Einführung in die Computerlinguistik

Parsing

WS 2021/2022
Vera Demberg

Automaten vs. CFG

Warum kann man endliche Automaten nicht für die Verarbeitung von Sätzen (Syntax) verwenden?

Eine kontextfreie Grammatik für deutsche Sätze

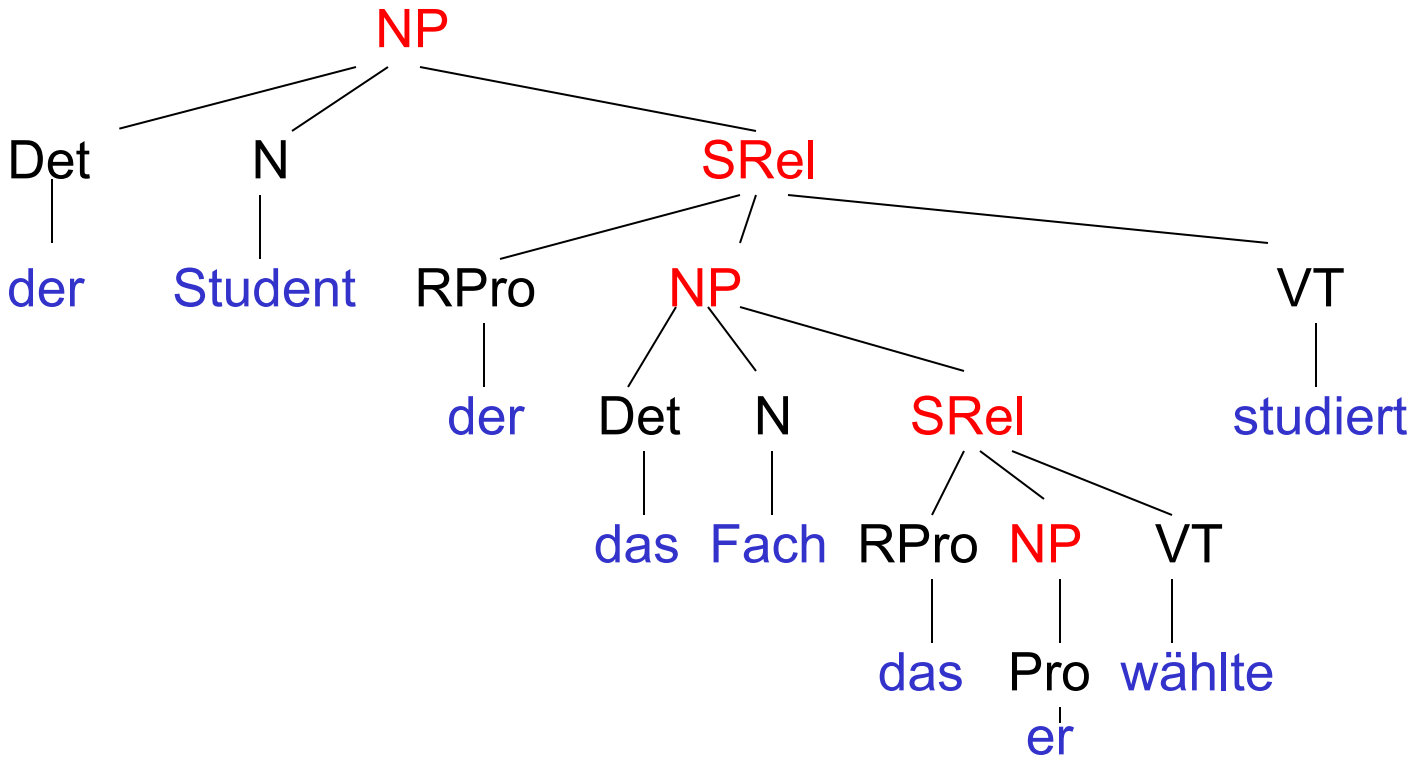
G1 = <V, Σ, P, S> mit

V = {S, SRel, NP, VI, VT, N, Det, RPro, Pro} $\cup$ Σ

Σ = {*schläft, arbeitet, studiert, wählte, Student, Fach, der, das, er, sie*}

S $\rightarrow$ NP VI	S $\rightarrow$ NP VT NP
SRel $\rightarrow$ RPro VI	SRel $\rightarrow$ RPro NP VT
NP $\rightarrow$ Det N (SRel)	NP $\rightarrow$ Pro
VI $\rightarrow$ *schläft* | *arbeitet* | *liest*	VT $\rightarrow$ *wählte* | *studiert*
N $\rightarrow$ *Student* | *Fach* | *Buch*	RPro $\rightarrow$ *der* | *das*
Det $\rightarrow$ *der* | *das* | *ein*	Pro $\rightarrow$ *er* | *sie*

CFG beschreibt auch **die Struktur**



CFG: Konstituentenstruktur

- Anders als endliche Automaten beschreibt eine CFG nicht nur die zulässigen Ausdrücke einer Sprache, sondern **gibt ihnen implizit auch eine Struktur**.

- Sie ordnet den Sätzen der Sprache Ableitungsbäume zu (auch „Parse-Bäume“ genannt, Parsing = automatische syntaktische Analyse).

Syntaktische Struktur und semantische Interpretation

- Die syntaktische Struktur ist Grundlage für die semantische Interpretation.
- Beispiel Arithmetik:
  - Terme bezeichnen („bedeuten“) Zahlen
  - Operatoren bilden (Paare von) Termbedeutungen/Zahlen auf Termbedeutungen/Zahlen ab.
  - Bedeutung einer Gleichung ist ein Wahrheitswert.
- Die Bedeutung des Gesamtausdrucks wird „kompositionell“, entlang der syntaktischen Struktur berechnet.
- Strukturelle Mehrdeutigkeit, wenn Klammern und Klammerkonventionen fehlen.
