Einführung in die Computerlinguistik

Syntaktische Verarbeitung

Benjamin Roth und Daniel Bauer

2. Dezember 2008

WS 2008/2009

Inhalt

Kontextfreie Grammatik und Konstituentenstruktur

- Kontextfreie Grammatiken ordnen Sätzen der Sprache, die sie beschreiben, Ableitungsbäume zu.
- Ableitungsbäume können als Konstituentenstruktur interpretiert werden:
  - Konstituenten beschreiben linguistisch sinnvolle, zusammengehörige Einheiten (S, VP, NP, PP ...)
  - syntaktische Struktur ist Grundlage für semantische Interpretation
- Grammatikregeln erlauben Modellierung struktureller Mehrdeutigkeit / unterschiedlicher Lesarten.

Parsing mit kontextfreien Grammatiken

- **Wortproblem** für kontextfreie Sprachen: Gegeben eine Grammatik, entscheide ob ein Satz in ihrer Sprache ist.
- **Parsing:** Gegeben eine Grammatik, finde eine oder alle gültigen syntaktischen Strukturen für einen Satz.

Die Suche nach gültigen grammatischen Strukturen ist beschränkt durch

1. die Grammatik.
2. den Satz.

Diese Beschränkungen bestimmen die zwei grundsätzlichen Parsing Strategien: Top-Down und Bottom-Up.

Top-Down Parsing

Top-Down

- Konstruiert Ableitungsbäume ausgehend vom Startsymbol.
- Anwendung möglicher Ersetzungsregeln ohne Berücksichtigung des Eingabesatzes.
- Alle (partiellen) Ableitungen sind immer von der Kategorie S.
- Top-Down Parser sind in diesem Sinne "zielgerichtet": Sie suchen nach **Satz**struktur für die Eingabe.

Bottom-Up Parsing

Bottom-Up

- Konstruieren Ableitungsbäume ausgehend von den Terminalsymbolen.
- Anwendung möglicher Ersetzungsregeln mit gegebener rechter Regelseite.
- (Partielle) Ableitungen bestehen immer aus Teilanalysen der Eingabe, sind aber nicht unbedingt Teil einer Satzstruktur.
- Bottom-Up Parser sind in diesem Sinne "datenorientiert".

Ein Top-Down Parser: Rekursiver Abstieg

Vorgehen:

- Erzeuge möglichen Ableitungen, ausgehend vom Startsymbol, und vergleiche sie mit der Eingabe.
- Ableitungen werden nach folgendem Schema erzeugt:
  - Betrachte in der Teilableitung den ersten, nicht expandierten Knoten (ganz links).
  - Wähle eine entsprechende Ersetzungsregel und wende sie an.
  - Wenn Terminalsymbole erzeugt werden, vergleiche sie mit den Eingabewörtern.
  - Stimmen die Wörter nicht überein, verfolge jeweils die nächste Alternativanalysen ("Backtracking").
- Der Rekursive Abstiegs-Parser erzeugt mögliche Ableitungen also von oben nach unten, tiefenorientiert (vergleiche 'depth-first search') und von links nach rechts.

Rekursiver Abstieg - Beispiel

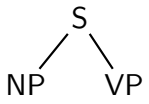
$S \rightarrow NP\ VP$
$NP \rightarrow Det\ N$
$NP \rightarrow PN$
$VP \rightarrow V\ NP$
$Det \rightarrow$ *die*
$N \rightarrow$ *Katze*
$V \rightarrow$ *mag*
$PN \rightarrow$ *Anna*

S

"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

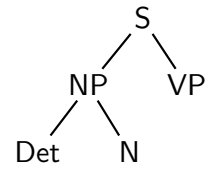
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

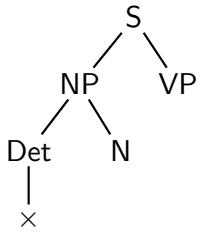
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

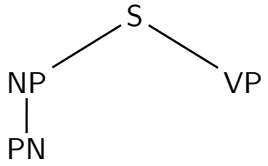
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

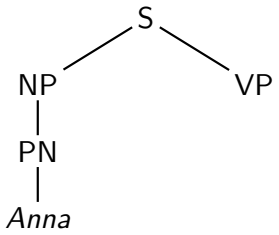
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

