe abgeglichen werden.
- **Beim systematischen ‚Rücksetzen' müssen identische Teilstrukturen immer wieder berechnet werden.**
- Kein wirklich effizientes Verfahren!
- Probleme mit Part-of-speech Matching können durch bottom-up Kombination (left-corner parsing) vermieden werden, andere Ineffizienz aber nicht…

# Recursive Descent Parsing Probleme: Effizienz

Beispiel für Ineffizienz trotz gutem POS matching mit einer NP-Grammatik:

S → NP

NP → AT NN

NP → NP PP

NP → NN

PP → IN NP

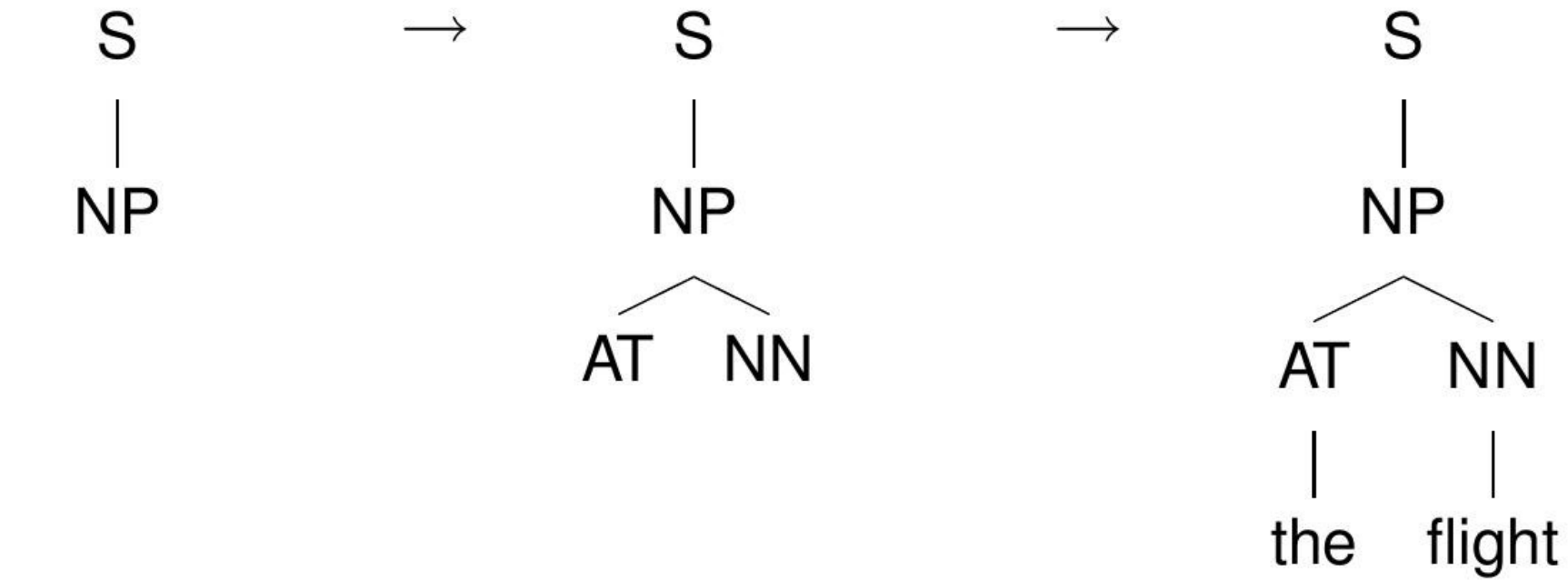
NN → {flight, London, Houston}

IN → {with, for, against, to, of}

AT → {the,a}

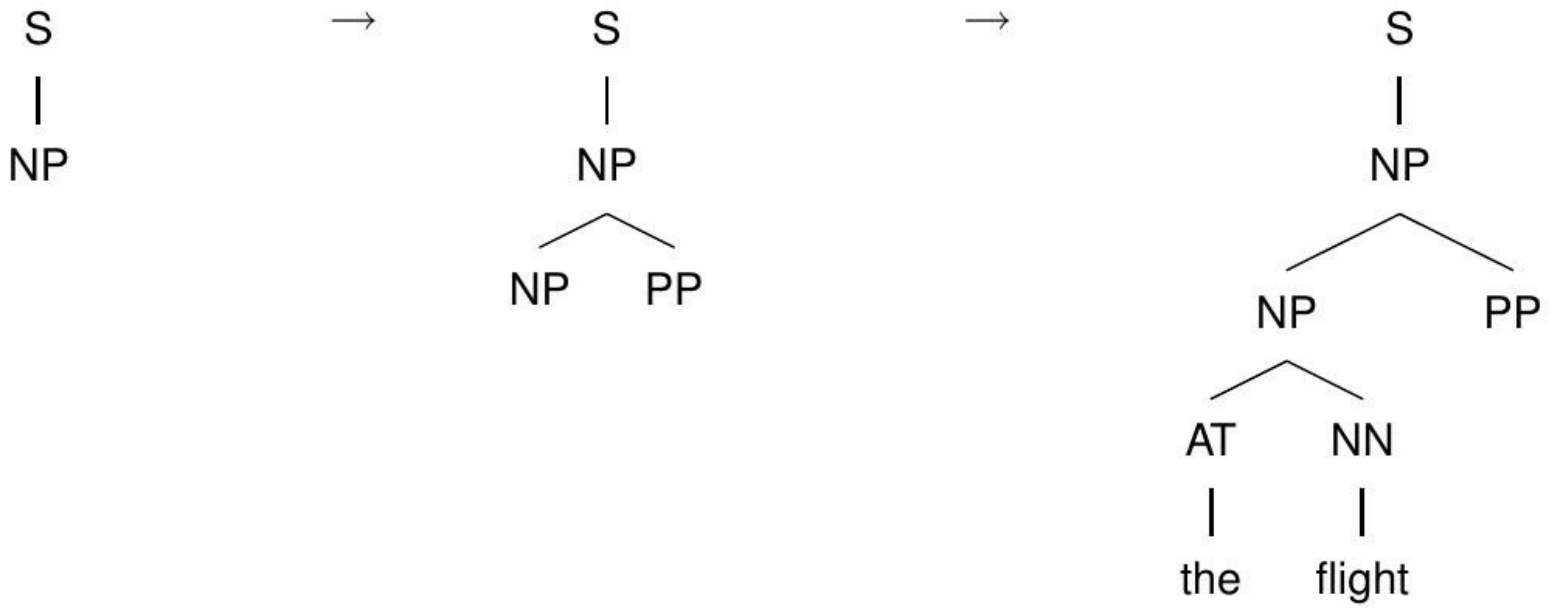
# Recursive Descent Parsing Probleme: Effizienz

