Einführung in die Computerlinguistik

Parsing

WS 2010/2011

Manfred Pinkal

Endliches Gedächtnis

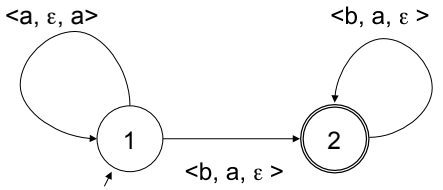
- Der endliche Automat kann nur beschränkte Information in seinen Zuständen kodieren.
- Das Gedächtnis eines endlichen Automaten mit n Zuständen reicht deshalb höchstens n-1 Zeichen zurück (→Pumping Lemma).
- Kontextfreie Grammatiken erzeugen aber beliebig tief geschachtelte Strukturen, in denen beliebig weit voneinander entfernte Elemente voneinander abhängen können.
- Wir können das Gedächtnisproblem durch einen zusätzlichen Speicher mit im Prinzip unbegrenzter Kapazität lösen.

Beispiel: L = $a^n b^n$

- Der Automat liest zunächst nacheinander a's ein und „merkt sie sich", indem er sie in den Speicher schreibt. Anschließend liest er nacheinander die b's und nimmt für jedes gelesene b ein a vom Speicher.
- Wenn Eingabewort und gespeicherte Folge von a's gleichzeitig aufgebraucht sind, wird das Eingabewort akzeptiert.
- Für die Verarbeitung kontextfreier Sprachen reicht es, den Speicher als Stack oder Stapel zu benutzen, auf den nach dem LastIn-FirstOut-Prinzip zugegriffen wird. Der Stack wird auch „Keller" oder „Push-Down-Store" genannt. Wir sprechen im Deutschen deshalb vom „Kellerautomaten", im Englischen vom „Push-down Automaton" (PDA).

Kellerautomaten: Ein Beispiel

- Ein Kellerautomat,der $a^n b^n$ akzeptiert:



- <u, v, w> als Kanteninschrift steht für: Lies Eingabe u, lösche v vom Speicher, schreibe w auf den Stack.
- Im Zustand 1 werden a's gelesen und in den Stack geschrieben.
- Beim ersten b wechselt der Automat in den Zustand 2 und löscht für jedes gelesene b ein a vom Stack.
- Wenn die Eingabe abgearbeitet, der Stack leer und ein Endzustand erreicht ist, wird das Eingabewort akzeptiert.

Keller-Automaten und kontextfreie Grammatiken

- Kontextfreie Grammatiken (CFGs) erlauben die einfache und elegante Definition von kontextfreien Sprachen – kommen aber zunächst ohne ein sinnvolles Analyseverfahren, das die Zugehörigkeit eines Wortes w zu L(G) entscheidet.
- Keller-Automaten stellen ein einfaches Verfahren zur Entscheidung von $w \in L(G)$ für bestimmte kontextfreie Sprachen zur Verfügung – aber keine intuitive Methode, um komplexe Sprachen direkt zu modellieren.
- Frage: Sind die Formalismen der CFG und des PDA gleich stark?
- Frage: Können wir eine CFG in einen äquivalenten PDA überführen?
- Die Antwort auf beide Fragen ist ja. Der Beweis erfolgt wie bei der NEA-DEA-Überführung konstruktiv, durch Spezifikation von Verfahren.

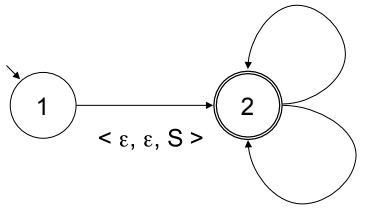
Keller-Automaten und kontextfreie Grammatiken

- Idee: Wir simulieren den Ableitungsprozess der CFG im Stack des PDA, und gleichen die auf dem Stack erzeugten Terminalsymbole mit der Eingabe ab.
- Die Abfolge der Regelanwendung gibt uns gleichzeitig Information über die syntaktische Struktur.
- Systeme, die für eine gegebene Grammatik und einen Eingabesatz die syntaktische Struktur bestimmen, nennen wir Parser.

Ein Schema für CFG-Parsing

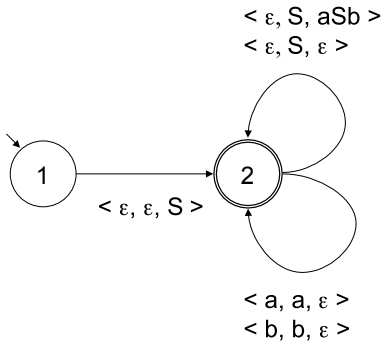
- Initialisiere den Stack des Automaten mit dem Startsymbol.
- Wenn sich das oberste Stack-Element ein nicht-terminales Symbol A ist, wähle eine Grammatikregel $A \rightarrow u$ und ersetze das Stack-Symbol A durch u.
- Wenn das oberste Stack-Element ein Terminalsymbol a und mit dem aktuelle Eingabesymbol identisch ist, lösche a vom Stack und rücke in der Eingabe vor.

