Einführung in die Computerlinguistik

Parsing

WS 2014/2015

Vera Demberg

Zwischenevaluation

- Bitte gehen Sie auf die Kurshomepage und füllen Sie die Zwischenevaluation aus.

  → HEUTE!

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 bei endlichen Automaten durch einen zusätzlichen Speicher mit im Prinzip unbegrenzter Kapazität lösen.

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

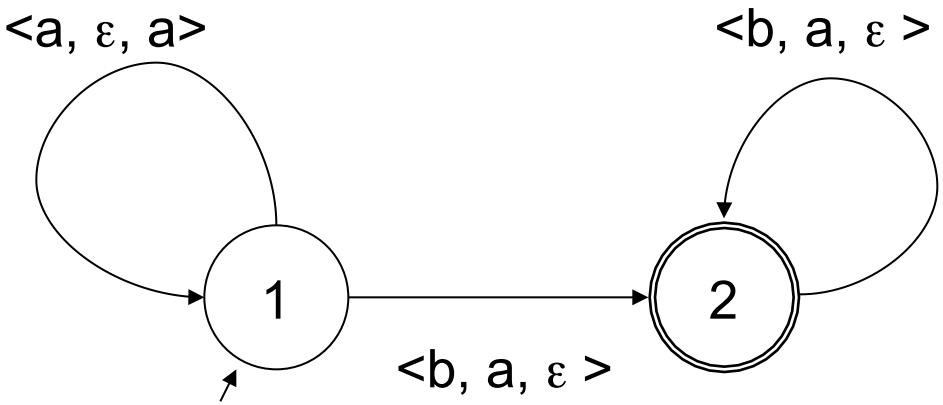
- Der Automat liest zunächst nacheinander a‘s ein und „merkt sie sich“, indem er sie in den Speicher schreibt. Anschließend liest er nacheinander die b‘s und nimmt für jedes gelesene b ein a vom Speicher.

- 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 Stack, 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 ∈ 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

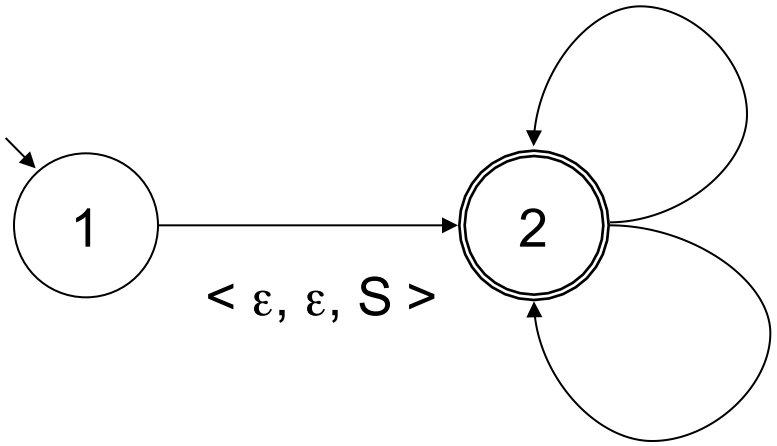
- **Idee**: Wir simulieren den Ableitungsprozess der CFG im Stack des PDA, und gleichen die auf dem Stack erzeugten Terminalsymbole mit der Eingabe ab.

- 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

< ε, A, u > *(für jede Regel A→u)*



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

Ein Schema für CFG-Parsing

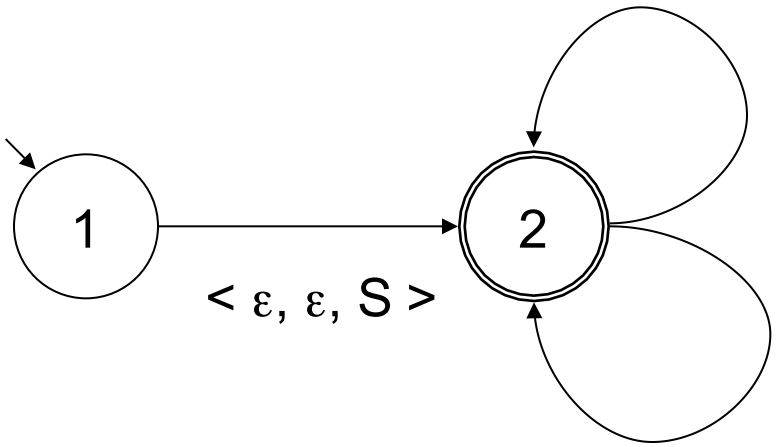
- Initialisiere den Stack des Automaten mit dem Startsymbol.
- Wenn sich das oberste Stack-Element ein nicht-terminales Symbol A ist, wähle eine Grammatikregel $A \rightarrow u$ und ersetze das Stack-Symbol A durch u.
- Wenn das oberste Stack-Element ein Terminalsymbol a und mit dem aktuelle Eingabesymbol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

Beispiel: $a^n b^n$

S → aSb

S → ε

< ε, A, u > *(für jede Regel A→u)*

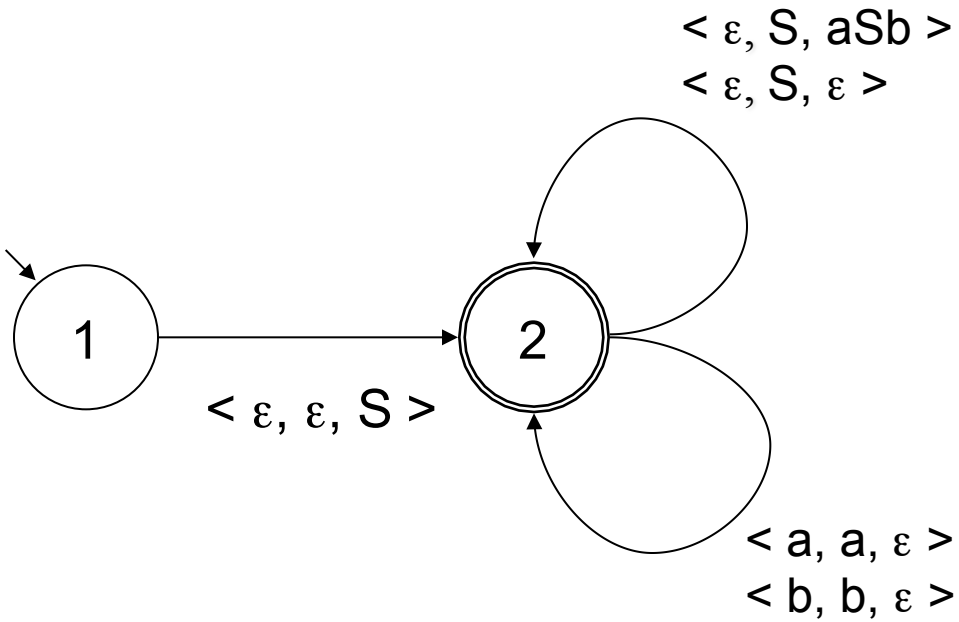


< a, a, ε > *(für jedes Terminalsymbol a)*

Beispiel: $a^n b^n$

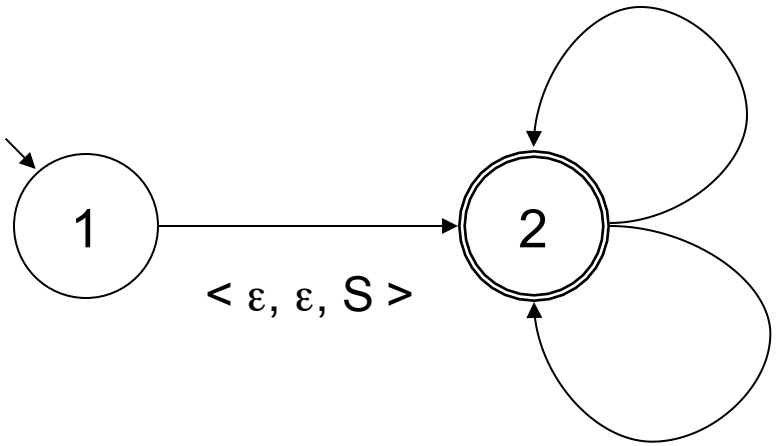
S → aSb

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*



< ε, S, NP VP>
< ε, NP, Det N>
< ε, NP , PN>
< ε, VP, V NP>
< ε, Det, *die*>
< ε, N, *Katze*>
< ε, V, *mag*>
< ε, PN, *Anna*>

<*die, die,* ε>
<*Katze, Katze,* ε>
<*mag, mag,* ε>
<*Anna, Anna,* ε>

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.

Kontextfreier Top-Down Parser

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

Anna mag die Katze

Kontextfreier Top-Down Parser

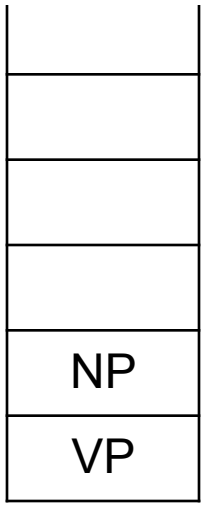
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

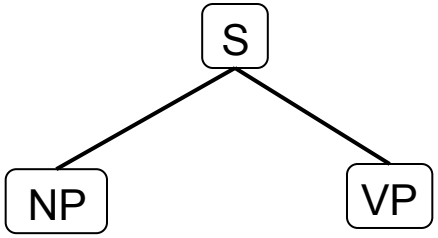
S

Anna mag die Katze

Kontextfreier Top-Down Parser

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

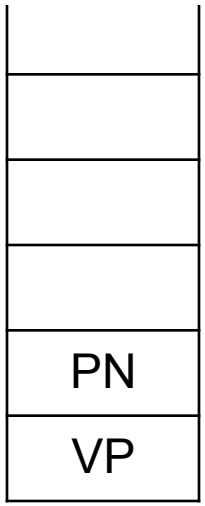


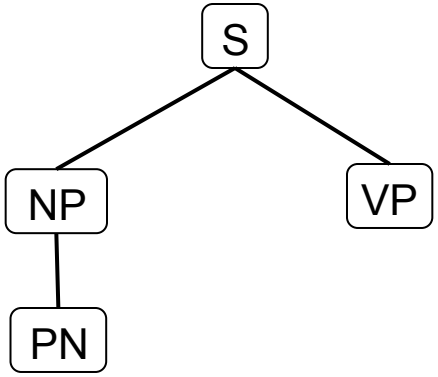


Anna mag die Katze

Kontextfreier Top-Down Parser

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

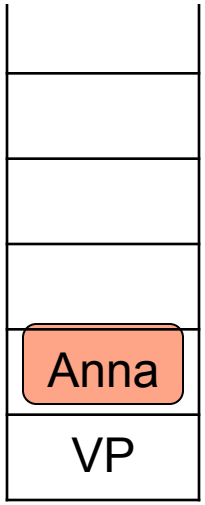


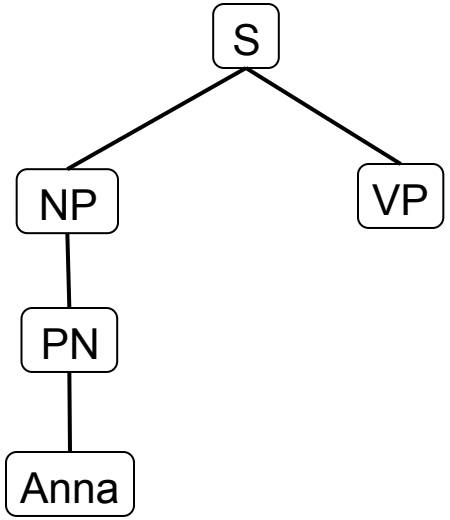


Anna mag die Katze

Kontextfreier Top-Down Parser

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

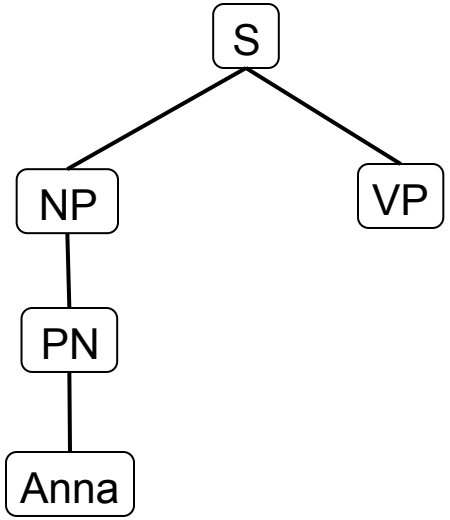




Anna mag die Katze

Kontextfreier Top-Down Parser

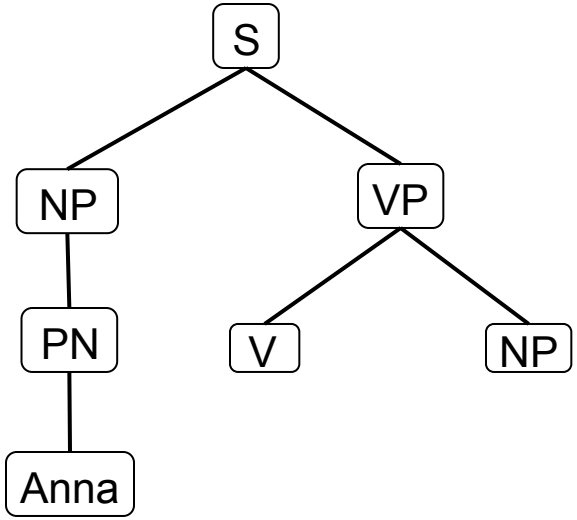
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