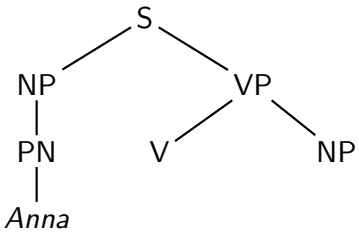
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

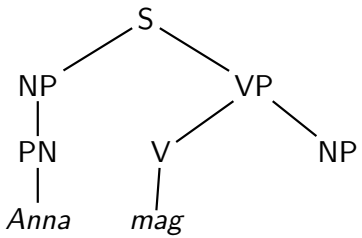
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

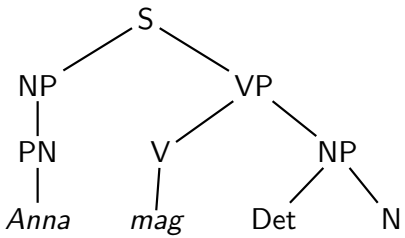
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

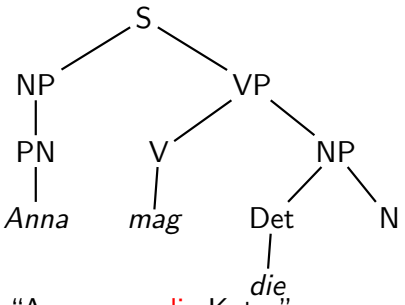
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

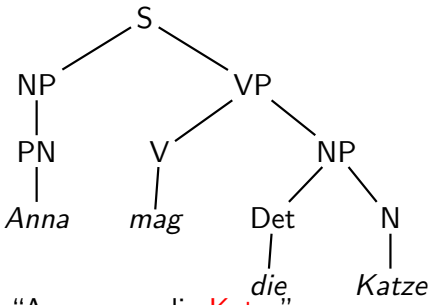
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



"Anna mag die Katze"

Rekursiver Abstieg - Eigenschaften/Probleme

Der Rekursive Abstiegs-Parser hat die folgenden Nachteile:

- Nichtdeterminismus in den Regelanwendungen führt zu teuren Alternativanalysen.
- Keine Berücksichtigung der Eingabe:
  - Erkundet viele Teilstrukturen die niemals zum Eingabesatz führen können.
  - Falsche Teilstrukturen werden immer wieder untersucht bis alle Expansionsmöglichkeiten ausgeschöpft sind.
- Strukturelle Ambiguitäten:
  Sollen alle Lesarten für die Eingabe gefunden werden muss unter Umständen der gesamte von der Grammatik aufgespannte Suchraum erkundet werden.

Rekursiver Abstieg - Eigenschaften/Probleme 2

Linksrekursion:

- Der Rekursive Abstiegs-parser kann nicht mit linksrekursiven Grammatiken umgehen: Der Parser expandiert das linke Nichtterminal immer weiter ohne jemals nur Terminale abuleiten.
- Rekursive Grammatiken:
  - Können regeln der Art $A \rightarrow A\alpha$ enthalten, oder mehrere Regeln die den gleichen Effekt haben z.B.:
    $A \rightarrow B\alpha$
    $B \rightarrow A\beta$
  - Für jede linksrekursive Grammatik gibt es eine schwach äquivalente Grammatik ohne linksrekursion.
    ABER: Diese Grammatiken werden oft zu groß für effizientes Parsen (und sie sind weder lesbar noch linguistisch adäquat)

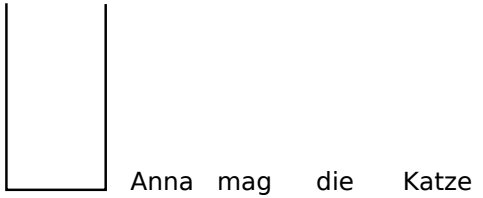
Ein Bottom-Up Parser: Shift-Reduce

Der Shift-Reduce Parser benötigt einen **Parsing Keller** (eng. 'Stack').
Vorgehen:

- Lege das nächste Wort aus dem Eingabesatz in den Keller. ('**shift**')
- Wenn die obersten n Symbole im Keller (α) die rechte Seite einer Ersetzungsregel $A \rightarrow \alpha$ bilden, können die Symbole aus dem Keller genommen und das Nichtterminal A in den Keller gelegt werden. ('**reduce**')
- Wenn im Keller nur noch das Startsymbol S steht und kein Eingabewort mehr verfügbar ist, wurde eine Ableitung gefunden.