Ein Schema für CFG-Parsing



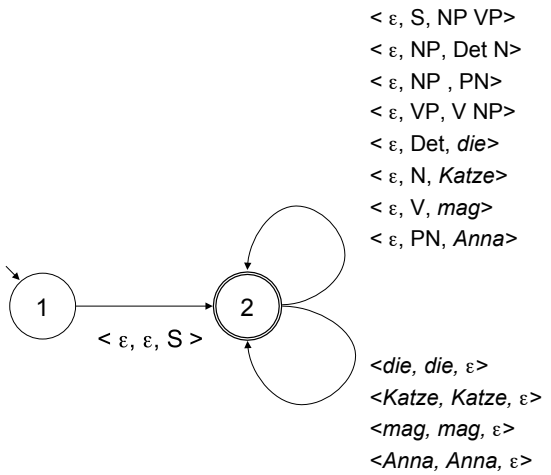
Beispiel: $a^n b^n$

S → aSb
S → ε



Beispiel: Kleine Grammatik fürs Deutsche

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



Kontextfreier Top-Down-Parser

- Der vorgestellte Parser erzeugt ausgehend vom Startsymbol Ableitungen und damit implizit einen Ableitungsbaum. Terminalsymbole werden von links nach rechts mit der Eingabe abgeglichen.
- Wir sprechen bei diesem Vorgehen von „Top-Down“-Parsing: Der Ableitungsbaum bzw. Parsebaum wird von oben nach unten, im „rekursiven Abstieg“ durch die Struktur, aufgebaut.
- Wir können das Verfahren äquivalent direkt durch Operationen auf Ableitungs-/Parsebäumen darstellen.

Kontextfreier Top-Down-Parser

- Generiert Baumstrukturen (Ableitungs-/Parsebäume), indem er die aktiven Knoten bearbeitet.
- Aktiv sind alle Blattknoten eines Baums, die noch nicht verbraucht sind.
- Initialisiere mit dem S-Knoten.
- Für den aktuellen Baum: Nimm den am weitesten links stehenden aktiven Knoten.
  - Nicht-Terminalsymbol A: Wähle eine Ersetzungsregel mit linker Seite A und wende sie auf den aktiven Knoten an.
  - Terminalsymbol a: Gleiche mit dem aktuellen Eingabesymbol ab, markiere den Knoten im Erfolgsfall als verbraucht, und rücke in der Eingabe vor.
- Wenn die Eingabe verbraucht ist und keine aktiven Knoten mehr vorhanden sind, akzeptiere die Eingabe und gib den Parsebaum als Analyse aus.

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

S

"Anna mag die Katze"

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

(S)

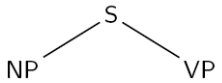
"Anna mag die Katze"

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

(S)

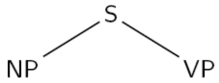
"Anna mag die Katze"

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



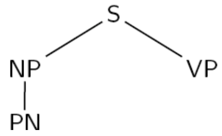
"Anna mag die Katze"

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



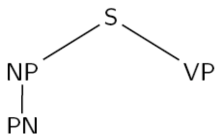
"Anna mag die Katze"

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



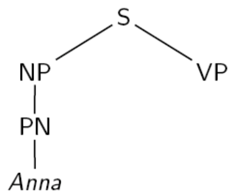
"Anna mag die Katze"

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



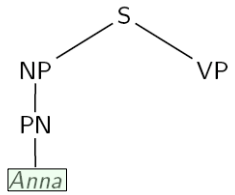
"Anna mag die Katze"

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



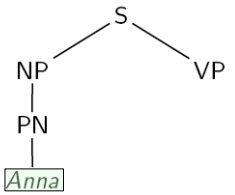
"Anna mag die Katze"

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



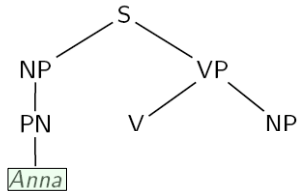
"Anna mag die Katze"

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



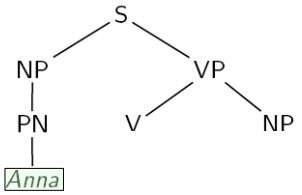
"Anna mag die Katze"

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



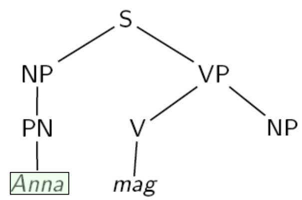
"Anna mag die Katze"

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



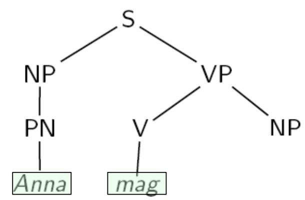
"Anna mag die Katze"

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



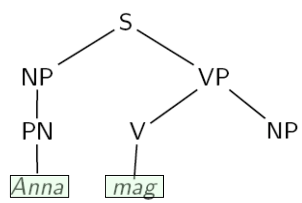
"Anna mag die Katze"

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



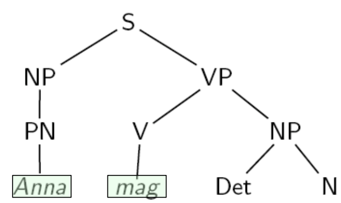
"Anna mag die Katze"