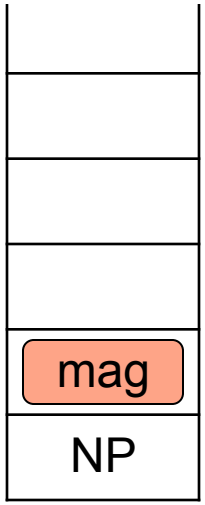
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

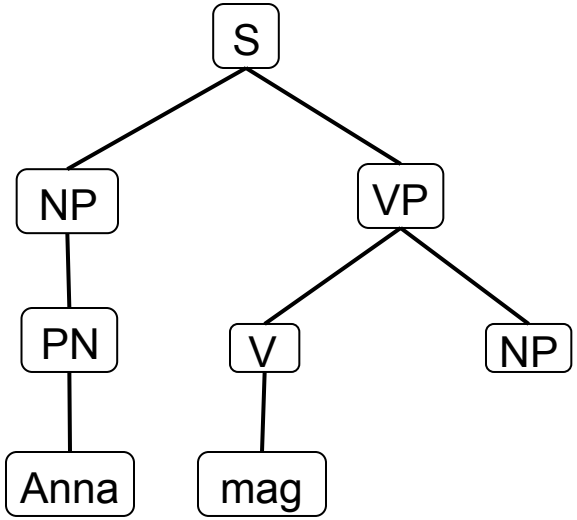


Anna mag die Katze

Kontextfreier Top-Down Parser

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

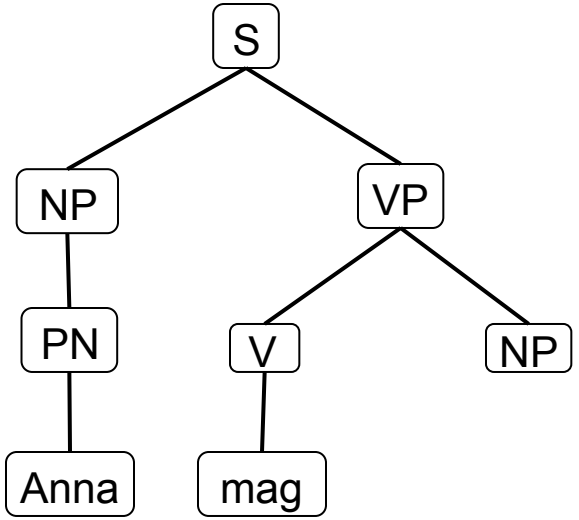




Anna mag die Katze

Kontextfreier Top-Down Parser

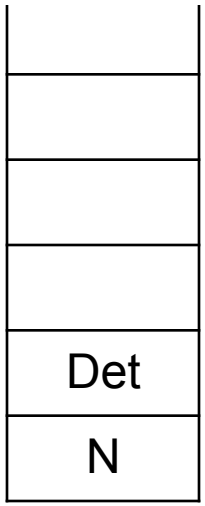
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

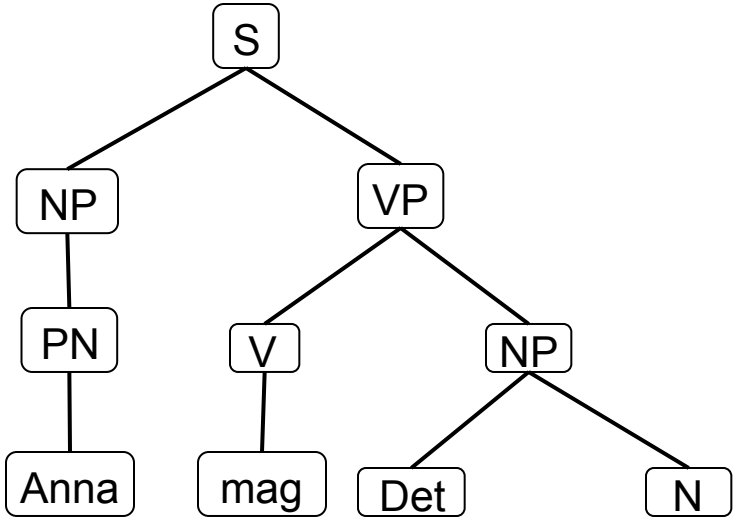


Anna mag die Katze

Kontextfreier Top-Down Parser

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

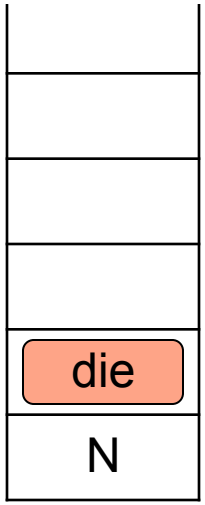


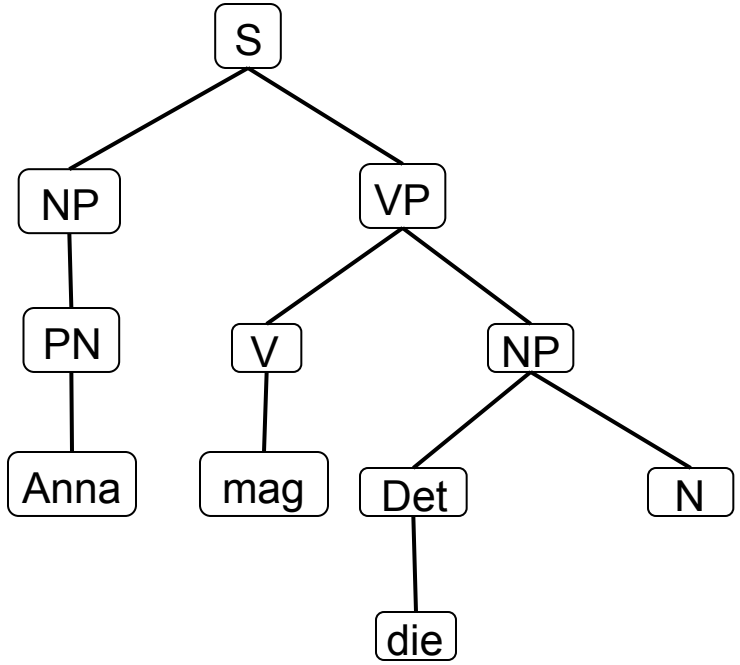


Anna mag die Katze

Kontextfreier Top-Down Parser

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

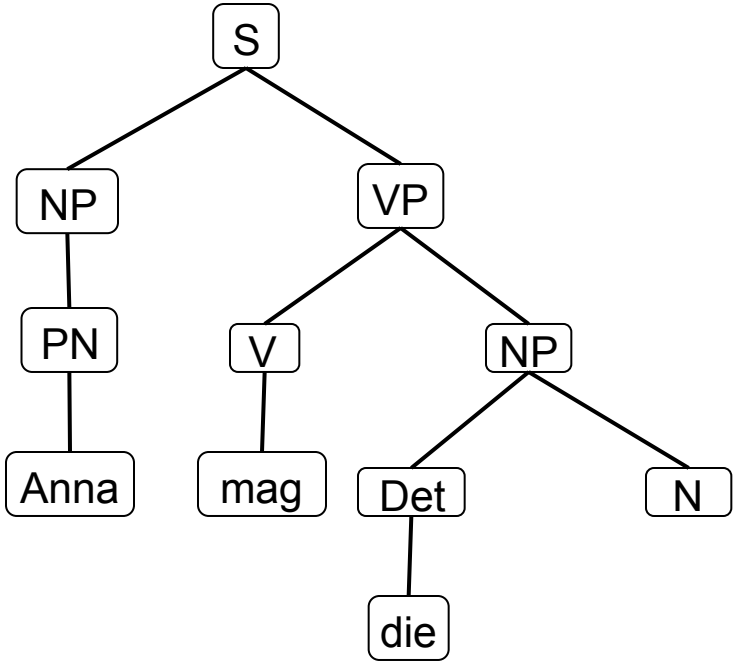




Anna mag die Katze

Kontextfreier Top-Down Parser

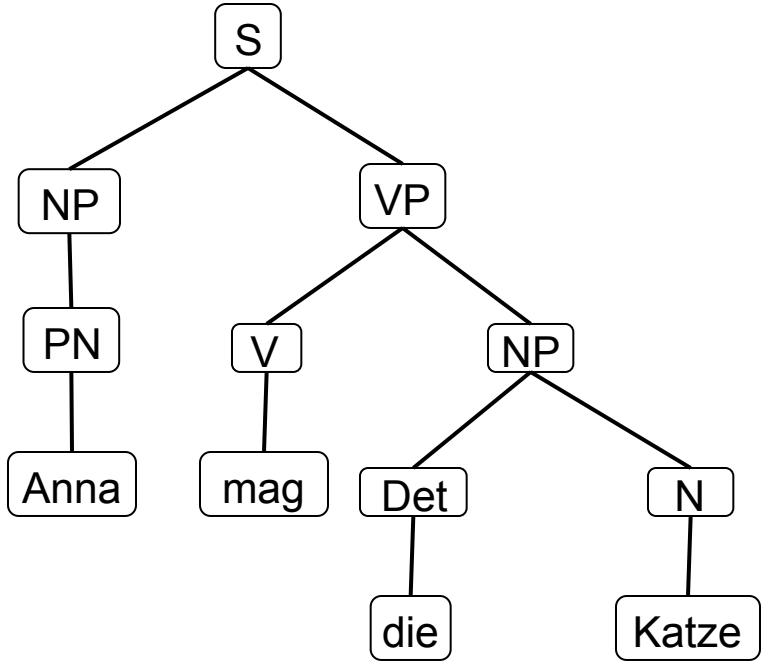
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die ***Katze***

Kontextfreier Top-Down Parser

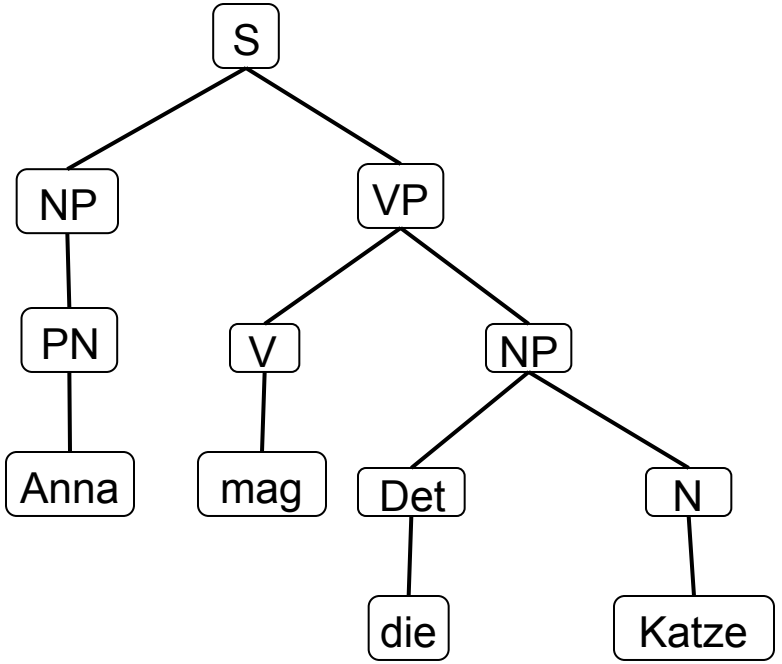
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