*The flight to London*




PP kann nicht inkorporiert werden → backtrack

# Recursive Descent Parsing Probleme: Effizienz




NP für “the flight” doppelt abgeleitet! Komplexität im schlimmsten Fall exponentiell zur Eingabelänge

# Recursive Descent Parsing mit einer einfachen CFG

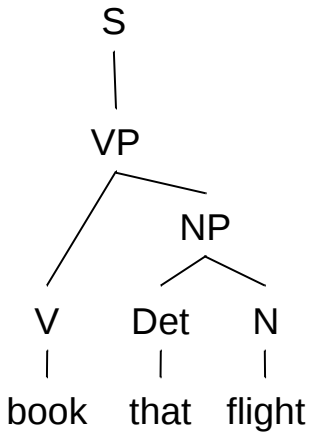
```
>> nltk.draw.rdparser.demo()
```

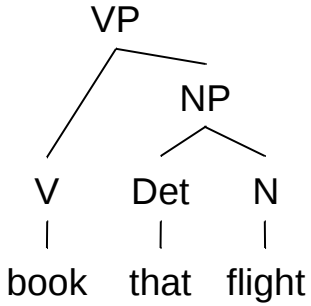
- Sie können Eingabesatz und Grammatikregeln modifizieren und den Parser bei der Suche nach einer gültigen Ableitung beobachten.

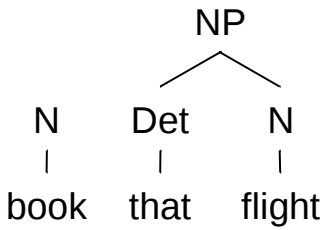
# Bottom-up Parsing

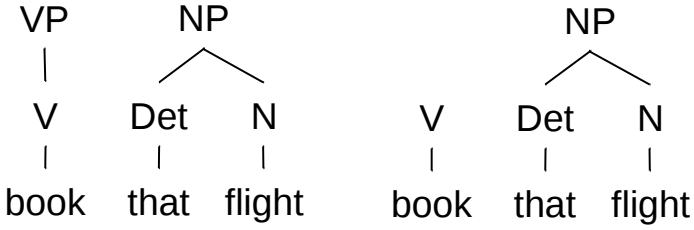
- *ausgehend von der Eingabesequenz* auf Basis von Ersetzungsregeln Teilbäume generieren, bis das *Startsymbol* der Grammatik erreicht ist.
- *erfolgreich*, wenn der Parsebaum die gesamte Eingabekette überspannt und einen einzigen Wurzelknoten enthält, der mit dem Startsymbol übereinstimmt.
- Algorithmus
  1. Startannahme: Die Blätter des Baumes sind identisch mit den Wörtern der Eingabesequenz
  2. Finde *alle* Regeln $R_{X,}$deren rechte Regelseiten auf die Wurzelkategorien der generierten Teilbäume passen
  3. Für jede Regel $R_X$:
     3.1. generiere einen Mutterknoten $M_x$ für die Kategorie der linken Regelseite von $R_x$ der die Tochterkategorien der rechten Regelseite im Baum dominiert
     3.2.

# Bottom-up Parsing





book that flight

# Bottom-Up-Parsing

**Invarianten** von Bottom-Up-Parsing:
Wenn der (gefundene) Parsebaum die Expansion
A ->B C enthält

- Wenn A Teil des gefundenen (Teil-)Baumes ist,
  ist auch B Teil des gefundenen (Teil-)Baumes (**Bottom-Up**)
- Wenn C Teil des gefundenen (Teil-)Baumes ist,
  ist auch B Teil des gefundenen (Teil-)Baumes (**Leftmost**)

Wenn der gefundene Parsebaum eine Expansion
A -> w (für ein Terminal w) enthält,
ist w Teil der Eingabe an dieser Stelle.