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



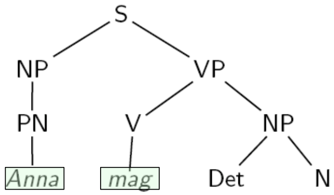
"Anna mag die Katze"

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



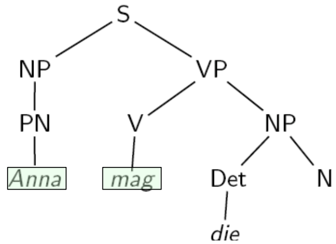
"Anna mag die Katze"

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



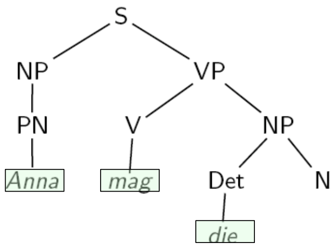
"*Anna mag die Katze*"

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



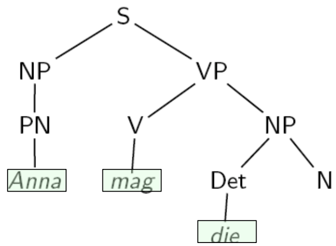
"*Anna mag die Katze*"

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



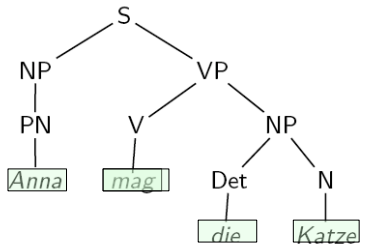
"*Anna mag die Katze*"

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



"*Anna mag die Katze*"

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



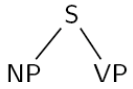
"Anna mag die Katze"

Nicht-Determinismus in der Regelanwendung

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

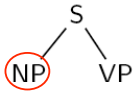
"Anna mag die Katze"

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



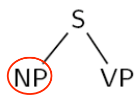
"Anna mag die Katze"

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



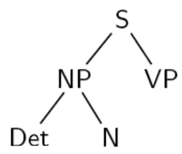
"Anna mag die Katze"

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



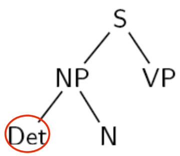
"Anna mag die Katze"

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



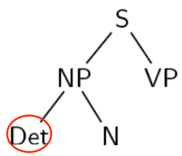
"Anna mag die Katze"

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



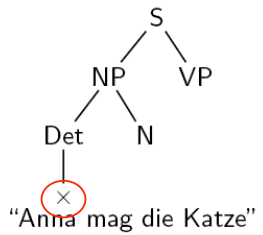
"Anna mag die Katze"

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



"Anna mag die Katze"

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



Top-Down-Parser, Eigenschaften

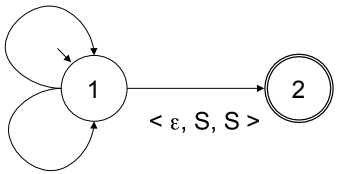
- Der Top-Down-Parser ist im Allgemeinen nicht-deterministisch: Für dasselbe Nichtterminal gibt es mehrere, eventuell sehr viele Ersetzungsregeln.
- Eine technische Lösung: Arbeiten mit einer Agenda, Last-In – First-Out, Tiefensuche mit Backtracking.
- Problem: Es werden viele Teilstrukturen erzeugt, die nie erfolgreich sein können, weil die Eingabekette keine passenden Wörter enthält.
  - Beispiel: Grammatik versucht, Subjekts-NP abzuleiten, der Satz fängt mit einer PP an.
- Im Falle „links-rekursiver" Grammatiken geht der Parser in eine Endlosschleife.
  - Beispiel: VP → VP PP
  - Links-rekursive Regeln kann man vermeiden, aber dann sind bestimmte Strukturen nicht mehr natürlich darstellbar.

Alternative: Bottom-Up Parser

- Der Bottom-Up-Parser (auch „Shift-Reduce-Parser") arbeitet sich „von unten nach oben" durch die Eingabe. Er liest zunächst Symbole der Eingabekette in den Stack ein und ersetzt geeignete Symbolfolgen im Stack durch entsprechende Kategoriesymbole.
- Der Parser verwendet zwei Operationen (falls beide möglich sind, können sie in beliebiger Reihenfolge angewendet werden):
  - Shift: Lies das nächste Symbol der Eingabekette in den Stack ein.
  - Reduce: Wenn es eine Grammatikregel $A \rightarrow u$ gibt und der Stack mit u^R beginnt (also die Symbole von u in umgekehrter Reihenfolge oben auf dem Stack liegen), ersetze u^R durch A.
- Wenn die Eingabe abgearbeitet ist und der Stack nur noch das Startsymbol S enthält, akzeptiere die Eingabe.