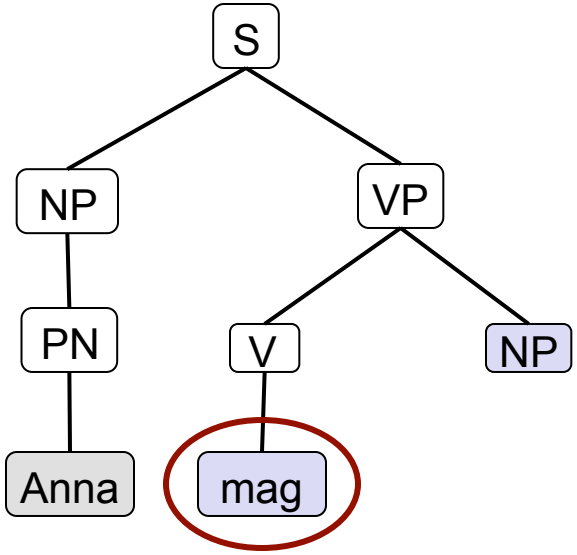
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



*Anna mag die Katze_*

Verbrauchte, aktive, aktueller Knoten

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



Anna mag die Katze

Kontextfreier Top-Down-Parser

- Die Operation des Top-Down-Parsers kann direkt auf Strukturbäumen dargestellt werden.
- Aktive Knoten sind alle Blattknoten eines Baums, die noch nicht verbraucht sind, d.h., entweder nicht-terminale Knoten oder terminale Knoten, die noch nicht mit der Eingabe abgeglichen wurden.
  – Aktive Knoten entsprechen genau dem Speicherinhalt.
- Initialisierung mit S. – Der aktuelle Knoten, auf dem der Parser operiert, ist der am weitesten links stehenden aktive Knoten.
  - Nicht-Terminalsymbol A: Wähle eine Ersetzungsregel mit linker Seite A und expandiere den aktiven Knoten entsprechend.
  - 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, wird die Eingabe akzeptiert und der erzeugte Parsebaum wird als Analyse ausgegeben.

Kontextfreier Top-Down Parser

S → NP VP

NP → Det N

NP → PN

VP → V NP

Det → *die*

N → *Katze*

V → *mag*

PN → *Anna*

Anna mag die Katze

Kontextfreier Top-Down Parser

S → NP VP

NP → Det N

NP → PN

VP → V NP

Det → *die*

N → *Katze*

V → *mag*

PN → *Anna*

S

Anna mag die Katze

Kontextfreier Top-Down Parser

S → NP VP

NP → Det N

NP → PN

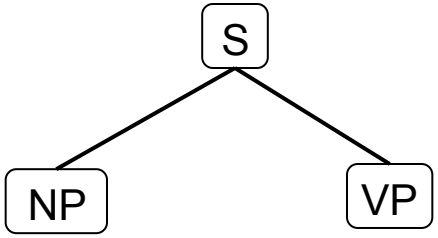
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

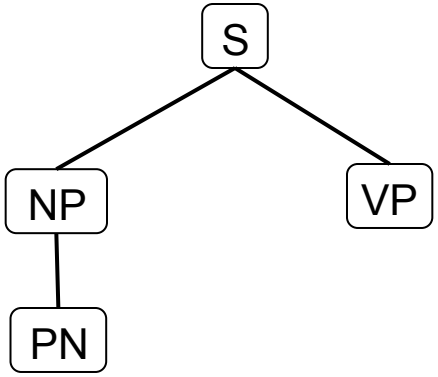
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

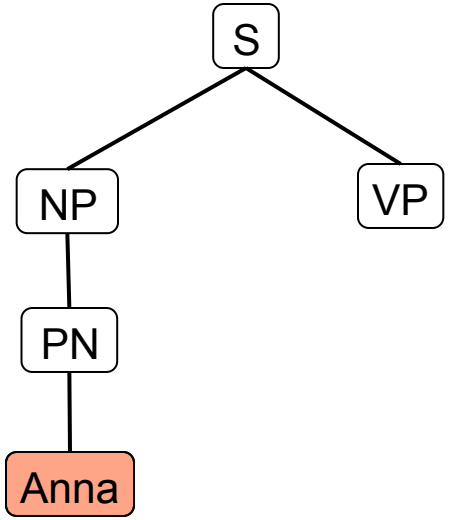
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



<u>Anna</u> mag die Katze

Kontextfreier Top-Down Parser

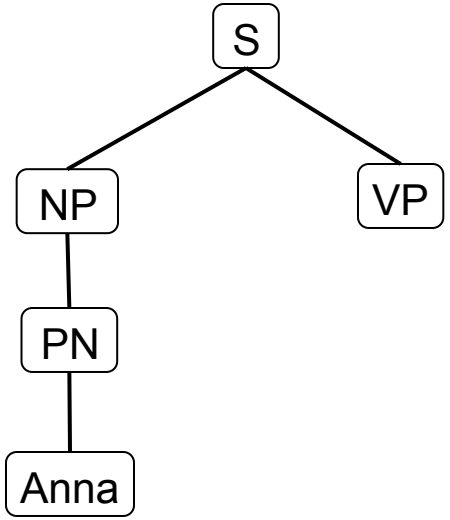
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

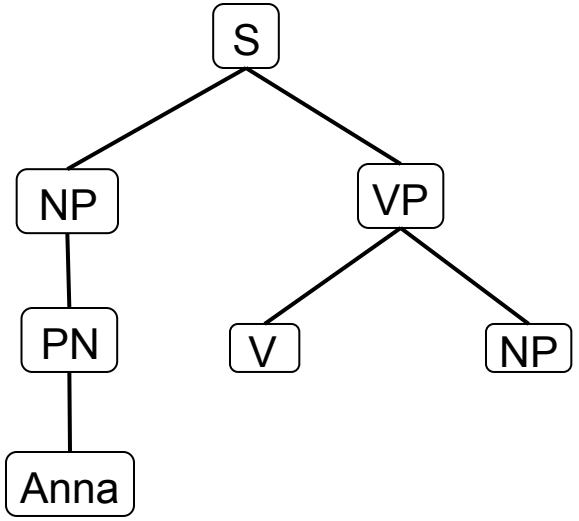
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

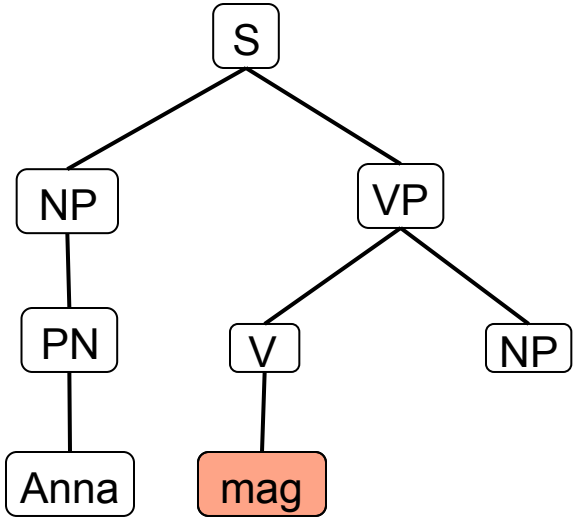
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

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



Anna mag die Katze

Kontextfreier Top-Down Parser

S → NP VP

NP → Det N

NP → PN

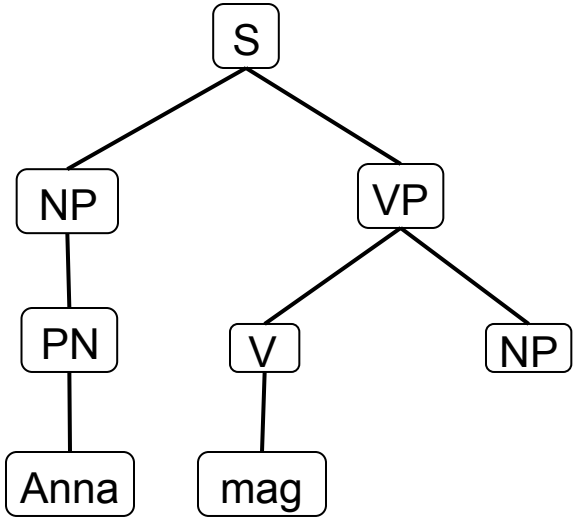
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

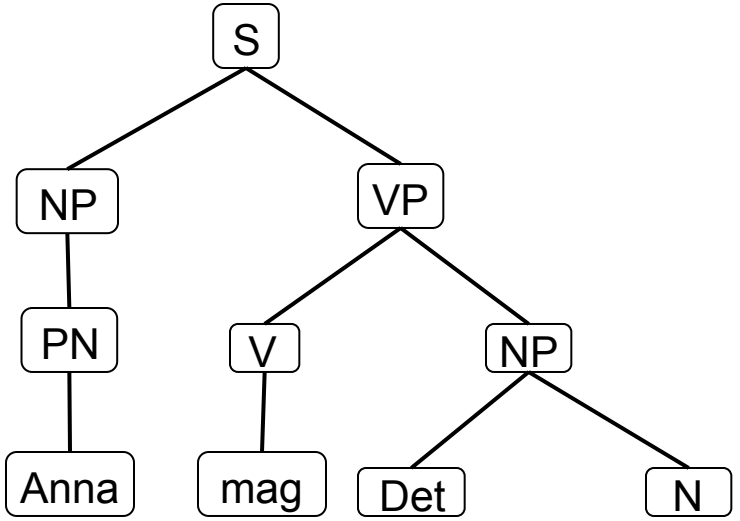
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

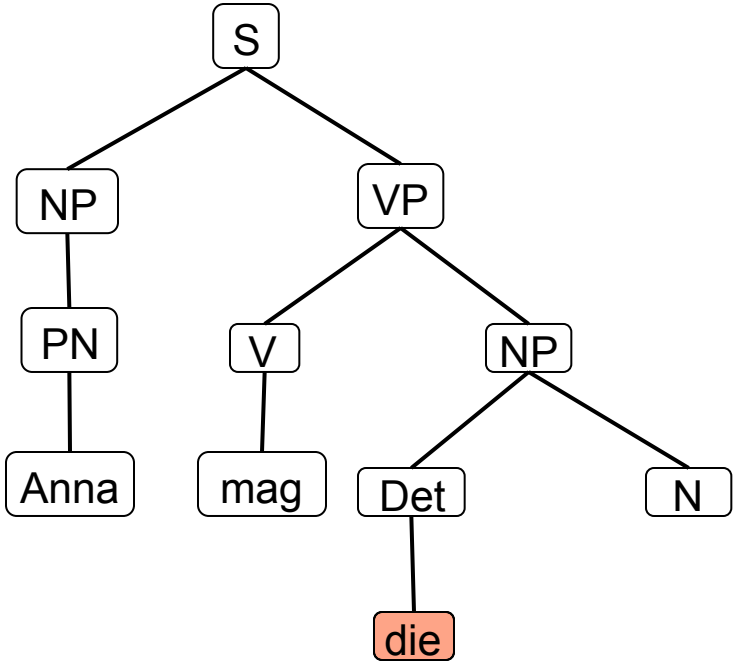
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