# Top-down und bottom-up Parsing: Vor- und Nachteile

- Vorteile
  - Der Top-down Parser exploriert keine ungrammatikalischen Bäume
  - Der Top-down Parser exploriert keine Bäume die keinen Platz in einem Baum mit Wurzelknoten S finden können.
  - Der Bottom-up Parser exploriert nur solche Bäume, die mit der Eingabesequenz kompatibel sind.
- Nachteile
  - Der Bottom-up Parser generiert viele Bäume, die keinerlei Aussicht haben, in einen Wurzelknoten S zu münden.
  - Der Top-down Parser generiert viele Bäume, die keinerlei Aussicht haben, auf die Eingabewörter zu passen.
- Viele Teilbäume sind nicht Teil einer vollständigen validen Analyse.
- Viele Teilbäume werden wiederholt berechnet. Effiziente Verfahren: speichere Teilresultate (→ dynamic programming)

# Fazit 1. Teil

Mit den Inhalten aus diesen Folien können Sie:

- Gegeben eine CFG zeigen, wie ein Bottom-Up oder Top-Down-Parser durch **Backtracking** zu Ableitungen kommt.
- Erklären, warum Backtracking für ambige Grammatiken ineffizient ist.

# *Computerlinguistik*

*– Parsing: Earley und CKY Algorithmus*

# Chart Parsing

- **Chart Parsing. (Beispiele Earley, CYK)**
- **Grundidee**
  - **Gesamtparse besteht aus validen Teilbäumen**
  - **Dynamische Programmierung***:*
    Speichern von Lösungen für Teilprobleme in Chart zur Vermeidung wiederholter Berechnungen → Polynomiale Laufzeit
- **Chart**: Tabelle, in der Teilresultate der Analyse abgelegt werden
  - Die Einträge in der Chart heißen *Items* oder *Kanten (edges)*
  - Ein Eintrag in der Chart enthält mindestens die folgende Information
    - Den **Satzabschnitt**, auf den sich das Item bezieht (Anfangs- und Endposition des Abschnitts, in Wortzählung)
    - Die **Syntaxregel**, die angewendet wurde

# Bottom-Up Chart Parsing (CKY/CYK)

- Eingabe: $_0$ *Book* $_1$ *the* $_2$ *flight* $_3$ *to* $_4$ *Houston* $_5$
- Ein Item der Chart: PP[3,5]
  - Überspannt den well-formed substring: *to Houston*
  - NP erkannt/erstellt durch Anwendung der Regel: PP -> Preposition NP
    - Kann sich nur zusammensetzen aus [3,4] und [4,5]
- CYK-Algorithmus: Erstellen von Chart-Items (Kasami 1965; Younger, 1967; Cocke und Schwarz 1970)
- CYK arbeitet nur mit Grammatiken in Chomsky Normalform, also Regeln der Form A-> B C oder A→β. Übung: wie konvertieren wir Beispielgrammatik L1 aus J&M nach CNF?

# Unsere Beispielgrammatik (links in CNF)

| | |
|---|---|
| $S \rightarrow NP\ VP$ | $S \rightarrow NP\ VP$ |
| $S \rightarrow Aux\ NP\ VP$ | $S \rightarrow X1\ VP$ |
| | $X1 \rightarrow Aux\ NP$ |
| $S \rightarrow VP$ | $S \rightarrow book \mid include \mid prefer$ |
| | $S \rightarrow Verb\ NP$ |
| | $S \rightarrow X2\ PP$ |
| | $S \rightarrow Verb\ PP$ |
| | $S \rightarrow VP\ PP$ |
| $NP \rightarrow Pronoun$ | $NP \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $NP \rightarrow TWA \mid Houston$ |
| $NP \rightarrow Det\ Nominal$ | $NP \rightarrow Det\ Nominal$ |
| $Nominal \rightarrow Noun$ | $Nominal \rightarrow book \mid flight \mid meal \mid money$ |
| $Nominal \rightarrow Nominal\ Noun$ | $Nominal \rightarrow Nominal\ Noun$ |
| $Nominal \rightarrow Nominal\ PP$ | $Nominal \rightarrow Nominal\ PP$ |
| $VP \rightarrow Verb$ | $VP \rightarrow book \mid include \mid prefer$ |
| $VP \rightarrow Verb\ NP$ | $VP \rightarrow Verb\ NP$ |
| $VP \rightarrow Verb\ NP\ PP$ | $VP \rightarrow X2\ PP$ |
| | $X2 \rightarrow Verb\ NP$ |
| $VP \rightarrow Verb\ PP$ | $VP \rightarrow Verb\ PP$ |
| $VP \rightarrow VP\ PP$ | $VP \rightarrow VP\ PP$ |
| $PP \rightarrow Preposition\ NP$ | $PP \rightarrow Preposition\ NP$ |