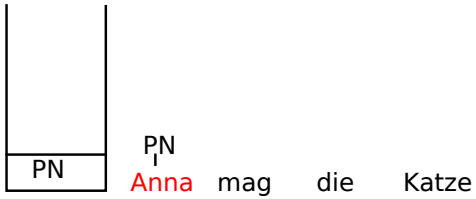
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



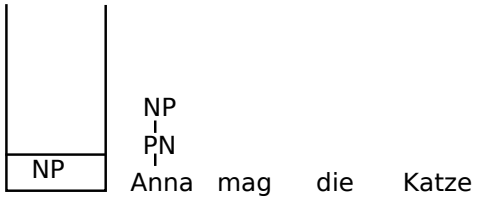
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



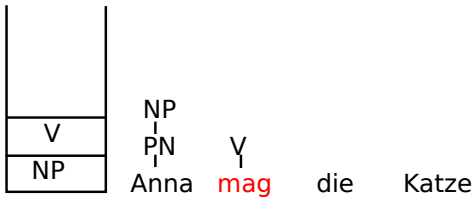
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



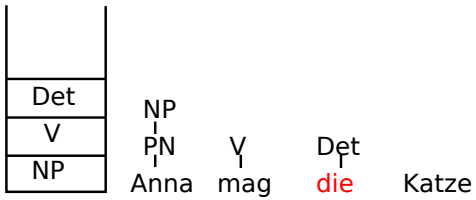
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



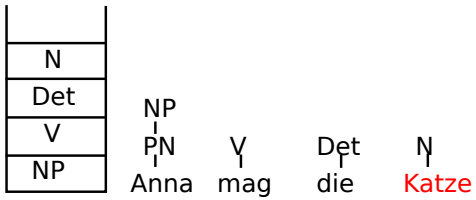
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



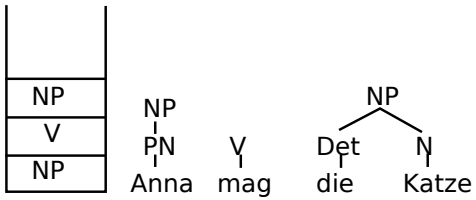
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



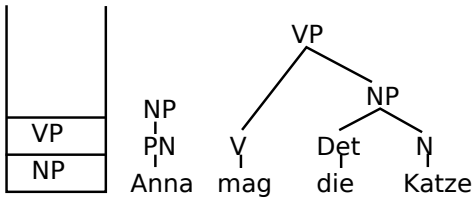
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



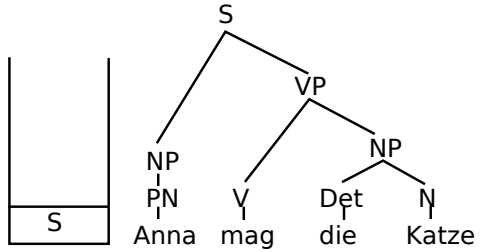
Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Shift-Reduce - Beispiel

S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Shift-Reduce - Eigenschaften

- Shift-reduce Parser werden oft LR Parser genannt, denn sie verarbeiten die Eingabe von **l**inks nach rechts und konstruieren eine umgekehrte **R**echtsableitung.
- Für eine bestimmte Klasse kontextfreier Grammatiken bei denen reduzieren sobald möglich immer die richtige Entscheidung ist, ist der Shift-Reduce Parser sehr effizient.

Shift-Reduce - Probleme

- Der Shift-Reduce Parser ist nicht deterministisch in der Entscheidung ob das nächste Wort in den Keller gelegt wird oder ob der Keller reduziert wird. Auch hier sind wieder teure Alternativanalysen notwendig.
- Der Shift-Reduce Parser erzeugt viele Teilanalysen der Eingabe, die nicht unbedingt zu vollständigen Satzstrukturen führen.
- Strukturelle Ambiguitäten:
  Sollen alle Lesarten für die Eingabe gefunden werden muss auch der Shift-Reduce Parser unter Umständen den ganzen Suchraum erkunden.