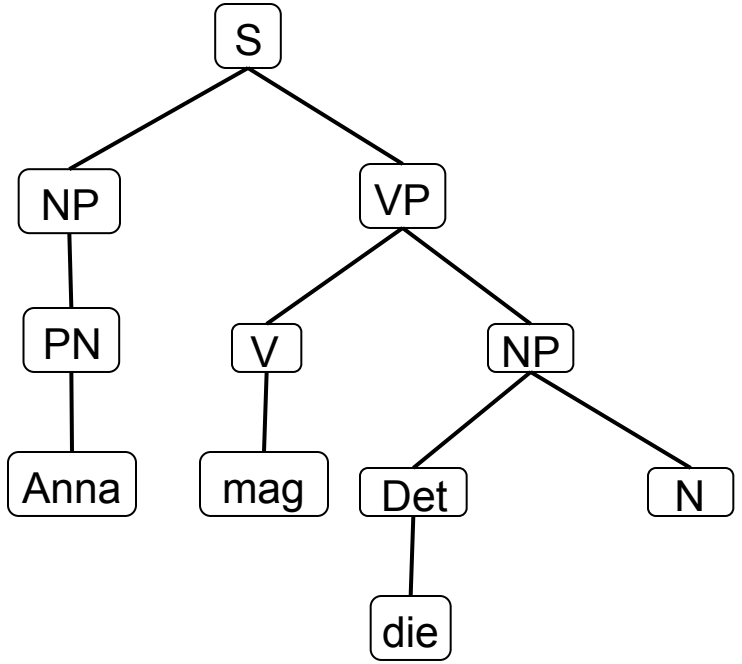
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

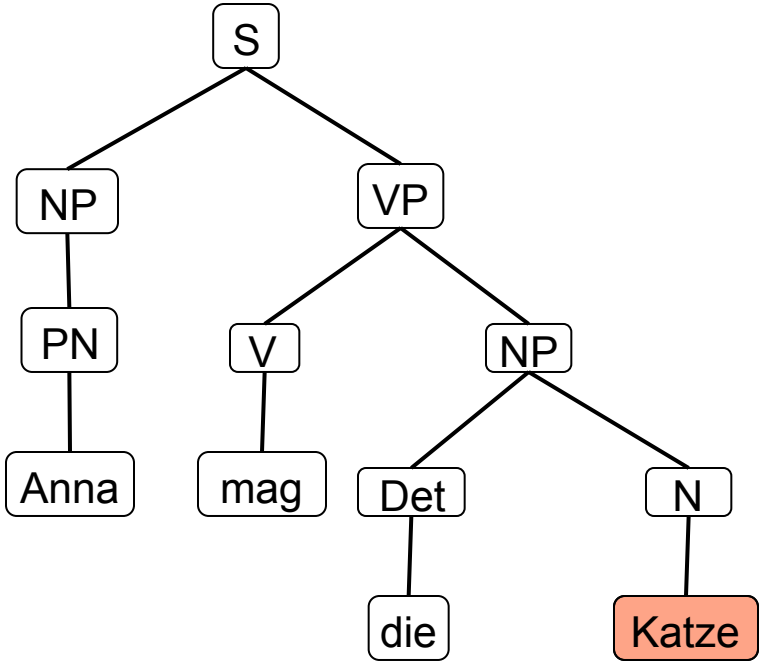
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die ***Katze***

Kontextfreier Top-Down Parser

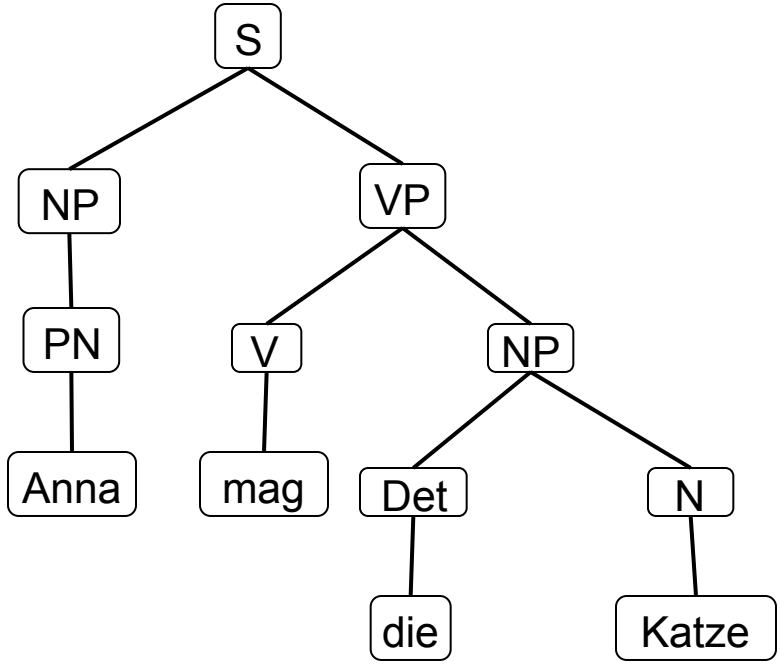
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

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*

S

Anna mag die Katze

Nicht-Determinismus in der Regelanwendung

S → NP VP

NP → Det N

NP → PN

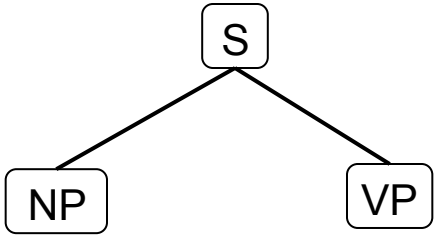
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

PN → *Anna*



<u>Anna</u> mag die Katze

Nicht-Determinismus in der Regelanwendung

S → NP VP

NP → Det N

NP → PN

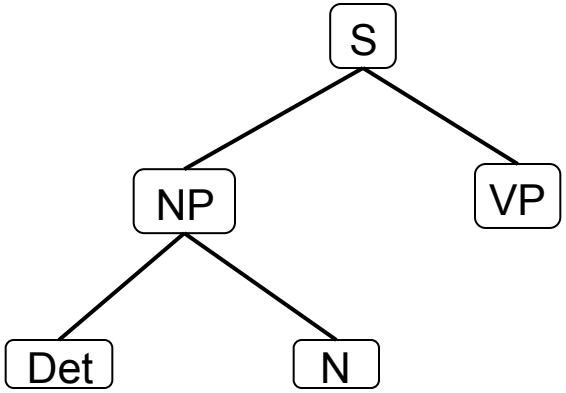
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

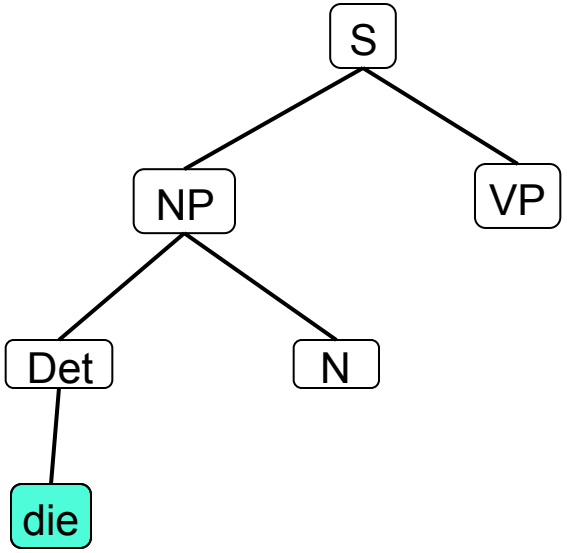
PN → *Anna*



<u>Anna</u> 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

Top-Down-Parser, Eigenschaften

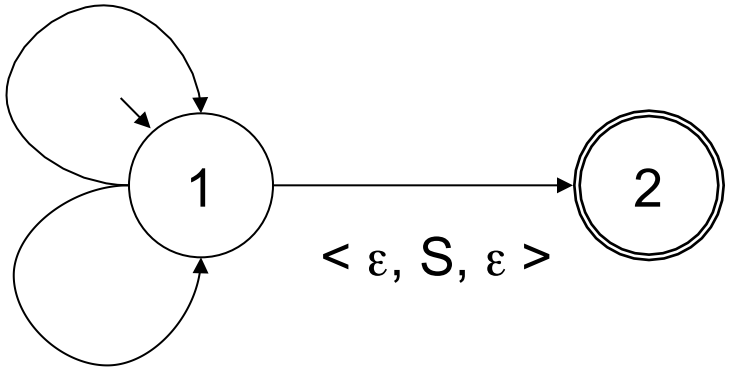
- Der Top-Down-Parser ist im Allgemeinen **nicht-deterministisch**: Für dasselbe Nichtterminal gibt es mehrere, eventuell sehr viele Ersetzungsregeln.
- Eine technische Lösung: Arbeiten mit einer Agenda, Last-In – First-Out, **Tiefensuche mit Backtracking**.
- Problem: Es werden viele Teilstrukturen 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

- Lies Symbole der Eingabekette und lege sie in den Stack.
- Wenn die **obersten n** Symbole im Stack umgekehrt gelesen die **rechte Seite** einer Grammatikregel **A → u** bilden, **ersetze** sie im Stack durch A.
- Wenn die Eingabe abgearbeitet ist und der Stack nur noch das Startsymbol S enthält, akzeptiere die Eingabe.
- Der **Bottom-Up** Parser heißt auch „**Shift-Reduce**“-Parser: Aktionen bestehen aus
  shift: Eingabesymbole werden in den Stack verschoben und
  reduce: zu Nicht-Terminalsymbolen reduziert.

Schema für den Shift-Reduce-Parser

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

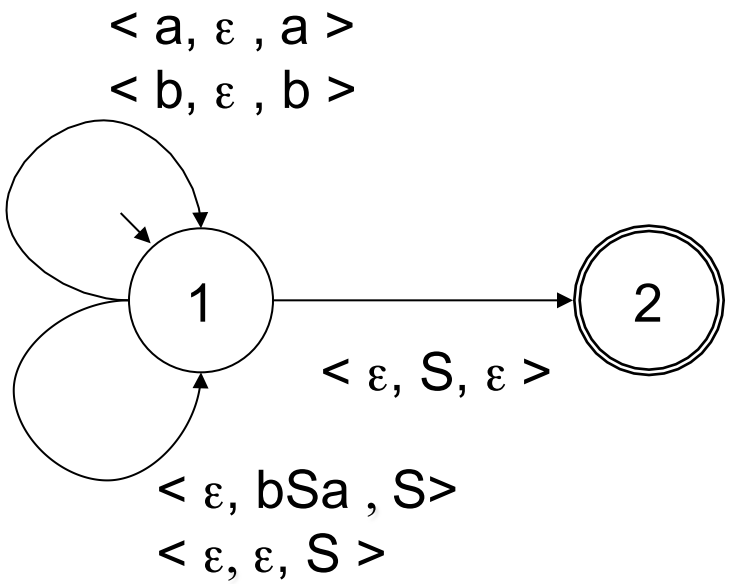


< ε, u^R, A > *(für jede Regel A → u)*

Beispiel: $a^n b^n$

S → aSb

S → ε



Shift-Reduce-Parser

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

Anna mag die Katze

Shift-Reduce-Parser

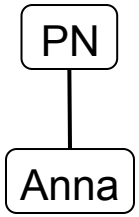
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

Anna

Anna mag die Katze

Shift-Reduce-Parser

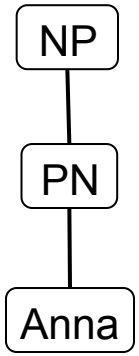
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Shift-Reduce-Parser

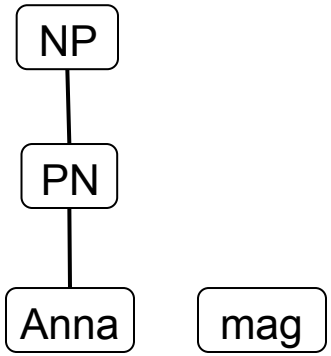
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Shift-Reduce-Parser

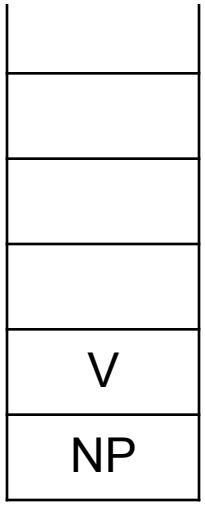
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