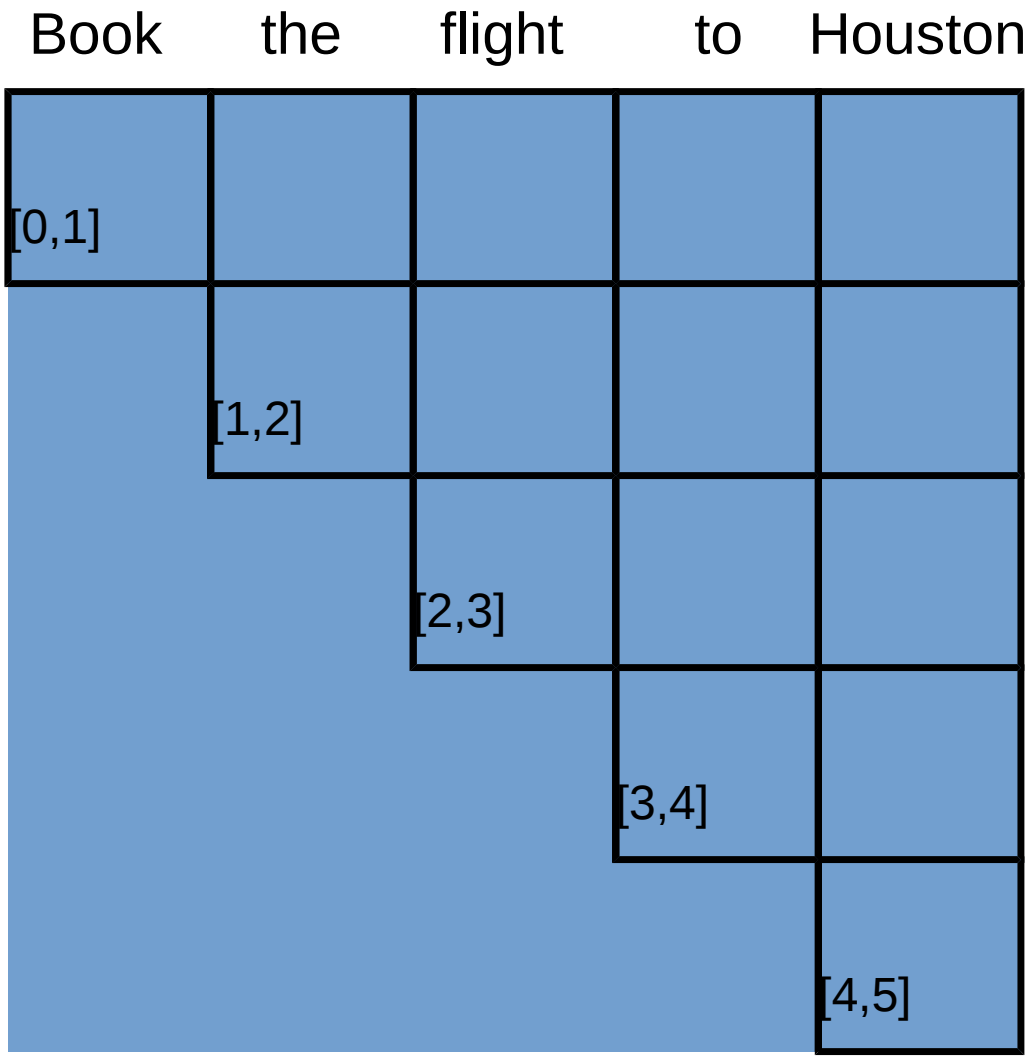
Beispiel aus Jurafsky und Martin, 2nd edition.

# Unsere Beispielgrammatik (in CNF mit allen Regeln, Lexikon links)

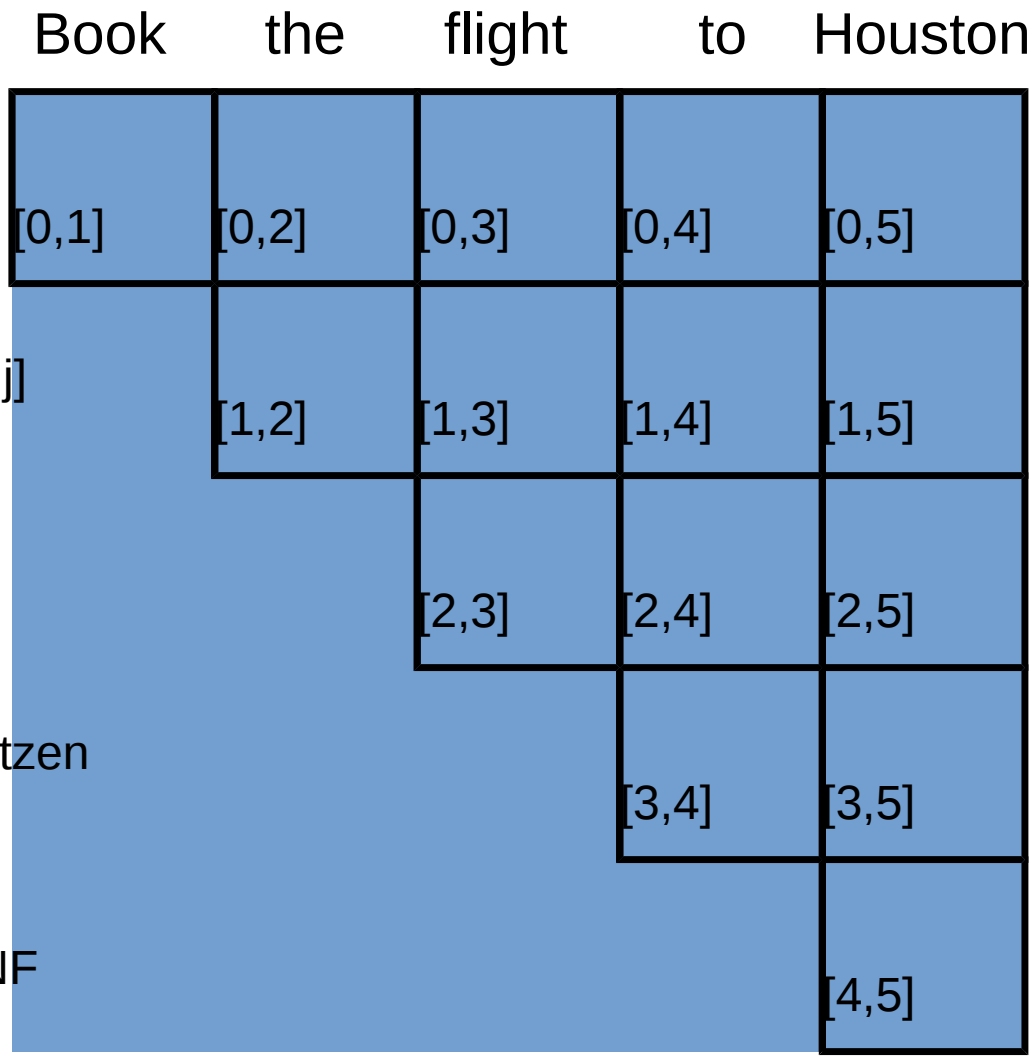
S-> NP VP
S→ X1 VP
X1 → Aux NP
S→ Verb NP
S → X2 PP
S→ Verb PP
S→ VP PP
NP→ Det Nominal
Nominal → Nominal Noun
Nominal → Nominal PP
VP→ Verb NP
VP → X2 PP
X2 → Verb NP
VP→ VP PP
PP→ Preposition NP