Schema für den kontextfreien Bottom-Up-Parser

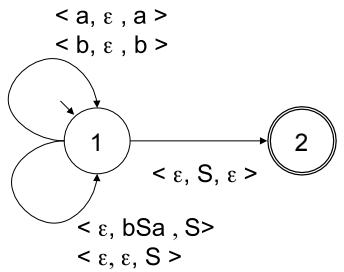
< a, ε, a > *(für jedes Terminalsymbol a)*



< ε, u^R, A > *(für jede Regel A → u)*

Beispiel: $a^n b^n$

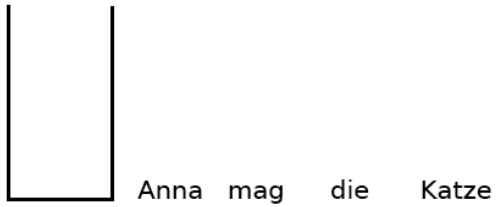
S → aSb
S → ε



Shift-Reduce - Beispiel

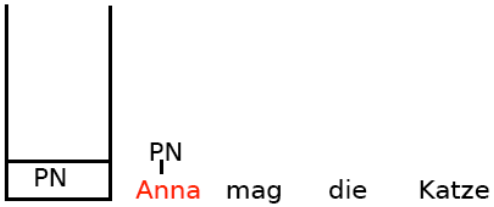
Im folgenden Beispiel wird der Parserzustand nach der Shift-Operation jeweils übersprungen!

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



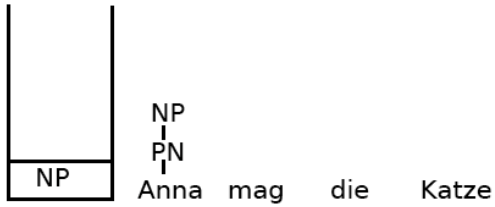
Shift-Reduce - Beispiel

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



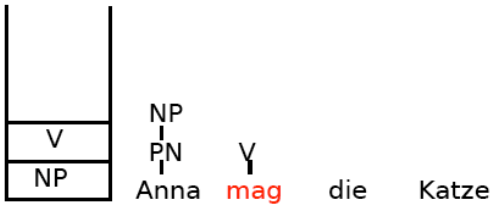
Shift-Reduce - Beispiel

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



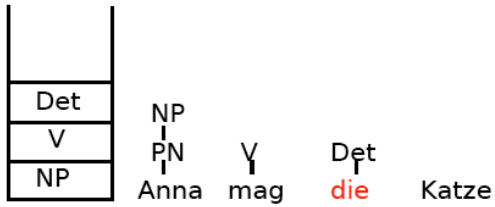
Shift-Reduce - Beispiel

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



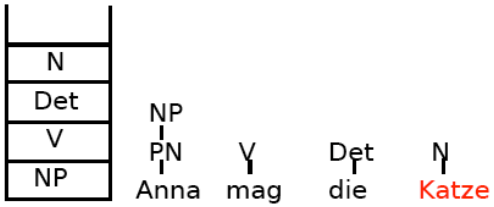
Shift-Reduce - Beispiel

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



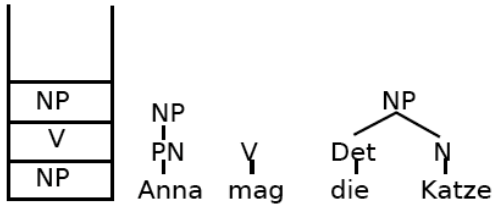
Shift-Reduce - Beispiel

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



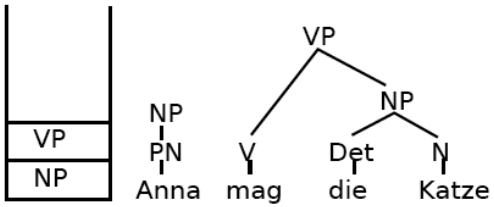
Shift-Reduce - Beispiel

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



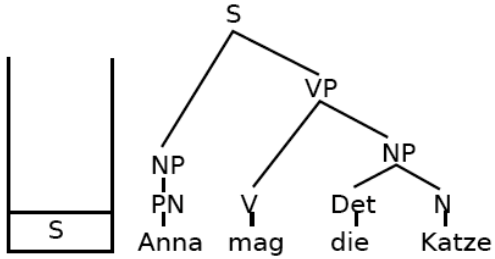
Shift-Reduce - Beispiel

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*



Bottom-Up-Parser: Eigenschaften

- Auch der Bottom-Up-Parser ist nicht-deterministisch.
- Er vermeidet Analysen, die durch lexikalisches Material nicht abgedeckt sind, und ist insofern effizienter als der Top-Down-Parser.
- Er erzeugt jedoch Teilstrukturen auch dann, wenn absehbar ist, dass sie nie zu vollständigen Satzstrukturen führen.

CFG-Parsing und Determinismus

- Top-Down- und Bottom-Up-Parser für $a^n b^n$ sind beide nicht-deterministisch.
- Der Nicht-Determinismus ist aber nicht essenziell: $a^n b^n$ kann durch einen deterministischen Kellerautomaten erkannt werden (siehe ganz oben).
- Frage: Gibt es für jede kontextfreie Sprache L einen deterministischen Automaten/ Parser, der L erkennt?
- Können Sätze natürlicher Sprachen in linearer Zeit analysiert werden?
- Die Antwort ist Nein.

CFG-Parsing und Determinismus

- Es gibt kontextfreie Sprachen, die deterministisch geparst werden können, und Sprachen, bei denen das nicht möglich ist.
- Wir unterscheiden „deterministisch kontextfreie Sprachen“ und „nicht-deterministisch kontextfreie Sprachen“.
- Beispiel:
  - L1 = { wcw^R | w∈{a,b}*}
  - L2 = { ww^R | w∈{a,b}*}
  - L1 ist deterministisch, L2 nicht.