Chart Parsing - Idee

Hauptprobleme der betrachteten Parsingalgorithmen:

- Wiederholte Analyse von Teilbäumen (bei der Fehlerkorrektur) sind ineffizient.
- Strukturelle Ambiguität: Wie lassen sich alle Lesarten finden ohne den gesamten Suchraum zu erkunden?

Lösungsidee: **Chart Parsing**

Chart Parsing und dynamische Programmierung

Chart Parsing Algorithmen liegt das Paradigma der **dynamischen Programmierung** zugrunde[1].

- Vermeide rechenintensive wiederholte Analyse von Teilproblemen (abwägung von Rechenzeit und Speicherbedarf).
- Speichere Lösungen für Teilprobleme (hier: mögliche Parses für Teilstrings der Eingabe) in einer **Chart**.
- Die Chart repräsentiert alle möglichen Analysen (**Parse-Wald**).
- Top-Down Algorithmus: Earley Algorithmus
- Bottom-Up Algorithmus: CYK (Cocke, Younger, Kasami) Algorithmus

[1](ein anderes Beispiel für dieses Konzept ist der Algorithmus für Levenshtein-Distanz)

Chart Parsing - Beispiel

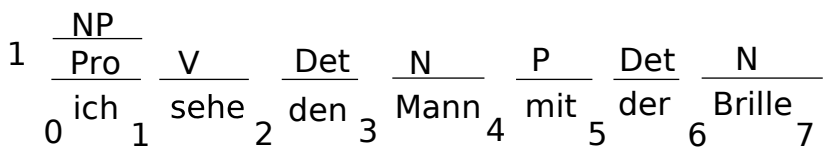


Chart Parsing - Beispiel

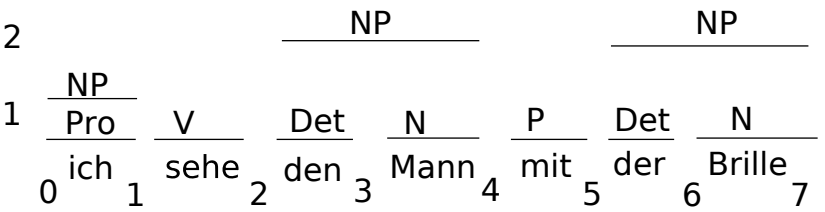


Chart Parsing - Beispiel

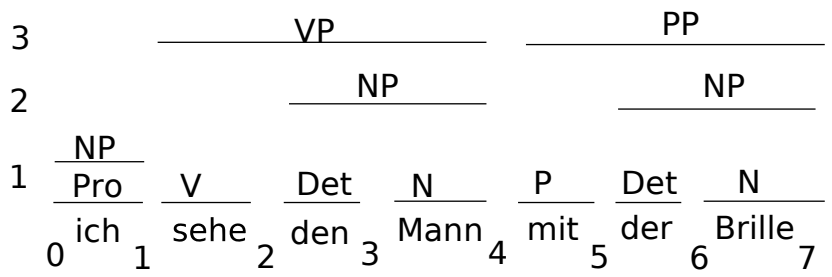


Chart Parsing - Beispiel

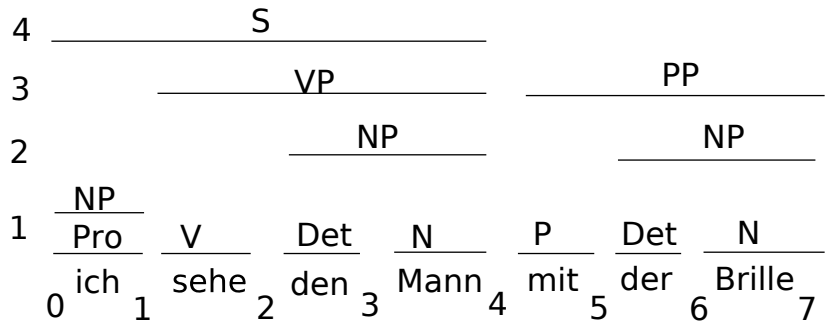


Chart Parsing - Beispiel

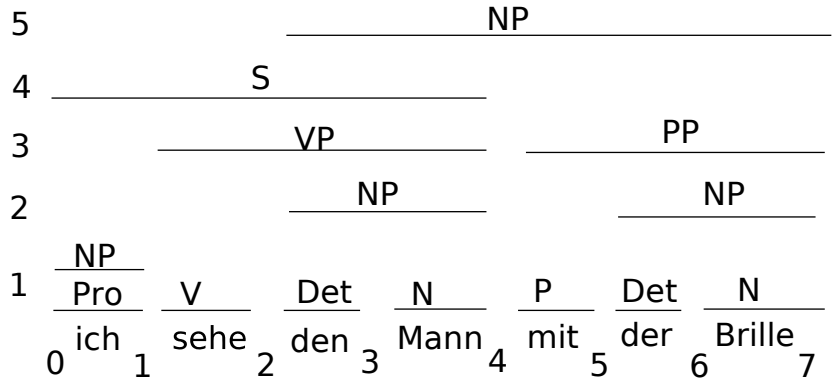