  Beispiel: 3+4*5 =23 oder =35?

Grammatische Mehrdeutigkeit

- Grammatiken für formale Sprachen werden so definiert, dass strukturelle Mehrdeutigkeit in jedem Fall vermieden wird.
- Natürliche Sprachen sind strukturell mehrdeutig:

  Peter sah den Mann mit dem Teleskop

- Eine Grammatik, die die Mehrdeutigkeit modelliert:

  $S \rightarrow NP\ VP$ $NP \rightarrow ART\ N' \mid EN$ $N' \rightarrow N'\ PP$ $N' \rightarrow N\ (NP)$

  $VP \rightarrow V'$ $V' \rightarrow V'\ PP$ $V' \rightarrow VT\ NP$ $V' \rightarrow VI$

  $PP \rightarrow P\ NP$

- Die zwei Analysevarianten (Zwischenknoten z.T. weggelassen)
  - [$_S$ *Peter* [$_{VP}$ *sah* [$_{NP}$ *den* [$_{N'}$ *Mann* [$_{PP}$ *mit dem Teleskop*]]]]]
  - [$_S$ *Peter* [$_{VP}$ [$_{V'}$ [$_{V'}$ *sah* [$_{NP}$ *den Mann*]] [$_{PP}$ *mit dem Teleskop*]]]]

Syntaktische Struktur und semantische Interpretation

93.000€ in Gütersloh gefunden in der Handtasche einer Rentnerin, die auf einem Friedhof am Lenker eines Fahrrads baumelte.

Aus dem Spiegel, Rubrik „Hohlspiegel“

Syntaktische Struktur und semantische Interpretation

Siemens ist wegen seiner Pläne für einen massiven Personalabbau im Bundestag von allen Parteien kritisiert worden.

Tagesschau, 21.11.2017

Inhalte der VL

Wie können wir Satzstruktur automatisch berechnen?

- 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.
- Bei endlichen Automaten hingegen konnten wir einen **Algorithmus**, ein Suchverfahren spezifizieren, das beantwortet, ob ein Wort zu einer Sprache gehört (= ob ein Wort gegeben die morphologischen Regeln ein gültiges Wort der Sprache ist).

Inhalte der VL

Wie können wir Satzstruktur automatisch berechnen?