CFG-Parsing und Determinismus

- Tendenziell sind alle interessanten formalen Sprachen („Klammersprache“, Arithmetik, Programmiersprachen) deterministisch kontextfrei.
- Natürliche Sprachen sind nicht-deterministisch.
- Sie müssen schon deswegen nicht-deterministisch sein, damit sie syntaktische Mehrdeutigkeit modellieren können.
- Es gibt allerdings deutlich effizientere Parsing-Techniken als die einfachen Top-Down- und Bottom-Up-Parser.

CFG-Parsing und Determinismus

Peter sieht den Mann mit dem Teleskop

CFG-Parsing und Determinismus

Peter sieht den Mann mit dem Teleskop durch ein Fernglas.

Peter sieht den an computerlinguistischen Fragestellungen interessierten Studenten im ersten Semester mit dem Teleskop durch ein Fernglas.

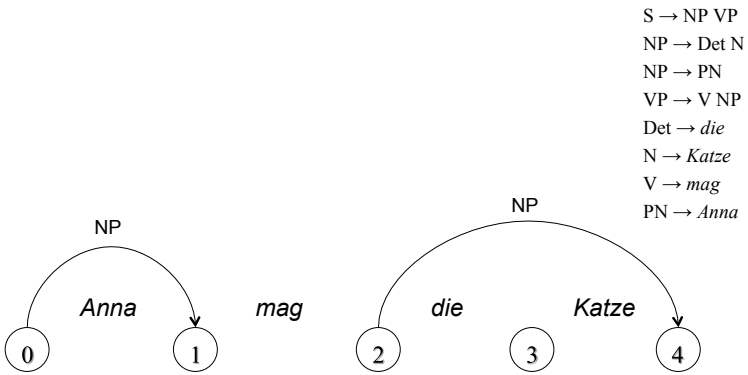
Chart-Parsing

- Das Problem: Teilstrukturen werden unnötigerweise immer wieder analysiert.
- Die Lösung: Der Parser speichert Zwischenresultate in einer Datenstruktur (der „Chart“), auf die er im weiteren Verlauf der Verarbeitung zurückgreifen kann.
- Wir sprechen von „Chart-Parsing“.

Chart, graphische Darstellung

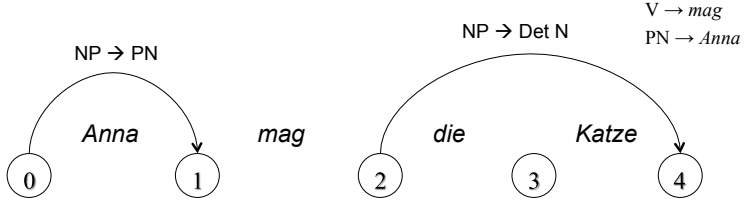
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

Chart, graphische Darstellung



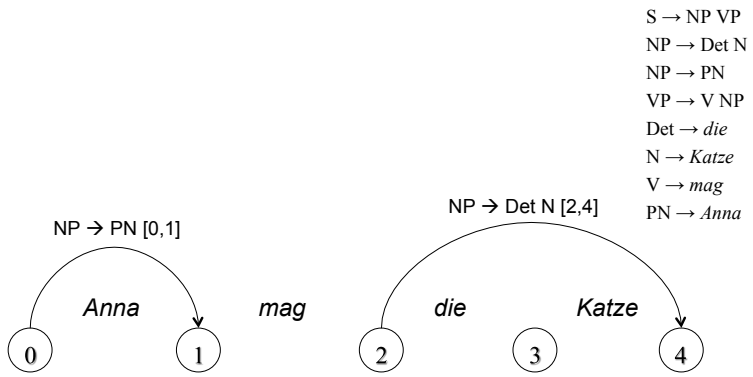
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

Chart, graphische Darstellung



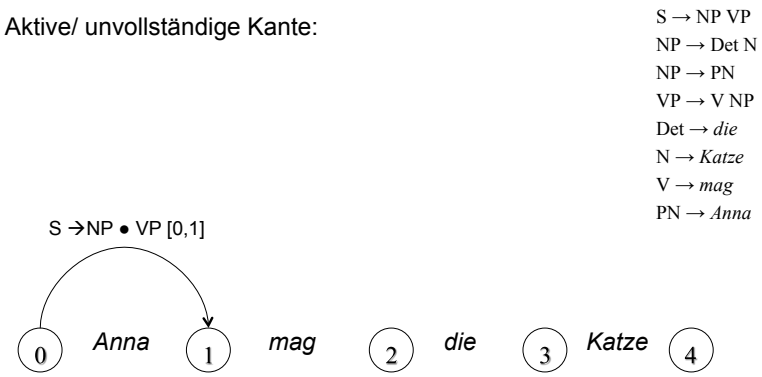
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

Chart, graphische Darstellung



Chart, graphische Darstellung

Aktive/ unvollständige Kante:



Beispiel-Chart

Aktive, initiale Kante

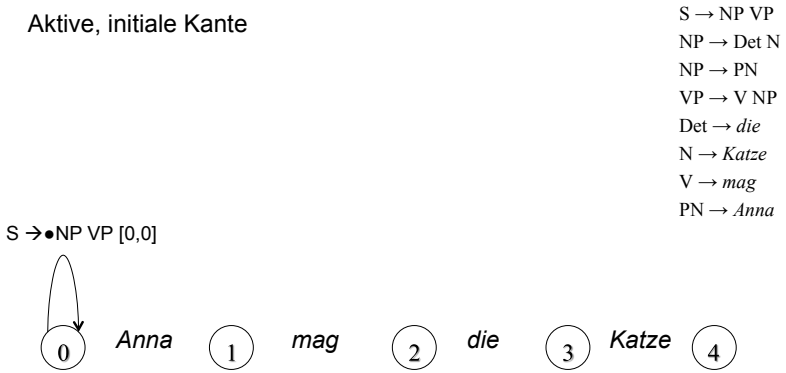


Chart: Formale Darstellung

- Für einen Eingabesatz der Länge n wird eine Chart mit n+1 Spalten eingerichtet.
- Einträge bestehen aus zwei Informationen, die Zusammen ein (potentiell unvollständiges) Parseresultat kodieren:
  - „Punktierte Regel“: Eine Regel A→u, bei der die Symbole auf der rechten Seite durch einen Punkt getrennt sind
  - Paar [i, j], das einen Teilstring der Eingabekette bezeichnet.
- Beispiel: <S → NP • VP, [0, 2]>
  - Wenn die Regel S → NP VP auf den Teil der Eingabe angewandt wird, der an Position 0 beginnt, kann an Position 2 der NP-Teil der Regel vollständig abgearbeitet sein.

Chart: Tabllen-Darstellung

Position 0	Position 1	Position 2	Position 3	Position 4
NP → PN, [0,1]		NP → Det N, [2,4]		

Chart: Tabllen-Darstellung

Position 0	Position 1	Position 2	Position 3	Position 4
S → NP•VP, [0,1]				

Chart-Parsing

- Das Problem: Teilstrukturen werden unnötigerweise immer wieder analysiert.
- Die Lösung: Der Parser speichert Zwischenresultate in einer Datenstruktur (der „Chart“), auf die er im weiteren Verlauf der Verarbeitung zurückgreifen kann.
- Wir sprechen von „Chart-Parsing“.

Earley-Algorithmus

- Ein Standard-Algorithmus für das Chart-Parsing ist der Earley-Algorithmus, der von links nach rechts durch den Eingabesatz geht und in einer Top-Down-Strategie alle Analyse-Alternativen gleichzeitig verfolgt:

Earley-Algorithmus – Vorgehen

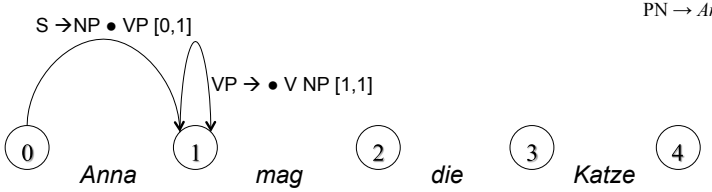
- Es seien u, v ∈ V*: (möglicherweise leere) Ketten von (Terminal- oder Nicht-Terminal-)Symbolen; P punktierte Erzeugungsregeln.
- Initialisiere die erste Spalte der Chart mit <S → • u, [0, 0]> für jede Regel S → u der Grammatik.
- Gehe schrittweise von links nach rechts durch die Chart. In jedem Schritt j:
  - Wende für jeden Eintrag <P, [i, j]> eine der folgenden Operationen an:
  - Wenn P unvollständig und von der Form A → u • av ist (a Terminalsymbol) : Scan
  - Wenn P unvollständig und von der Form A → u • Bv ist (B Nicht-Terminalsymbol) : Predict
  - Wenn P vollständig (von der Form A → u •) ist: Complete.

Predictor

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

Predictor

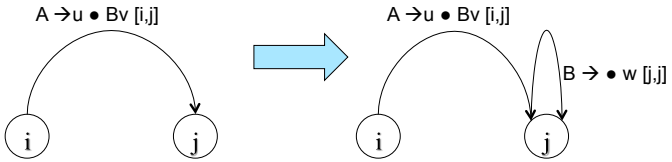
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Earley-Algorithmus – Predictor

- Für Position j, Eintrag <P, [i, j]>:
  - Wenn P unvollständig und von der Form A → u • Bv ist (B Nicht-Terminalsymbol) :
  - Füge für jede Regel B → u der Grammatik <B → • u, [j, j]> zur Position j hinzu.
- Die punktierte Regel A → u • Bv im Eintrag drückt aus, dass als nächstes eine Konstituente der Kategorie B kommen könnte. Der Predictor schreibt alle Regeln in die Chart, die auf der linken Seite ein B haben könnten.