Chart Parsing - Beispiel

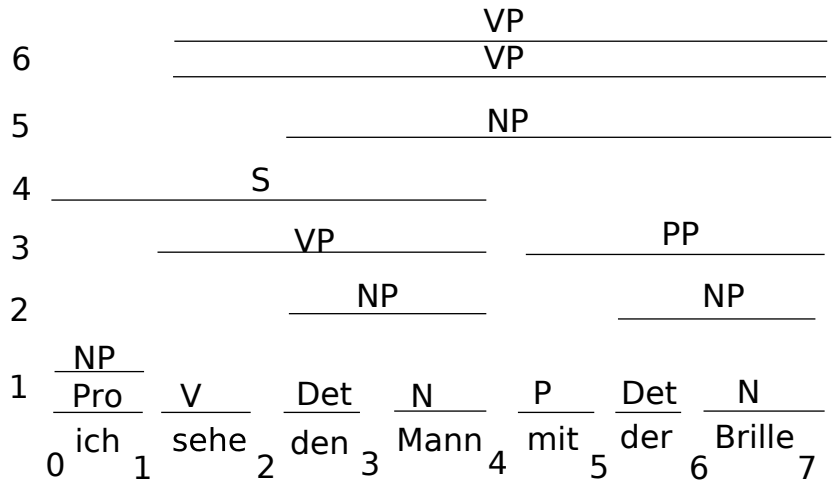


Chart Parsing - Beispiel

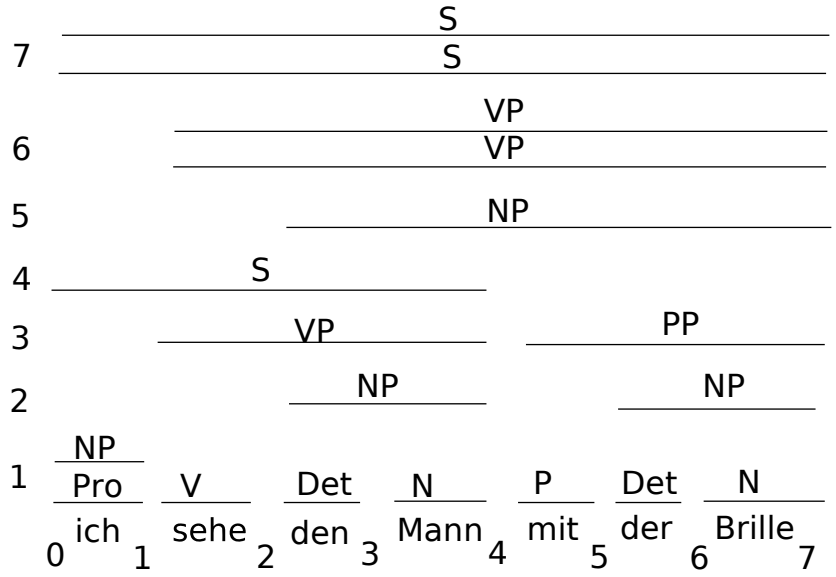


Chart Parsing - Beispiel

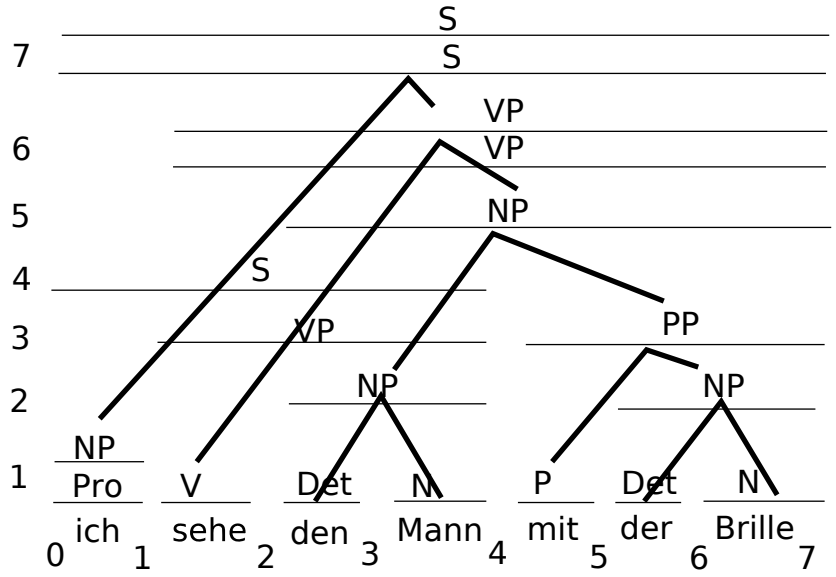
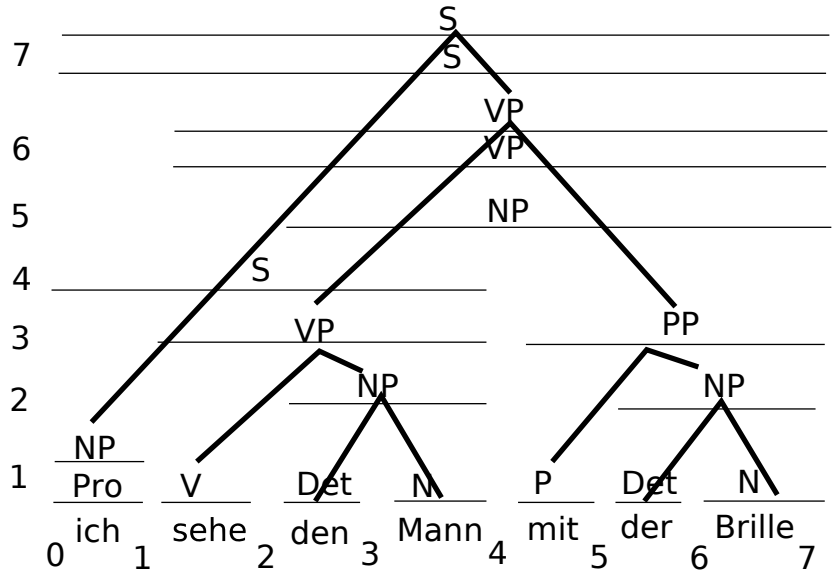


Chart Parsing - Beispiel



Der Earley Algorithmus

[Earley, 1970]

- Top-Down Chart Parsing Algorithmus mit Bottom-Up Kontrolle.
- Verarbeitet die Eingabe in einem einzigen Durchlauf von links nach rechts.
- Grundidee:
  - Erzeuge parallel alle möglichen Ableitungen top-down.
  - Filtere frühstmöglich Ableitungen heraus die zum Eingabesatz inkonsistent sind.