- **vom Endlichen Automaten zum Kellerautomaten**
- Kellerautomat vs. CFG
- Parsen mit CFG / Kellerautomat
  - Top-down Parser
  - Bottom-up Parser (= “Shift-Reduce Parser”)

Endliche Automaten -- 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.

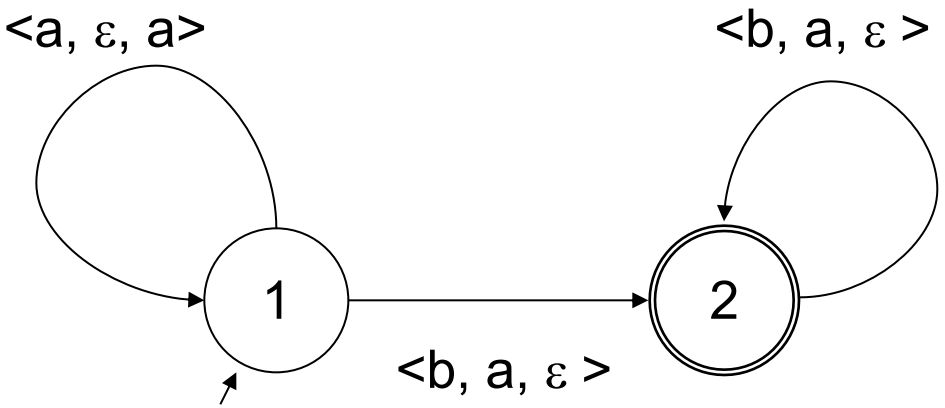
- Wenn Eingabewort und gespeicherte Folge von a‘s gleichzeitig aufgebraucht sind, wird das Eingabewort akzeptiert.

- 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)**.

<lies, lösche, schreibe>

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.

Inhalte der VL

- Kellerautomat
- **Kellerautomat ⟷ CFG**
- Parsen mit CFG / Kellerautomat
  - Top-down Parser
  - Bottom-up Parser (= “Shift-Reduce Parser”)

Keller-Automaten und kontextfreie Grammatiken

- Kontextfreie Grammatiken (CFGs) erlauben die **einfache und elegante Definition** von kontextfreien Sprachen – haben aber zunächst kein 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 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.

<lies, lösche, schreibe>

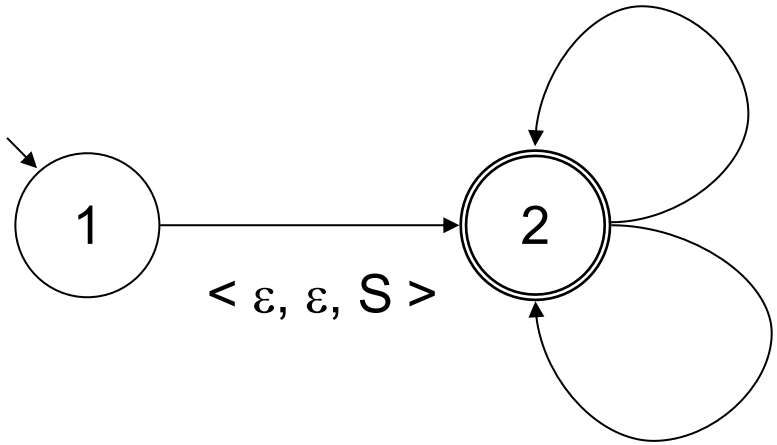
Wie könnte ein Kellerautomat für die $a^n b^n$ CFG aussehen?

- S -> aSb
- S-> ε

<lies, lösche, schreibe>

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.