S→ book |include |prefer
NP → I |she |me
NP → TWA|Houston
Nominal → book | flight | meal | money
VP → book|include|prefer
Verb → book |include|prefer
Noun → book |flight |meal |money
Proper Noun → TWA |Houston
Pronoun →I | she | me
Aux → does
Det → that | this | a | the
Preposition → from | to | on |near |through

# CKY Algorithmus



# CKY Algorithmus




Initialisierung: Auf Diagonale [j-1,j] stehen mögliche Pos Tags (bottom-up)

Dann spaltenweise links nach Rechts, unten nach oben

Zelle [i,j] kann sich zusammensetzen aus [i,k] und [k,j] für alle k zwischen i+1 und j-1

Binäre Zusammensetzung → CNF

# CKY Algorithmus

Aus Jurafsky und Martin, 2nd edition

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar }
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j] }
```

**Figure 13.10** The CKY algorithm

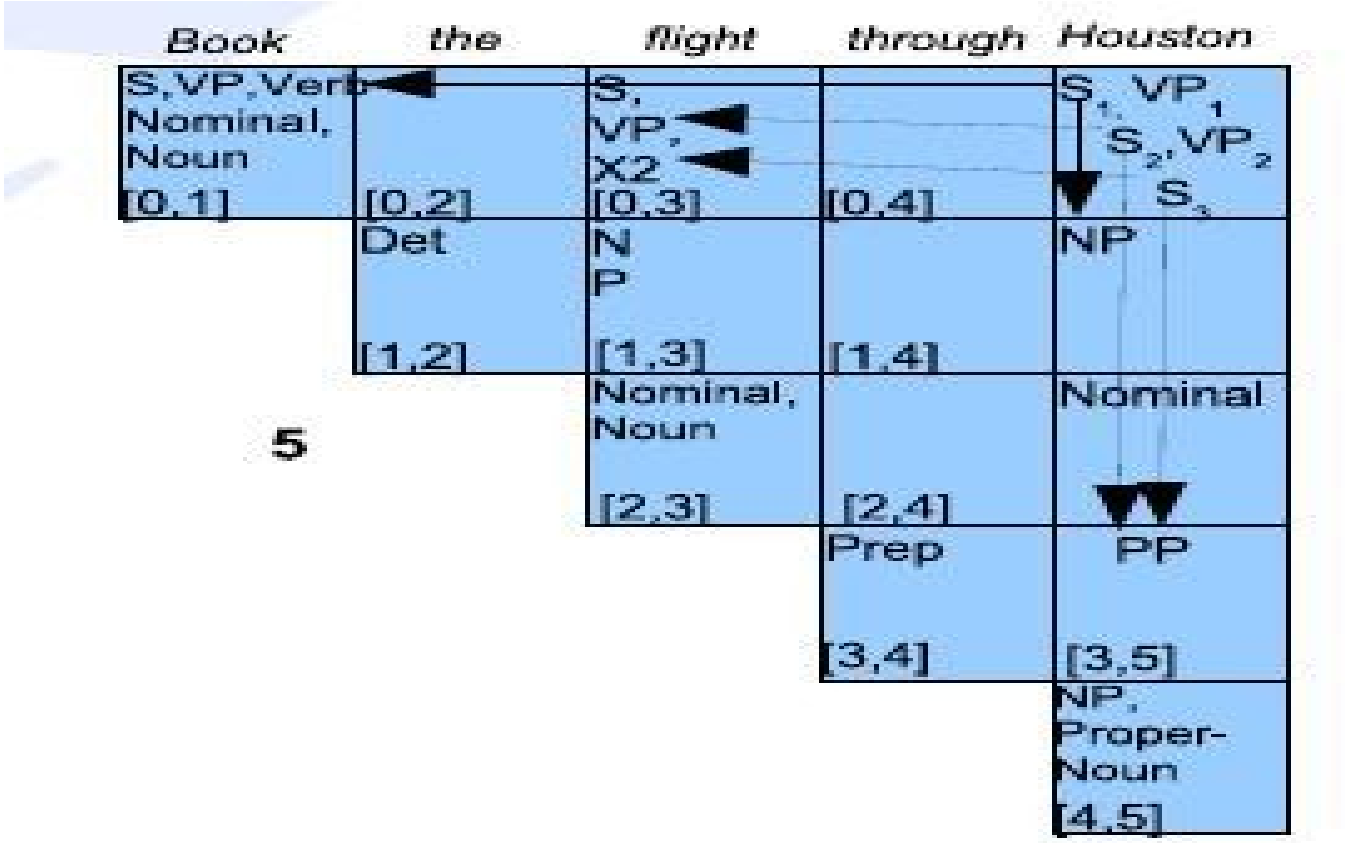
# CKY Algorithmus

Der Algorithmus ist nur ein Recognizer. (Wenn in (0,n) S steht, dann ist der Satz erkannt.)
Um ihn in einen Parser zu verwandeln:

(I) merken wir uns wie ein Nicht-Terminal zustande kam

(ii) erlauben mehr als eine Kopie eines NT in einer Zelle, um Ambiguitäten zu repräsentieren

# CKY Algorithmus Endergebnis




Aus Jurafsky und Martin, 2nd edition

# Well-formed Substrings zu Parses

- Bei jeder Regelanwendung: speichern, woher die neue Konstituente kam
- Beim Auslesen: Anfangen am Start-Item S[0,n]
  jeweils Expansion von Ursprungs-Items
- Vorteil: Alle möglichen Parsebäume werden berechnet
- Vorteil: Chart erlaubt das Packen von Ambiguitäten:

  *Peter saw the man on the hill with the telescope from Canada*
- Vorteil: Laufzeit $O(n^3)$, wenn n Satzlänge. Grund: Teilbäume werden nachgeschaut, nicht neu geparst.
- **Nachteil:** braucht CNF

# Fazit 2. Teil (CYK)

Mit den Inhalten aus diesen Folien können Sie:

- Erklären, was ein **Well-formed substring** ist, und warum diese Idee uns die Anwendung der **dynamischen Programmierung** erlaubt.
- Mit dem CYK-Algorithmus die Well-formed substrings für eine Grammatik (in CNF) und eine Eingabesequenz finden.
- Anhand der Well-formed substring Table, Parses für den jeweiligen Eingabesatz extrahieren sowie Ambiguitäten erkennen.

CKY-Demonstrator http://lxmls.it.pt/2015/cky.html (leider mit anderer Chartdarstellung)

Jurafsky und Martin, 3rd online edition, Kapitel 12 sowie Übungsblatt 8

# Earley-Parsing (Top-Down Chart parsing)

- Eingabe: $_0$ *Der* $_1$ *Hund* $_2$ *sieht* $_3$ *die* $_4$ *Katze* $_5$
- Ein Item der Chart: [3,5] NP -> Det N
  - Überspannt den Eingabeabschnitt: *die Katze*
  - NP erkannt/erstellt durch Anwendung der Regel: NP -> Det N
- Top-Down-Erwartung: .NP[3]
  - NP erwartet wegen der Regel: VP -> V NP
- Aktive Chartparser: Items haben **aktive und inaktive Abschnitte**