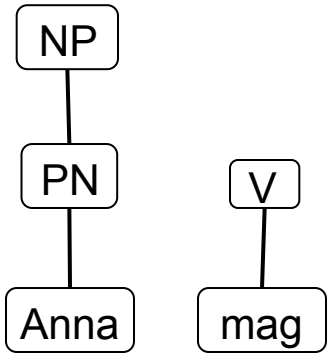


Anna mag die Katze

Shift-Reduce-Parser

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

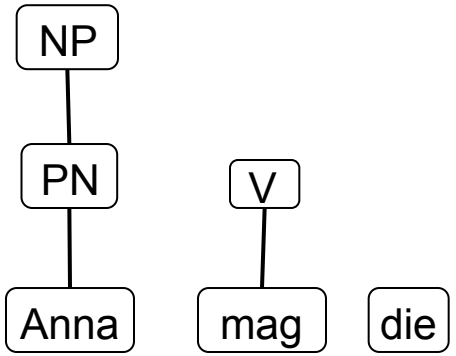




Anna mag die Katze

Shift-Reduce-Parser

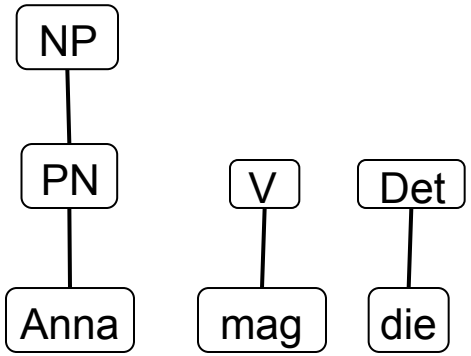
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



*Anna mag die **Katze***

Shift-Reduce-Parser

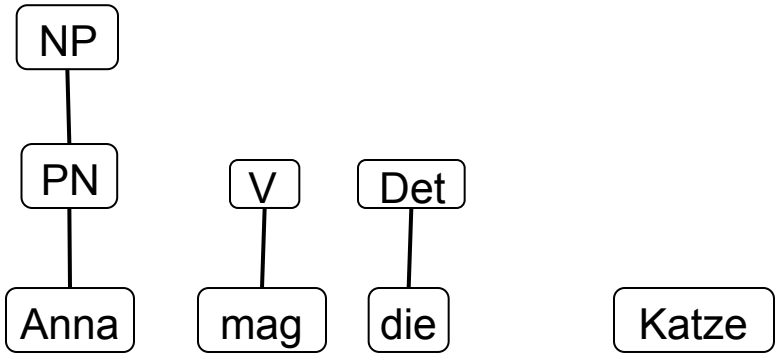
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Shift-Reduce-Parser

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



*Anna mag die Katze_*

Shift-Reduce-Parser

S → NP VP

NP → Det N

NP → PN

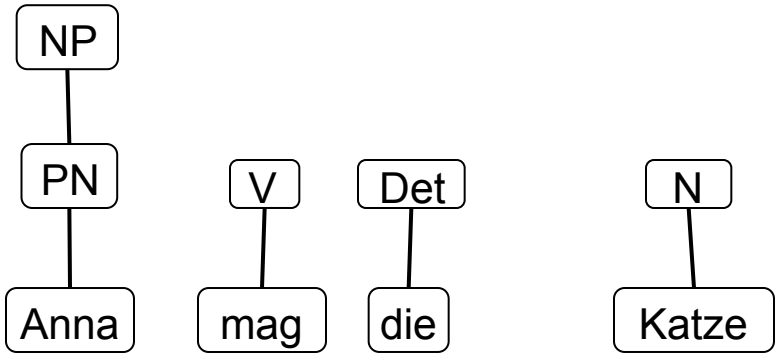
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

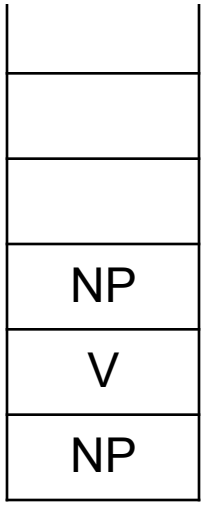
PN → *Anna*

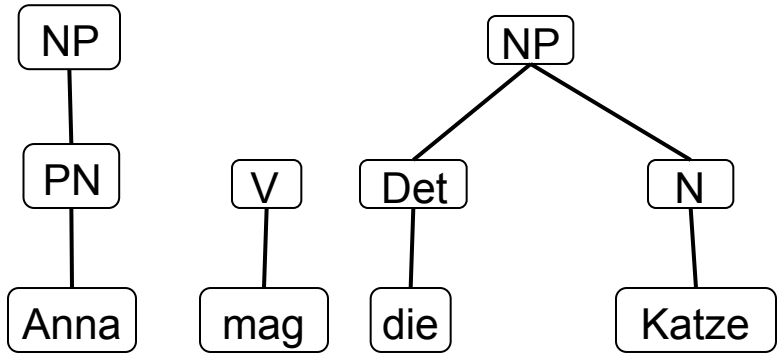


*Anna mag die Katze_*

Shift-Reduce-Parser

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

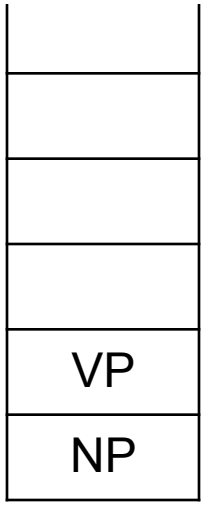


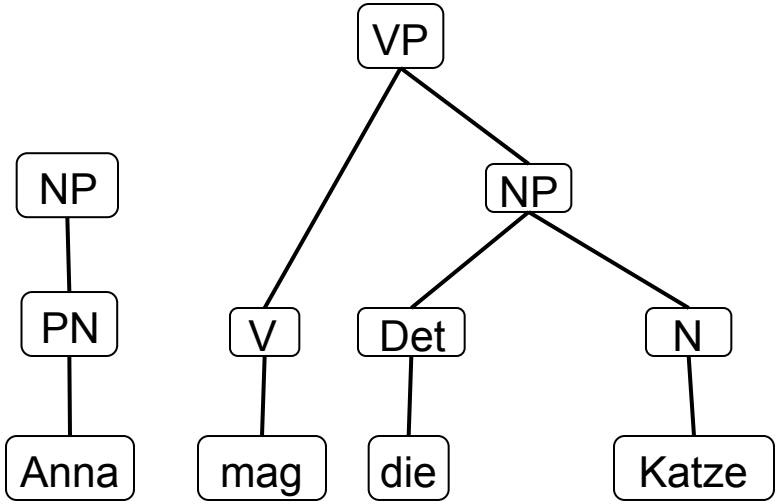


*Anna mag die Katze_*

Shift-Reduce-Parser

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

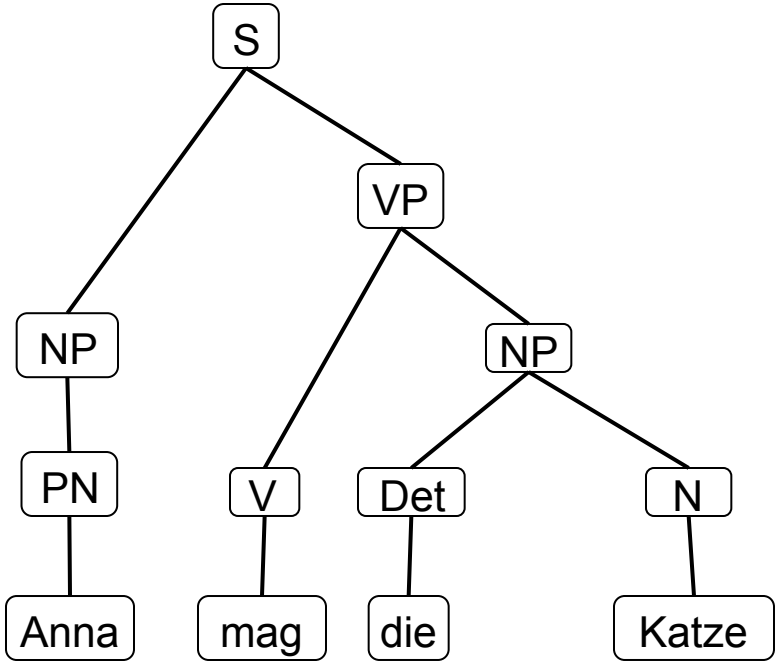




*Anna mag die Katze_*

Shift-Reduce-Parser

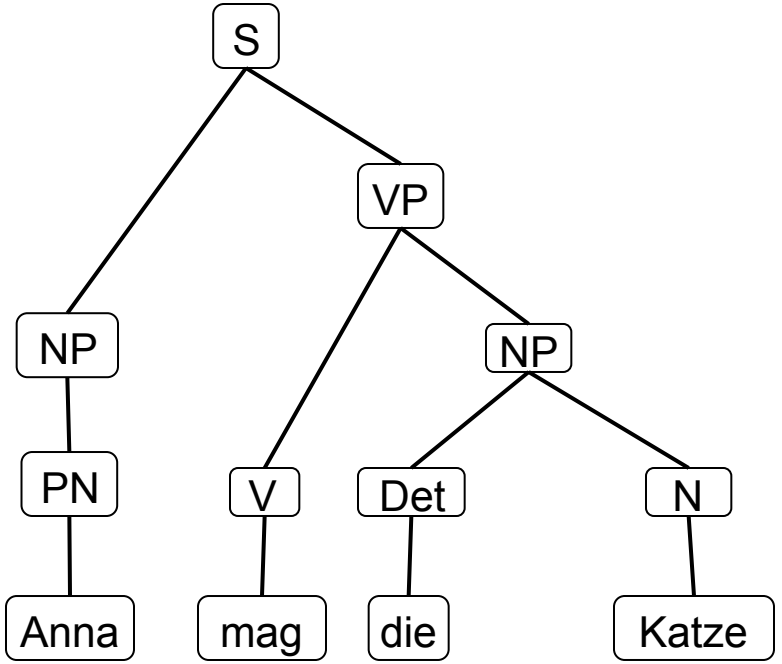
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



*Anna mag die Katze_*

Shift-Reduce-Parser

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



*Anna mag die Katze_*

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 sind nicht-deterministisch – auch für die einfache Grammatik, die $a^n b^n$ erkennt.
- Der Nicht-Determinismus ist aber nicht unbedingt essenziell: $a^n b^n$ kann durch einen deterministischen PDA erkannt werden.
- Frage: Gibt es für jede kontextfreie Sprache L einen PDA, der L erkennt?
- Können Sätze natürlicher Sprachen also in linearer Zeit syntaktisch analysiert werden?
- Die Antwort ist leider 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 | $w \in \{a,b\}^*$ }
  - L2 = { ww^R | $w \in \{a,b\}^*$ }
  - 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**. (sonst könnten sie auch keine syntaktischen Mehrdeutigkeit modellieren)

- Es gibt allerdings Parsing-Methoden, die **effizientere** Analysen als Top-Down- und Bottom-Up-Parser ermöglichen.

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.

Chart-Parsing

- **Beobachtung**: Teilstrukturen werden unnötigerweise immer wieder analysiert.

- **Idee**: Der Parser speichert Zwischenresultate in einer Datenstruktur, auf die er im weiteren Verlauf der Verarbeitung zurückgreifen kann.

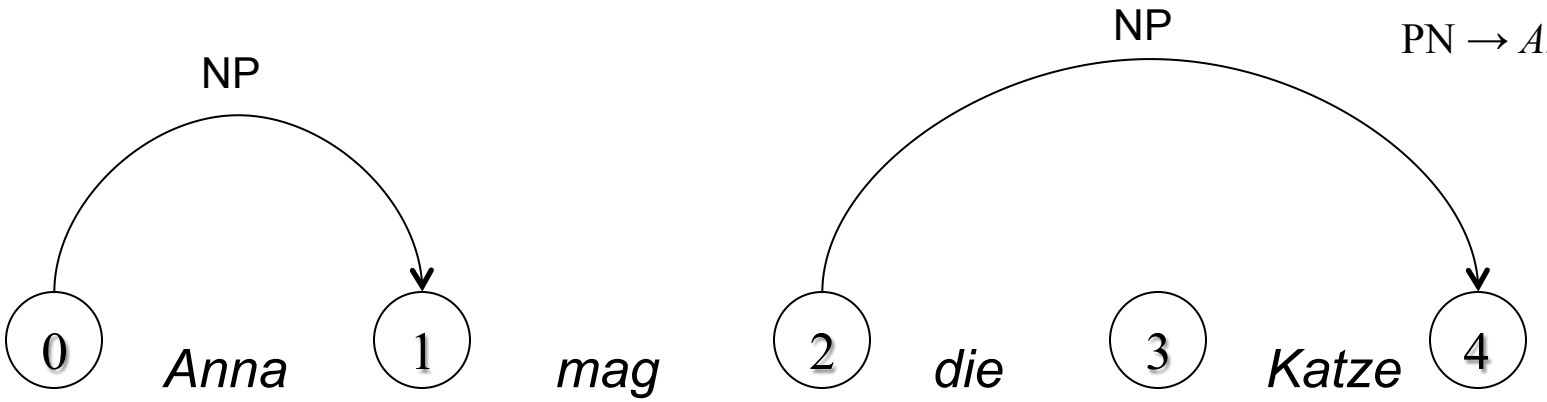
- Wir nennen die Datenstruktur „Chart“ und sprechen deshalb von „Chart-Parsing“.