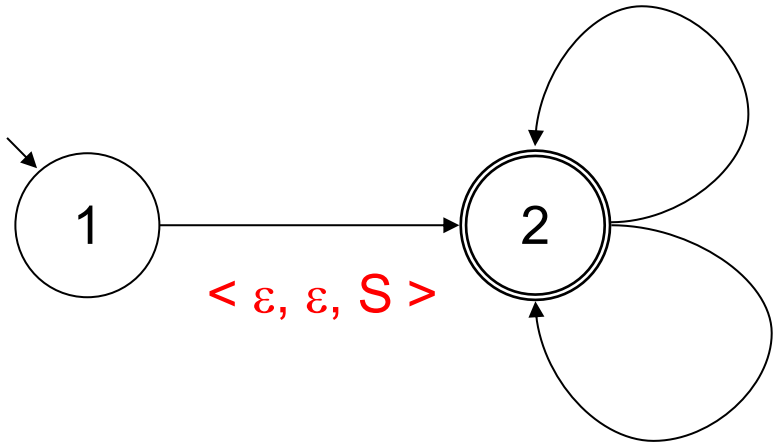
< ε, A, u > *(für jede Regel A→u)*

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

<lies, lösche, schreibe>

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 → 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.



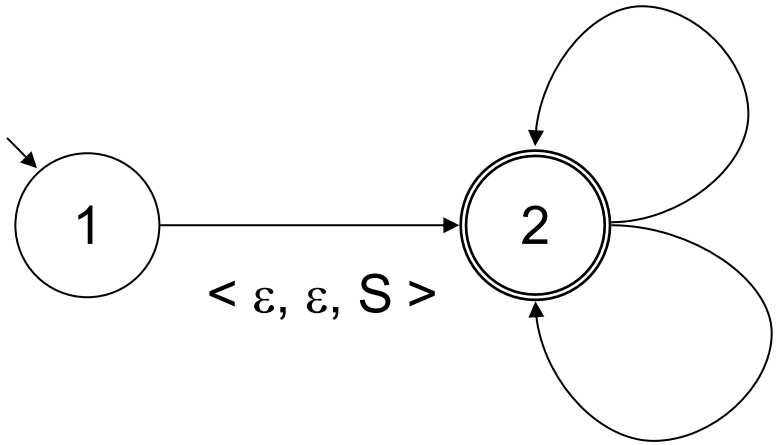
< ε, A, u > *(für jede Regel A→u)*

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

<lies, lösche, schreibe>

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 → 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.



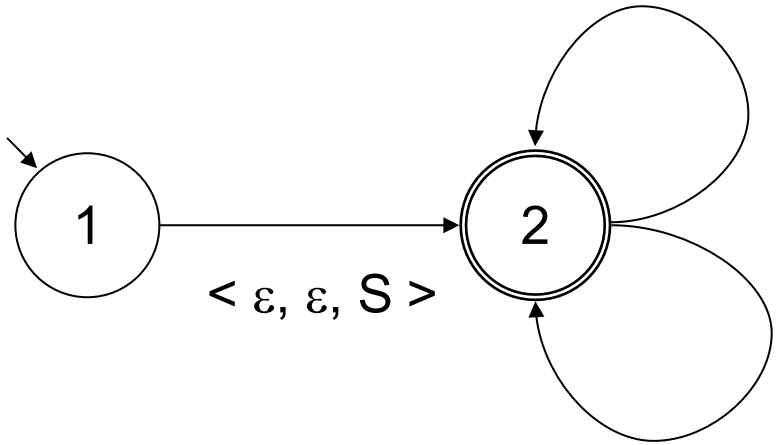
< ε, A, u > *(für jede Regel A→u)*

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

<lies, lösche, schreibe>

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 → 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.**