  - Bsp: [0 ,2] S -> NP • VP
  - Inaktiv: Bereich der Regel, der bereits analysiert wurde
  - Aktiv: Bereich der Regel, der noch zu analysieren ist
  - Man spricht von **„geteilten Produktionen“** oder **„dotted items/rules“**

# Earley Algorithmus

Der Earley Algorithmus besteht aus 3 Prozeduren

- **Expand** (oder: **predict**):
  - Top-down Komponente: legt aktive Items an
- **Scan** (oder: **shift**):
  - Erzeugt inaktive Items für die Wörter der Eingabekette
- **Complete** (oder: **reduce**):
  - Bottom-up Komponente:
    fasst vorhandene Chart-Einträge zu größeren (inaktiven) Einheiten zusammen

    Vorteile Earley: Benutzt Erwartungen (kann Laufzeit verringern). Braucht CNF nicht.

# Earley Algorithmus: Expand („Predict“)

- Expand („predict“) produziert top-down Hypothesen über die Feinstruktur bereits bestehender Annahmen über die Eingabekette
- Daten: eine kontextfreie Grammatik G = <Φ,∑,S,R>
- Methode

  Wenn die Chart bereits eine Kante der Form

  i j  A -> α • B β

  enthält, dann wird für jede Grammatikregel der Form

  B -> γ

  ein **neues (aktives) Chart-Item** der Form

  j  j  B -> • γ

  angelegt

Bsp:
Item: [0 0] S -> • NP VP
Regel: NP -> Det N
EXPAND:
Neues Item: [ 0 0] NP ->• Det N

# Earley Algorithmus: Scan („Shift“)

- Daten: Eine Eingabekette w = $w_1$ $w_2$ . $w_j$ .. $w_n$ (mit j ≤ n)
- Methode:
  Wenn die Chart ein Item der Form

  i j-1 A -> α • $w_j$ β

  enthält, und $w_j$ ist gleich dem aktuellen Eingabezeichen w an Position j, dann wird ein **neues Item** der Form

  i j A -> α $w_j$ • β angelegt.
- Scan erzeugt inaktive Kanten (es werden Terminalsymbole „gescannt“)

Bsp:
Item: 0 0 Det -> • der
Eingabe: $_0$ der $_1$
SCAN:
Neues Item: 0 1 Det -> der •

# Earley Algorithmus: Complete („Reduce“)

- Complete repräsentiert die Bottom-up Komponente
  - Complete verbindet inaktive Kanten (d.h. vollständig erkannte Teilstrukturen) mit aktiven Kanten
  - D.h. eine Teilkonstituente wurde vollständig erkannt und wird in der „übergeordneten“ (aufrufenden) Regel als bearbeitet (inaktiv) vermerkt
- Enthält die Chart das inaktive Item

  0 1 Det -> der •

  sowie das aktive Item

  0 0 NP -> • Det N

  dann erzeugt COMPLETE das **neue Item**

  0 1 NP -> Det • N

# Eine einfache Variante des Earley-Algorithmus

- Input: eine kontextfreie Grammatik G = <Φ,∑,S,R>
- Input: eine Eingabekette w = $w_1$ $w_2$ ... $w_n$
- Output: Kette erkannt / nicht erkannt

Algorithmus

1. **Initialisierung:**
   a) für alle Regeln der Form S -> α, S Startsymbol von G, erzeuge Chart-Items der Form: **[0 0] S -> • α**
   b) wende auf alle diese Kanten die Prozedur **Expand („predict“)** an, bis keine neuen Items mehr erzeugt werden

# Beispiel: Grammatik

- G = <Φ,∑,S,R>