Chart, graphische Darstellung

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

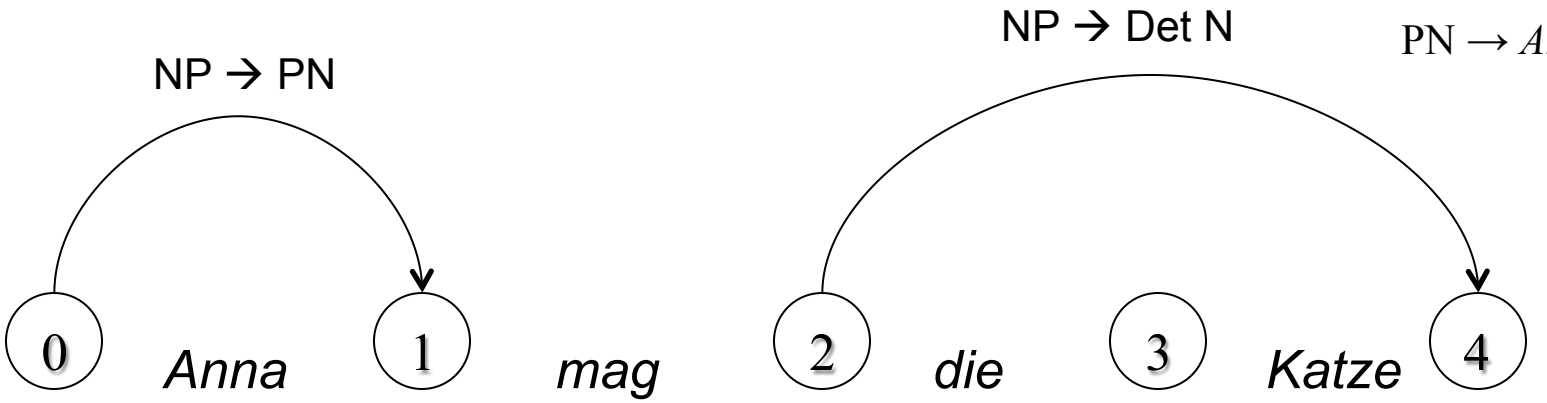
Chart, graphische Darstellung

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



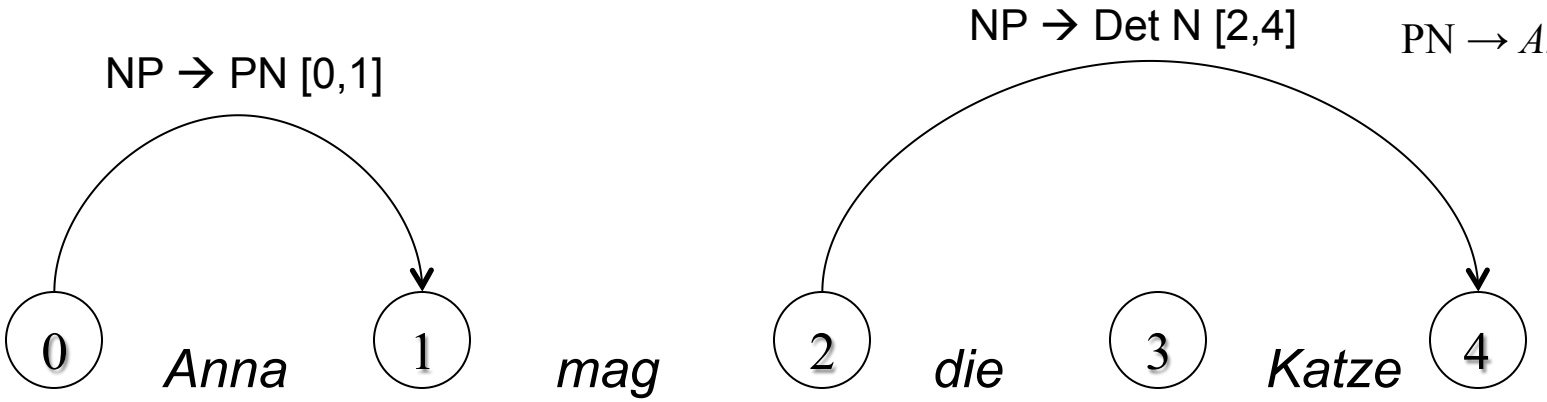
Chart, graphische Darstellung

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



Chart, graphische Darstellung

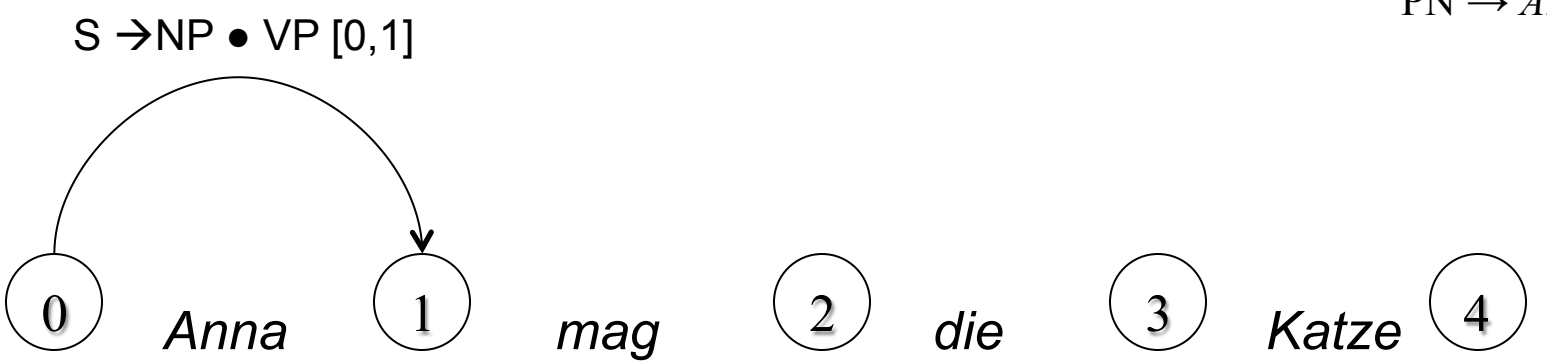
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Chart, graphische Darstellung

Aktive/ unvollständige Kante:

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



Beispiel-Chart

Aktive, initiale Kante

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

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“ der Form A→u●v: A→w mit w=uv ist Regel der Grammatik, der Punkt teilt die rechte Regelseite in den abgearbeiteten und den noch offenen Teil.
  - Paar [i, j] bezeichnet den Teilstring der Eingabekette, auf dem dies Ergebnis erzielt wurde.
- Beispiel: <S → NP ● VP, [0, 2]>
  - Regel S → NP VP wurde in Position 0 angesetzt, an Position 2 ist eine NP erkannt, von Position 2 an muss noch eine VP erkannt werden.

Chart, graphische Darstellung

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

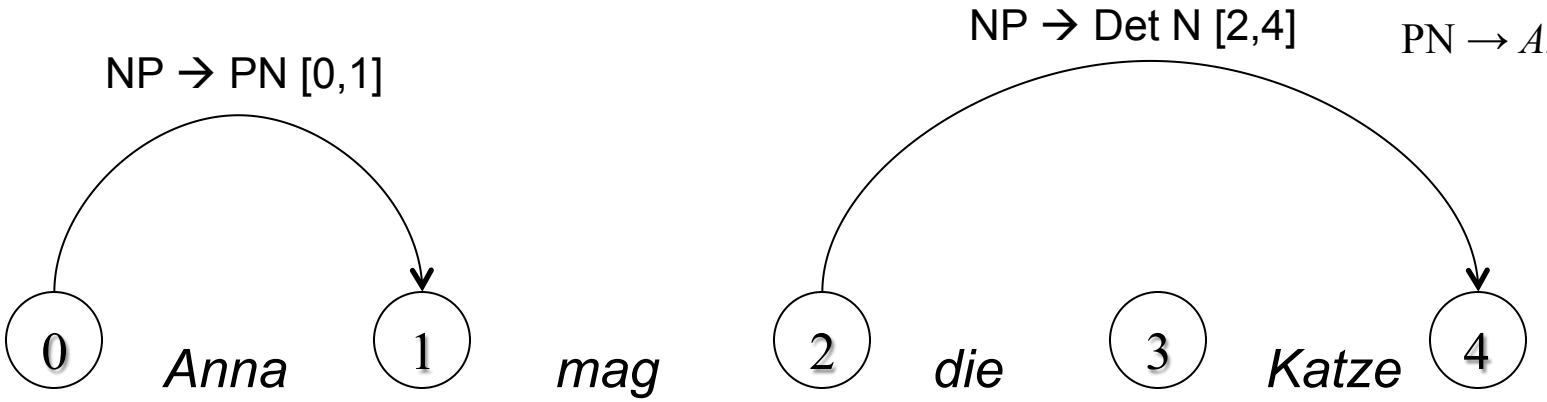


Chart: Tabellen-Darstellung

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

Chart, graphische Darstellung

Aktive/ unvollständige Kante:

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

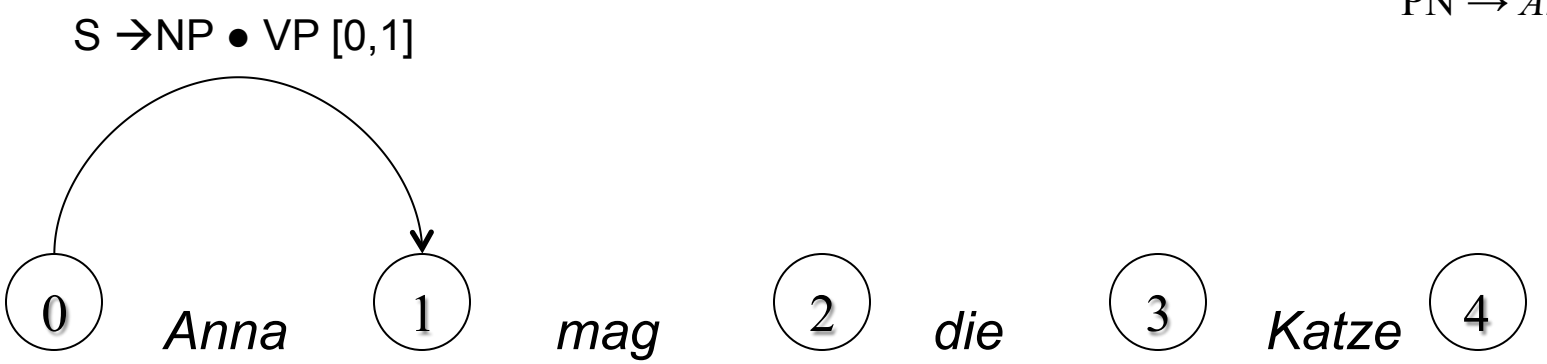


Chart: Tabllen-Darstellung

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

Earley-Algorithmus

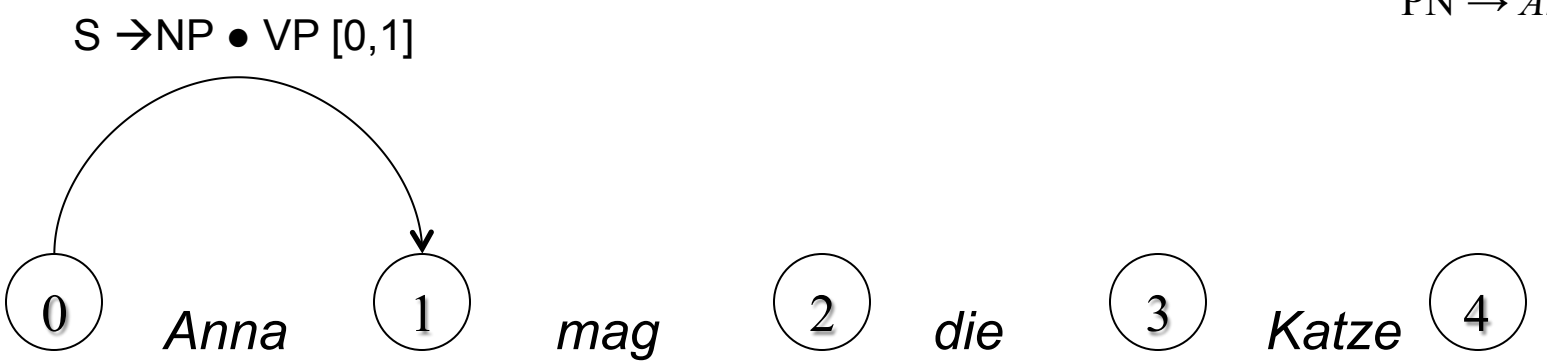
- Ein Standard-Algorithmus für das Chart-Parsing ist der Earley-Algorithmus, der von links nach rechts durch den Eingabesatz geht und in einer Top-Down-Strategie alle Analyse-Alternativen parallel verfolgt.