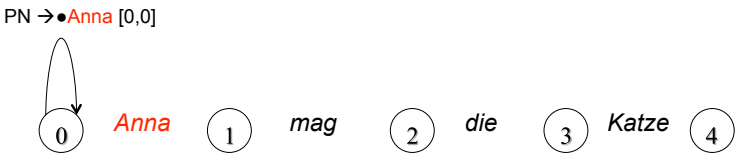
Predictor, Schema



Scanner: Beispiel

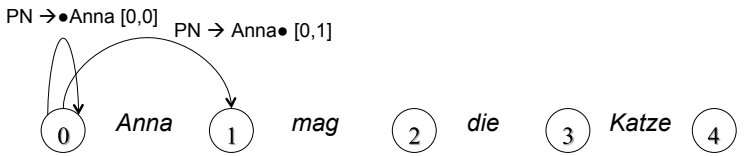
Scanner: Beispiel

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



Scanner: Beispiel

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

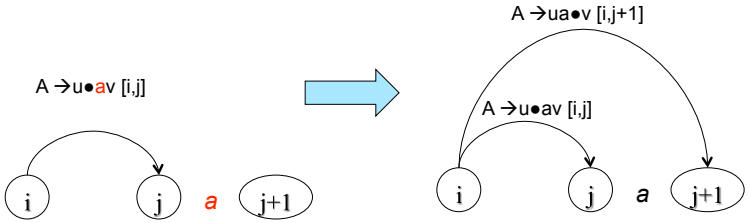


Earley-Algorithmus – Scanner

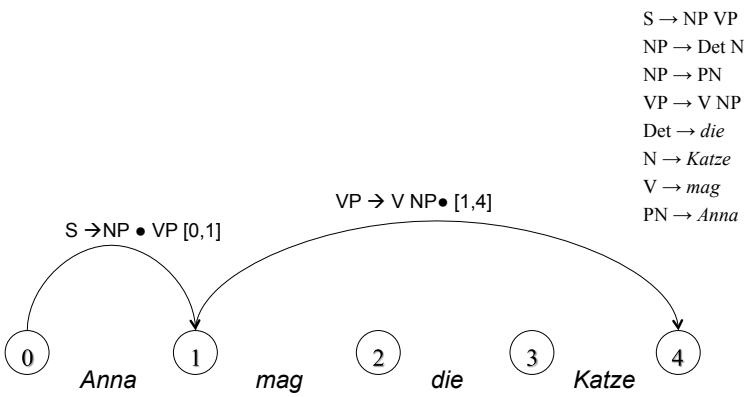
- Für Position j, Eintrag <P, [i, j]>:
  - Wenn P unvollständig und von der Form A → u ● av ist (a Terminalsymbol) , und die Position [j, j+1] der Eingabe mit a gefüllt ist:
  - Füge <A → ua●v , [i, j+1]> zur Spalte j+1 der Chart hinzu.

- Die punktierte Regel A → u ● av drückt aus, dass als nächstes das Eingabesymbol a erwartet wird. Der Scanner liest das Eingabewort und gleicht es mit der Regel ab. Im Erfolgsfall wird der Punkt über a hinweg geschoben und ein neuer Eintrag in der nächsten Spalte der Chart gemacht.

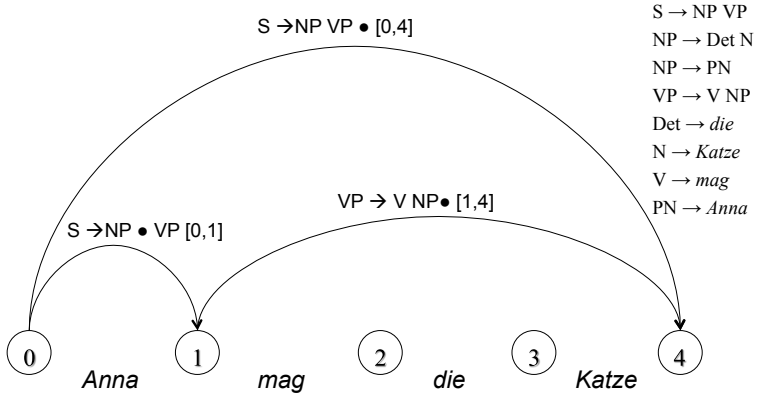
Scanner, Schema



Completer, Beispiel



Complete, Beispiel

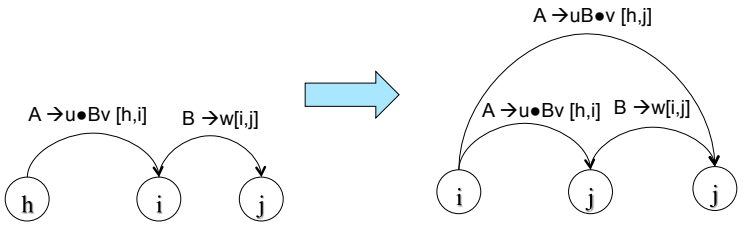


Earley-Algorithmus – Completer

- Für Position j, Eintrag <P, [i, j]>:
  - Wenn P vollständig und von der Form B → u:
  - Füge für jeden bestehenden Eintrag <A → u●Bv , [h, i]> einen neuen Eintrag <A → uB●v , [h, j]> hinzu.

- B → u ● besagt, dass die rechte Seite der Regel vollständig abgearbeitet wurde. Das Resultat kann verwendet werden, um eine bereits bestehende partielle Analyse, die als nächstes einen Ausdruck der Kategorie B erwartet, zu komplettieren.