- Φ = { S, NP, VP, DET, N, V}
- ∑ = { der, Hund, bellt, sieht, die, Katze}
- S = S
- R = {
  S -> NP VP, (1)
  NP -> Det N, (2)
  VP -> V, (3)
  VP -> V NP, (4)
  Det -> der, (5)
  Det -> die, (6)
  N -> Hund, (7)
  N -> Katze, (8)
  V -> bellt, (9)
  V -> sieht } (10)

# Beispiel: Initialisierung

- Input: s = $_0$ der $_1$ Hund $_2$ bellt $_3$
- Initialisierung:
  1. Schritt [0 0] S -> • NP VP
  2. Schritt [0 0] NP -> • Det N
  3. Schritt [0 0] Det -> • der
  4. Schritt [0 0] Det -> • die

# Fortsetzung Algorithmus

2. **Erzeugung weiterer Chart-Kanten:**
   Für alle Positionen j = 0, ..., n und alle Positionen i = 0, ..., j:
   Berechne Items mit Startposition i und Endposition j wie folgt:
   - Wende die Prozedur Scan auf alle Items mit der Startposition i und Endposition j-1 an
   - Wende Expand und Complete solange an, bis diese beiden Prozeduren keine neuen Kanten mehr erzeugen
3. **Auswertung:** Wenn die Chart ein Item der Form

   [0 n] S -> α •

   enthält, ist die Eingabekette *akzeptiert*, ansonsten ist die Kette *nicht akzeptiert.*

# Beispiel: Erzeugung weiterer Items Scan, Expand und Complete

- Input: s = $_0$ der $_1$ Hund $_2$ bellt $_3$
- Expand/Predict [0 0] S -> • NP VP
  [0 0] NP -> • Det N
  [0 0] Det -> • der
  [0 0] Det -> • die
- Scan/Shift [0 1] Det -> der •
- Complete/Reduce [0 1] NP -> Det • N
- Expand/Predict [1 1] N -> • Hund
  [1 1] N -> • Katze
- Scan/Shift [1 2] N -> Hund •
- Complete/Reduce [0 2] NP -> Det N •
  [0 2] S -> NP • VP
- ...

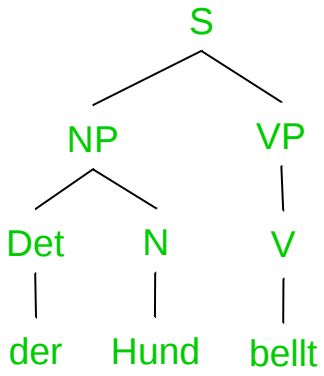
**Beispiel:**
Chart für
*der Hund bellt*

| Nr | Item | | |
|---|---|---|---|
| 1. | | S -> • NP VP | Initialisierung (a) |
| 2. | | NP -> • Det N | Initialisierung (b): Predict/Expand 1. |
| 3. | | Det -> • der | Initialisierung (b): Predict/Expand 2. |
| 4. | | Det -> • die | Initialisierung (b): Predict/Expand 3. |
| 5. | | Det -> der • | |
| 6. | | NP -> Det • N | |
| 7. | | N -> • Hund | |
| 8. | | N -> • Katze | |
| 9. | | N -> Hund • | |
| 10. | | NP -> Det N • | |
| 11. | | S -> NP • VP | |
| 12. | | VP -> • V | |
| 13. | | VP -> • V NP | |

**Beispiel:**
Chart für
*der Hund bellt*

| Nr | Item | | | |
|---|---|---|---|---|
| 1. | 0 | 0 | S -> • NP VP | Initialisierung (a) |
| 2. | 0 | 0 | NP -> • Det N | Initialisierung (b): Predict/Expand 1. |
| 3. | 0 | 0 | Det -> • der | Initialisierung (b): Predict/Expand 2. |
| 4. | 0 | 0 | Det -> • die | Initialisierung (b): Predict/Expand 3. |
| 5. | 0 | 1 | Det -> der • | Shift 3. |
| 6. | 0 | 1 | NP -> Det • N | Reduce 2. mit 5. |
| 7. | 1 | 1 | N -> • Hund | Predict/Expand 6. |
| 8. | 1 | 1 | N -> • Katze | Predict/Expand 6. |
| 9. | 1 | 2 | N -> Hund • | Shift 7. |
| 10. | 0 | 2 | NP -> Det N • | Reduce 6. mit 9. |
| 11. | 0 | 2 | S -> NP • VP | Reduce 1. mit 10. |