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 auf jeden Eintrag <P, [i, j]> die Operationen Scan, Predict, und Complete so lange an, bis keine neue Anwendung mehr möglich ist. Dann gehe zu j+1.
- Operationen des Earley-Algorithmus:
  - 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.

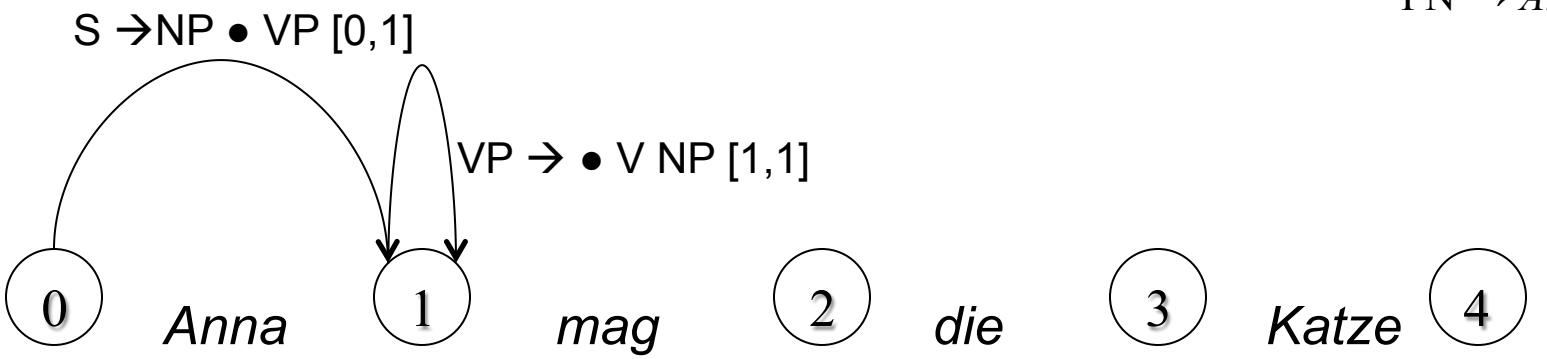
Predictor

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

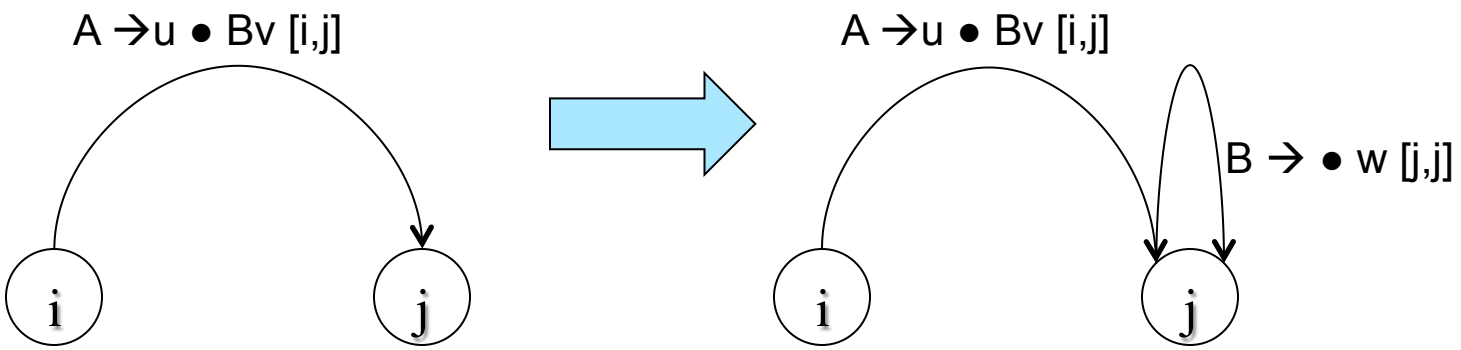


Predictor

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



Predictor, Schema



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 → w der Grammatik <B → • w, [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 Ausdrücke der Kategorie B erzeugen.

Scanner: Beispiel

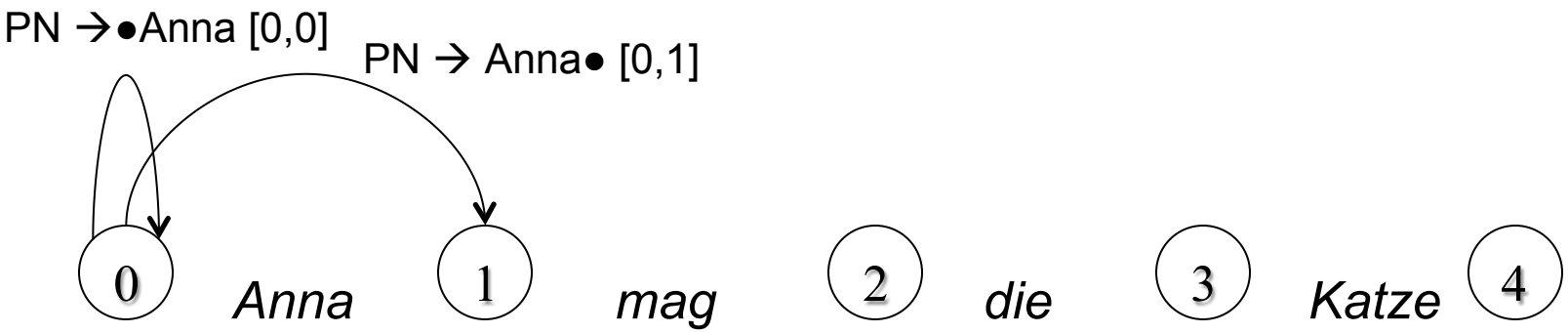
Scanner: Beispiel

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

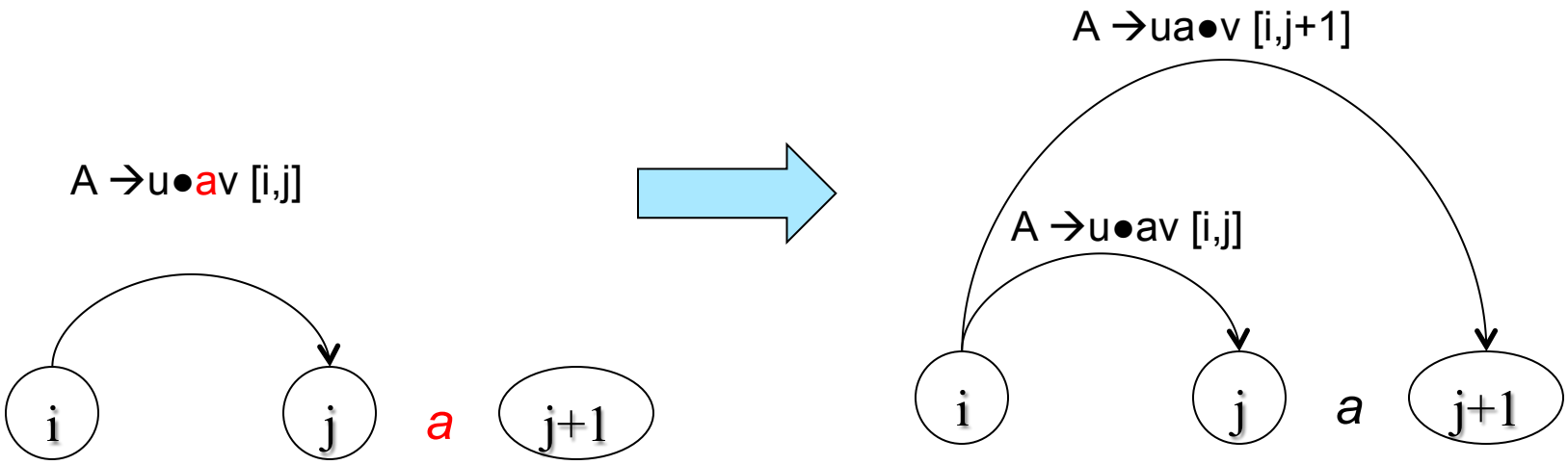
PN →●Anna [0,0]

Scanner: Beispiel

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



Scanner, Schema



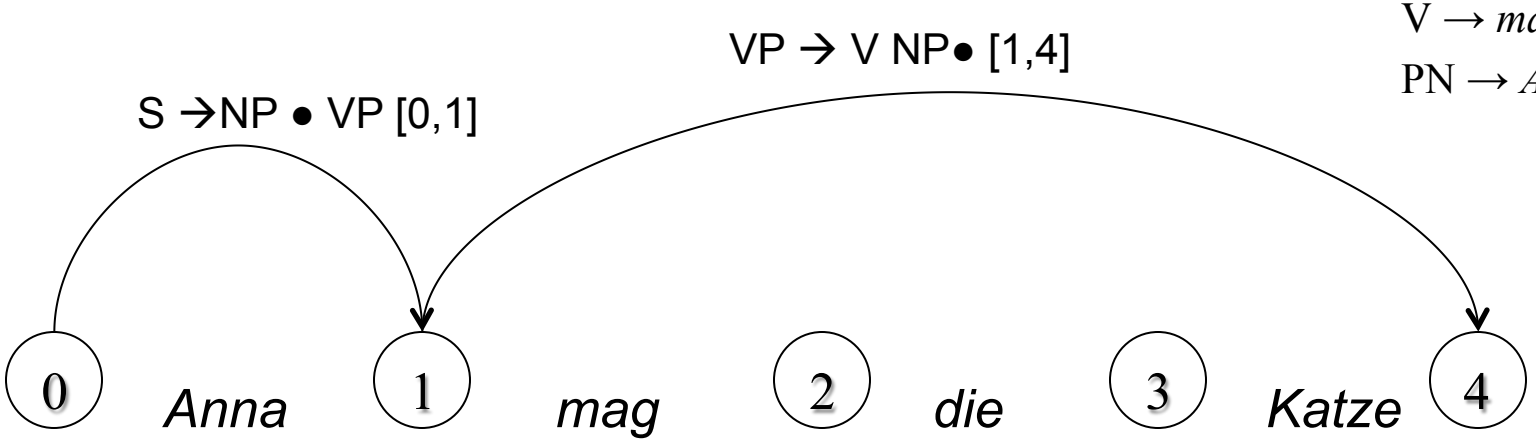
Earley-Algorithmus – Scanner

- 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.