Completer, Schema



Earley-Algorithmus: Abschluss

- Die Eingabe ist grammatisch, wenn der Algorithmus einen Eintrag <S → u● , [0, n]> erzeugt.

- Auf den folgenden Folien wird die Chart aufgebaut für die obige Beispielgrammatik und den Beispielsatz

  Anna mag die Katze

Beispiel-Chart: Initialisierung

Position 0	Position 1	Position 2	Position 3	Position 4
S → ● NP VP, [0,0]				

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

Beispiel-Chart: Operationen auf Pos. 0

Position 0	Position 1	Position 2	Position 3	Position 4
S → ●NP VP, [0,0]	Scanner:			
Predictor: NP → ●Det N, [0,0]	PN → *Anna*●, [0,1]			
NP → ●PN, [0,0]				
Det → ●*die*, [0,0]				
PN → ●*Anna*, [0,0]				

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

Beispiel-Chart: Operationen auf Pos. 1

Position 0	Position 1	Position 2	Position 3	Position 4
S → ●NP VP, [0,0]	Scanner:	Scanner:		
Predictor: NP → ●Det N, [0,0]	PN → *Anna*●, [0,1]	V → *mag*●, [1,2]		
NP → ●PN, [0,0]	Completer:			
Det → ●*die*, [0,0]	NP → PN●, [0,1]			
PN → ●*Anna*, [0,0]	S → NP●VP, [0,1]			
	Predictor:			
	VP → ●V NP, [1,1]			
	V → ●*mag*, [1,1]			

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

Beispiel-Chart: Operationen auf Pos. 2

Position 0	Position 1	Position 2	Position 3	Position 4
S → ●NP VP, [0,0]	Scanner:	Scanner:	Scanner:	
Predictor: NP → ●Det N, [0,0]	PN → *Anna*●, [0,1]	V → *mag*●, [1,2]	Det → *die*●, [2,3]	
NP → ●PN, [0,0]	Completer:	Completer:		
Det → ●*die*, [0,0]	NP → PN●, [0,1]	VP → V●NP, [1,2]		
PN → ●*Anna*, [0,0]	S → NP●VP, [0,1]	Predictor:		
	Predictor:	NP → ●Det N, [2,2]		
	VP → ●V NP, [1,1]	NP → ●PN, [2,2]		
	V → ●*mag*, [1,1]	Det → ●*die*, [2,2]		
		PN → ●*Anna*, [2,2]		

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

Beispiel-Chart: Operationen auf Pos. 4

Position 0	Position 1	Position 2	Position 3	Position 4
S → ●NP VP, [0,0]	Scanner:	Scanner:	Scanner:	Scanner:
Predictor: NP → ●Det N, [0,0]	PN → *Anna*●, [0,1]	V → *mag*●, [1,2]	Det → *die*●, [2,3]	N → Katze●, [3,4]
NP → ●PN, [0,0]	Completer:	Completer:	Completer:	Completer:
Det → ●*die*, [0,0]	NP → PN●, [0,1]	VP → V●NP, [1,2]	NP → Det●N, [2,3]	NP → Det N●, [2,4]
PN → ●*Anna*, [0,0]	S → NP●VP, [0,1]	Predictor:	Predictor:	VP → V NP●, [1,4]
	Predictor:	NP → ●Det N, [2,2]	N → ●Katze, [3,3]	S → NP VP●, [0,4]
	VP → ●V NP, [1,1]	NP → ●PN, [2,2]		
	V → ●*mag*, [1,1]	Det → ●*die*, [2,2]		
		PN → ●*Anna*, [2,2]		

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

Beispiel-Chart: Operationen auf Pos. 4

Position 0	Position 1	Position 2	Position 3	Position 4
S → ●NP VP, [0,0]	Scanner:	Scanner:	Scanner:	Scanner:
Predictor: NP → ●Det N, [0,0]	PN → *Anna*●, [0,1]	V → *mag*●, [1,2]	Det → *die*●, [2,3]	N → Katze●, [3,4]
NP → ●PN, [0,0]	Completer:	Completer:	Completer:	Completer:
Det → ●*die*, [0,0]	NP → PN●, [0,1]	VP → V●NP, [1,2]	NP → Det●N, [2,3]	NP → Det N●, [2,4]
PN → ●*Anna*, [0,0]	S → NP●VP, [0,1]	Predictor:	Predictor:	VP → V NP●, [1,4]
	Predictor:	NP → ●Det N, [2,2]	N → ●Katze, [3,3]	S → NP VP●, [0,4]
	VP → ●V NP, [1,1]	NP → ●PN, [2,2]		
	V → ●*mag*, [1,1]	Det → ●*die*, [2,2]		
		PN → ●*Anna*, [2,2]		

Earley-Algorithmus

- Der Eintrag S → NP VP● , [0, 4] zeigt, dass die analysierte Wortkette ein in unserer Beispielgrammatik grammatischer Satz ist.
- Wenn wir die vollständigen Regeleinträge der Chart so miteinander verlinken, dass ein Eintrag auf die Einträge verweist, auf denen er aufbaut, können wir außerdem die syntaktische Struktur / den Parsebaum ablesen.

Beispiel-Chart

- Auf vollständige Einträge beschränkt, Abhängigkeiten sind markiert