| 12. | 2 | 2 | VP -> • V | Predict/Expand 11. |
| 13. | 2 | 2 | VP -> • V NP | Predict/Expand 11. |
| 14. | 2 | 2 | V -> • bellt | Predict/Expand 12. bzw. 13. (!) |
| 15. | 2 | 2 | V -> • sieht | Predict/Expand 12. bzw. 13. (!) |
| 16. | 2 | 3 | V -> bellt • | Shift 14. |
| 17. | 2 | 3 | VP -> V • | Reduce 12. mit 16. |
| 18. | 2 | 3 | VP -> V • NP | Reduce 13. mit 16. |
| 19. | 0 | 3 | S -> NP VP • | Reduce 11. mit 17. |

**Beispiel:**
Chart für
*der Hund bellt*




| Nr | Item | | | |
|---|---|---|---|---|
| 1. | 0 | 0 | S -> • NP VP | Initialisierung (a) |
| 2. | 0 | 0 | NP -> • Det N | Initialisierung (b): Predict/Expand 1. |
| 3. | 0 | 0 | Det -> • der | Initialisierung (b): Predict/Expand 2. |
| 4. | 0 | 0 | Det -> • die | Initialisierung (b): Predict/Expand 3. |
| 5. | 0 | 1 | Det -> der • | Shift 3. |
| 6. | 0 | 1 | NP -> Det • N | Reduce 2. mit 5. |
| 7. | 1 | 1 | N -> • Hund | Predict/Expand 6. |
| 8. | 1 | 1 | N -> • Katze | Predict/Expand 6. |
| 9. | 1 | 2 | N -> Hund • | Shift 7. |
| 10. | 0 | 2 | NP -> Det N • | Reduce 6. mit 9. |
| 11. | 0 | 2 | S -> NP • VP | Reduce 1. mit 10. |
| 12. | 2 | 2 | VP -> • V | Predict/Expand 11. |
| 13. | 2 | 2 | VP -> • V NP | Predict/Expand 11. |
| 14. | 2 | 2 | V -> • bellt | Predict/Expand 12. bzw. 13. |
| 15. | 2 | 2 | V -> • sieht | Predict/Expand 12. bzw. 13. |
| 16. | 2 | 3 | V -> bellt • | Shift 14. |
| 17. | 2 | 3 | VP -> V • | Reduce 12. mit 16. |
| 18. | 2 | 3 | VP -> V • NP | Reduce 13. mit 16. |
| 19. | 0 | 3 | S -> NP VP • | Reduce 11. mit 17. |

# **Chart für** *der Hund bellt*

- In der Chart sind alle möglichen Analysen gespeichert
- Daher können in ambigen Sätzen Teilphrasen, die für eine Lesart konstruiert wurden, für eine andere Lesart wiederverwendet werden
- Die Analysen (Baumstrukturen) können rekonstruiert werden durch Verweise auf verwendete Kanten in Reduce-Aktionen
- Mehrfachberechnungen sind nicht mehr noetig
- Effizienz aber mehr Speicherplatz
- Die Chart enthält auch aktive Kanten, die nicht für Reduktion genutzt wurden
  - Kante 18: diese aktive Kante wird für diesen Satz nicht gebraucht und wird auch nicht weiterverfolgt

# Fazit 3. Teil

Mit den Inhalten aus diesen Folien können Sie:

- Erklären, was die **aktiven Kanten** im Chart Parsing (hier: Earley-Algorithmus) tun.
- Erklären, was die **dotted Rules** im Earley-Algorithmus mit *aktiven Kanten* und mit *n-ären Regeln* zu tun haben.
- Anhand des Earley-Algorithmus eine well-formed-substring table ohne „überflüssige“ passive Kanten berechnen.

**Lektüre**

Daniel Jurafsky and James H. Martin (2008), Kap. 13

Ein weiteres ausgezeichetes und komplexeres Beispiel unter

http://www.cs.jhu.edu/~jason/465/PDFSlides/lect10-earley.pdf (Folie 19 folgende)