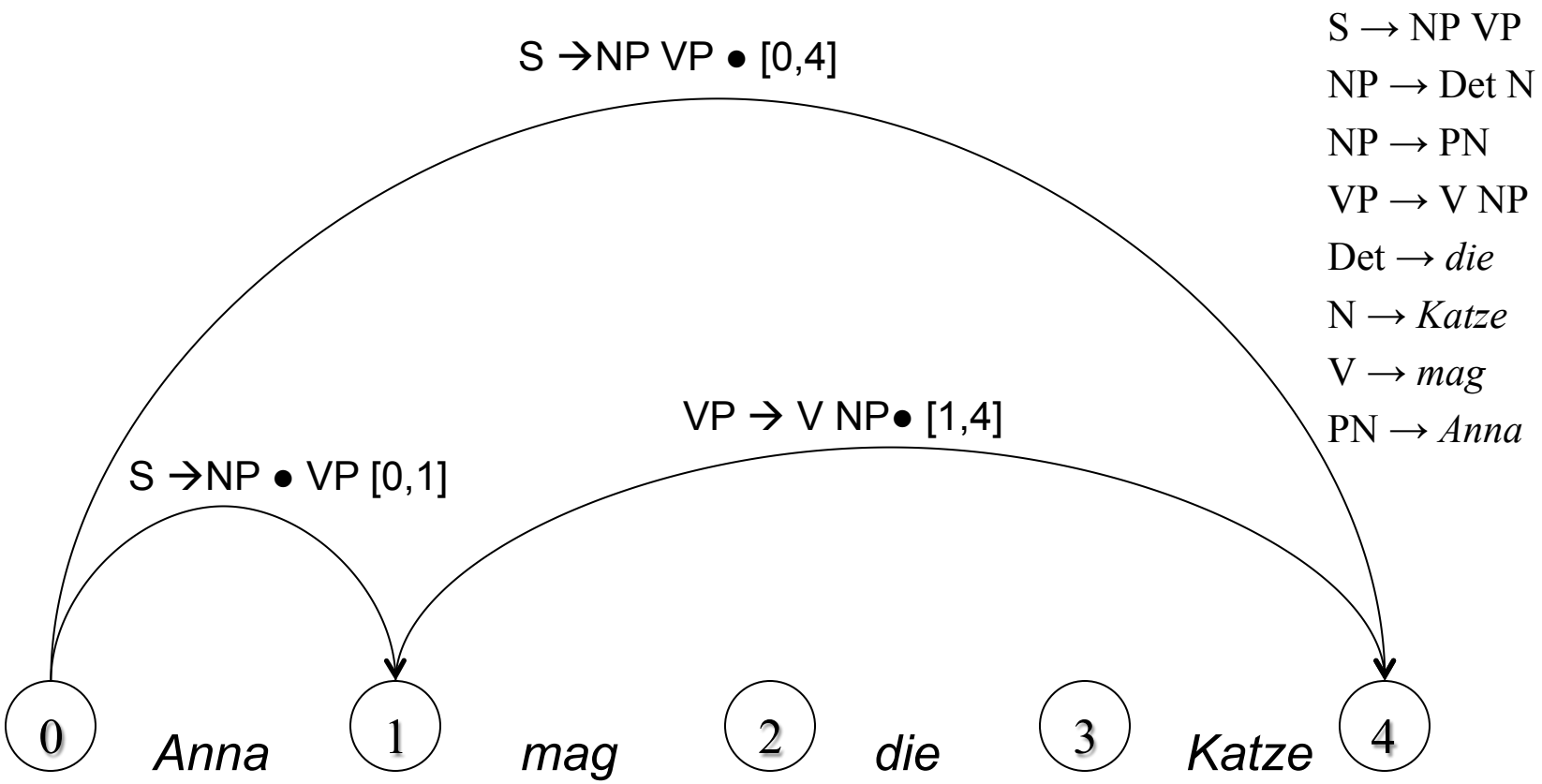
- Die punktierte Regel A → u ● av drückt aus, dass als nächstes das Eingabesymbol a erwartet wird. Der Scanner liest das Eingabewort und gleicht es mit der Regel ab. Im Erfolgsfall wird der Punkt über a hinweg geschoben und ein neuer Eintrag in die nächste Spalte der Chart gemacht.

Completer, Beispiel

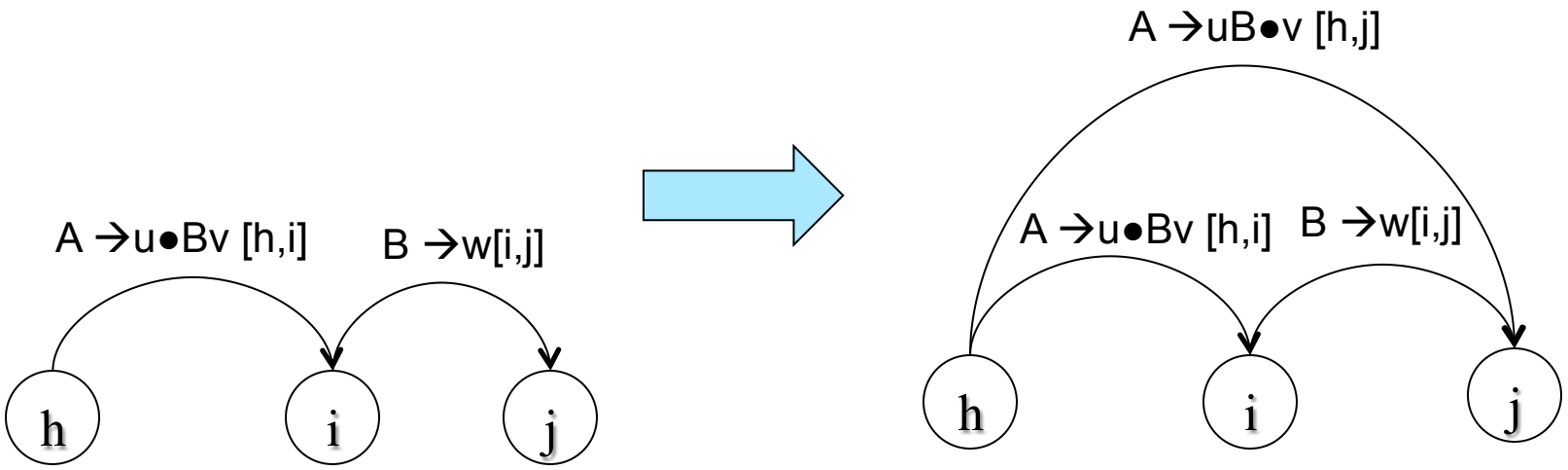
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Complete, Beispiel



Completer, Schema



Earley-Algorithmus – Completer

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

- B → w ● 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.

Earley-Algorithmus: Abschluss

- Der Aufbau der Chart ist abgeschlossen, wenn der n+1-te Schritt beendet ist.

- Die Eingabe wird akzeptiert, wenn die Chart <S → u● , [0, n]> enthält.

- 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]				

$S \rightarrow NP\ VP$
$NP \rightarrow Det\ N$
$NP \rightarrow PN$
$VP \rightarrow V\ NP$
$Det \rightarrow \textit{die}$
$N \rightarrow \textit{Katze}$
$V \rightarrow \textit{mag}$
$PN \rightarrow \textit{Anna}$

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]				

$S \rightarrow NP\ VP$
$NP \rightarrow Det\ N$
$NP \rightarrow PN$
$VP \rightarrow V\ NP$
$Det \rightarrow \textit{die}$
$N \rightarrow \textit{Katze}$
$V \rightarrow \textit{mag}$
$PN \rightarrow \textit{Anna}$

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]			

$S \rightarrow NP\ VP$
$NP \rightarrow Det\ N$
$NP \rightarrow PN$
$VP \rightarrow V\ NP$
$Det \rightarrow \textit{die}$
$N \rightarrow \textit{Katze}$
$V \rightarrow \textit{mag}$
$PN \rightarrow \textit{Anna}$

Beispiel-Chart: Operationen auf Pos. 2

Position 0

S → ●NP VP, [0,0]

Predictor:
NP → ●Det N, [0,0]

NP → ●PN, [0,0]

Det → ●*die*, [0,0]

PN → ●*Anna*, [0,0]

Position 1

Scanner:

PN → *Anna*●, [0,1]

Completer:

NP → PN●, [0,1]

S → NP●VP, [0,1]

Predictor:

VP → ●V NP, [1,1]

V → ●*mag*, [1,1]

Position 2

Scanner:

V → *mag*●, [1,2]

Completer:

VP → V●NP, [1,2]

Predictor:

NP → ●Det N, [2,2]

NP → ●PN, [2,2]

Det → ●*die*, [2,2]

PN → ●*Anna*, [2,2]

Position 3

Scanner:

Det → *die*●, [2,3]

Position 4

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

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

$S \rightarrow NP\ VP$
$NP \rightarrow Det\ N$
$NP \rightarrow PN$
$VP \rightarrow V\ NP$
$Det \rightarrow \textit{die}$
$N \rightarrow \textit{Katze}$
$V \rightarrow \textit{mag}$
$PN \rightarrow \textit{Anna}$

Beispiel-Chart: Operationen auf Pos. 4

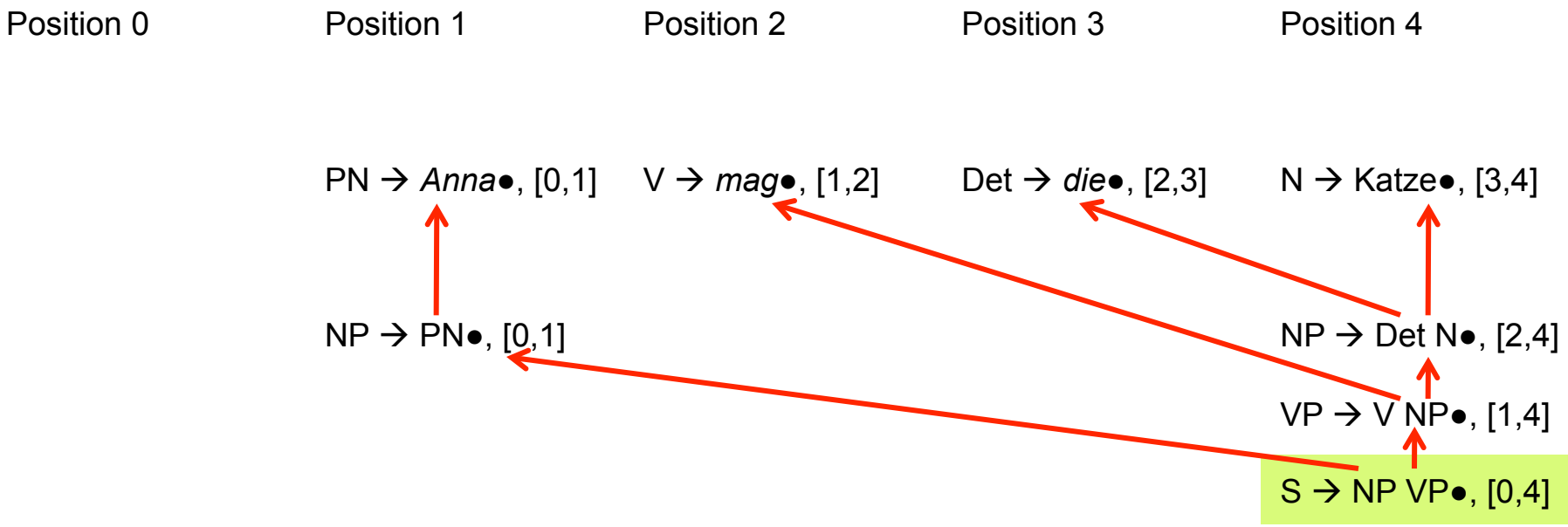
Position 0	Position 1	Position 2	Position 3	Position 4
S → ●NP VP, [0,0]	Scanner:	Scanner:	Scanner:	Scanner:
Predictor:	PN → *Anna*●, [0,1]	V → *mag*●, [1,2]	Det → *die*●, [2,3]	N → Katze●, [3,4]
NP → ●Det N, [0,0]	Completer:	Completer:	Completer:	Completer:
NP → ●PN, [0,0]	NP → PN●, [0,1]	VP → V●NP,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



Beispiel-Chart

- Alternative grafische Repräsentation der vollständigen Einträge