Earley Algorithmus - Die Chart

- Earley Chart enthält $n+1$ Einträge (n: Länge des Eingabesatzes).
- Jeder Eintrag enthält eine Menge von **Zuständen**. Jeder Zustand beschreibt:
  1. einen hypothetischen (unvollständigen) Teilbaum, entsprechend einer Grammatikregel.
  2. Information über den Fortschritt der Vervollständigung des Teilbaums.
  3. der Position dieses Teilbaumes über dem Eingabesatz.
- 1 und 2: Darstellung als punktierte Regel: $S \rightarrow VP \bullet NP$
- 3: Darstellung als Intervall: $[0, 2]$

Earley Algorithmus - Vorgehen

Betrachte Schrittweise alle $n+1$ Einträge (Zustandsmengen) auf der Chart. In jedem Schritt j:

- Für jeden Zustand $\langle R, [i, j] \rangle \in chart[j]$ wende einen der drei folgenden Operatoren an:
  - Wenn R unvollständig und von der Form $A \rightarrow \alpha \bullet a\beta$ ist (a ist ein Terminalsymbol):
    Scan
  - Wenn R unvollständig (von der Form $A \rightarrow \alpha \bullet B\beta$) ist ($B$ ist ein Nichtterminal):
    Predict.
  - Wenn R vollständig (von der Form $B \rightarrow \gamma\bullet$) ist **Complete.**

Earley Algorithmus - Initialisierung

- Anfangs enthält die Chart für jede Regel $S \rightarrow \alpha$ in der Grammatik genau einen Zustand: $< S \rightarrow \bullet\alpha, [0, 0] >$ (initialisierung)

Earley Algorithmus - Predict Operator

Für Zustand $\langle R, [i, j] \rangle \in chart[j]$

- Wenn R unvollständig und von der Form $A \rightarrow \alpha \bullet B\beta$ ist (B ist ein Nichtterminal):
  Predict:
  - Für jede Regel $B \rightarrow \gamma$ aus der Grammatik:
    Füge $\langle B \rightarrow \bullet\gamma, [j, j] \rangle$ zu Eintrag $chart[j]$ hinzu.
- Wenn rechts vom Punkt ein Nichtterminal B steht, schreibe alle Regeln mit B auf der linken Seite auf die Chart.
- Ein Top-Down Operator: "vermutet" mögliche Teilbäume und fügt sie der Chart hinzu.

Earley Algorithmus - Scan Operator

Für Zustand $\langle R, [i,j] \rangle \in chart[j]$

- Wenn R unvollständig und von der Form $A \rightarrow \alpha \bullet a\beta$) ist ($a$ ist ein Terminalsymbol) und eingabe$[j] = a$:
  Scan:
  - Für jeden Eintrag $\langle A \rightarrow \alpha \bullet a\beta, [h, j-1] \rangle$:
  - Füge $\langle A \rightarrow \alpha a \bullet \beta, [j, j+1] \rangle$ zu Eintrag $chart[j+1]$ hinzu.

- Wenn das nächste Symbol a rechts vom Punkt ein **Terminalsymbol** ist, versuche ein Eingabewort zu lesen.
- Ein Bottom-Up Operator: Filtert die möglichen Teilbäume auf der Chart entsprechend der Eingabe.

Earley Algorithmus - Complete Operator

Für Zustand $\langle R, [i,j] \rangle \in chart[j]$

- Wenn R vollständig (von der Form $B \rightarrow \gamma\bullet$) ist: **Complete**
  - Für jeden Eintrag $\langle A \rightarrow \alpha \bullet B\beta, [h,i]$ in $chart[i]$:
    Füge $\langle A \rightarrow \alpha B\bullet, [h,j] \rangle$ zu Eintrag $chart[j]$ hinzu.

- Wenn der Punkt am Ende eines Zustandes mit B auf der linken Seite steht (Teilbaum vollständig erkannt), Verschiebe den Punkt in allen unvollständigen Zuständen in denen das nächste Nichtterminal B ist.
- Vervollständigt mögliche Teilbäume mit Teilbäumen die mit der Eingabe konsistent sind.

Earley Algorithmus - Beispiel

sieheTafel

Earley Algorithmus - Eigenschaften

- Worst-case Laufzeit $O(n^3)$ (n: länge des Eingabesatzes).
- Der hier beschriebene Algorithmus ist eigentlich kein Parser:
  - Erkennt eigentlich nur ob der Eingabesatz in der Sprache der Grammatik ist.
  - Nach einer einfachen Erweiterung im Predict Operator lassen sich aber alle Lesarten in der Chart repräsentieren und ablesen.