< ε, A, u > *(für jede Regel A→u)*

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

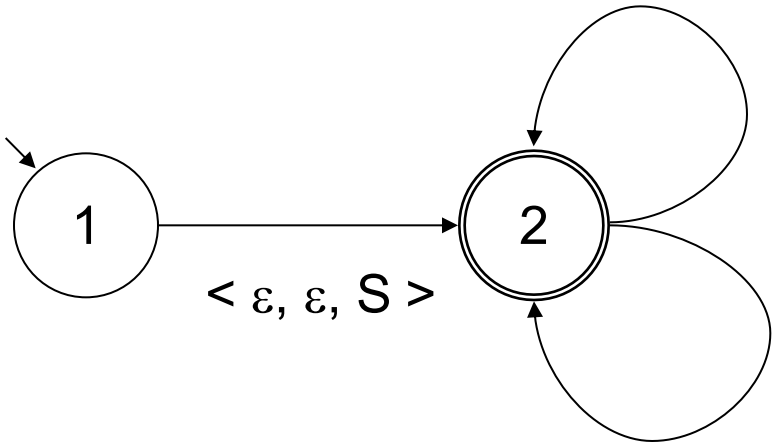
<lies, lösche, schreibe>

Beispiel: $a^n b^n$

S → aSb

S → ε

< ε, A, u > *(für jede Regel A→u)*



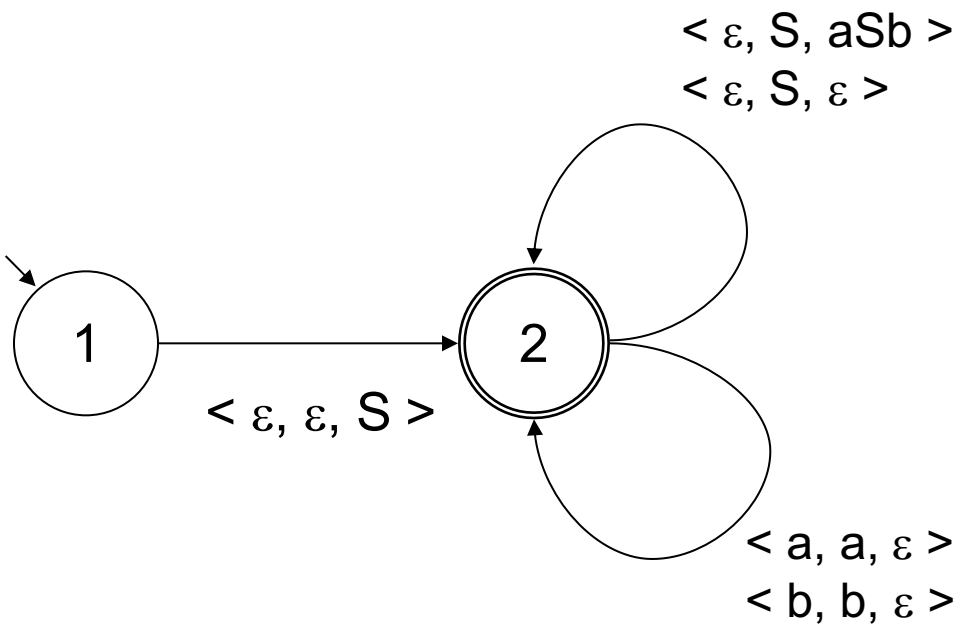
< a, a, ε > *(für jedes Terminalsymbol a)*

<lies, lösche, schreibe>

Beispiel: $a^n b^n$

S → aSb

S → ε

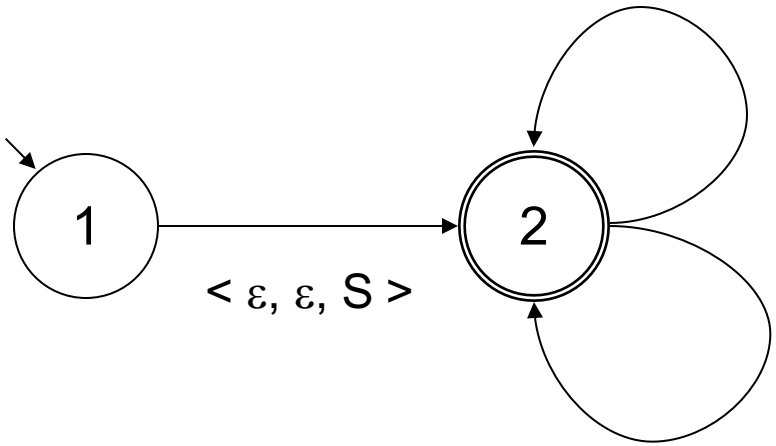


<lies, lösche, schreibe>

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, A, u > *(für jede Regel A→u)*



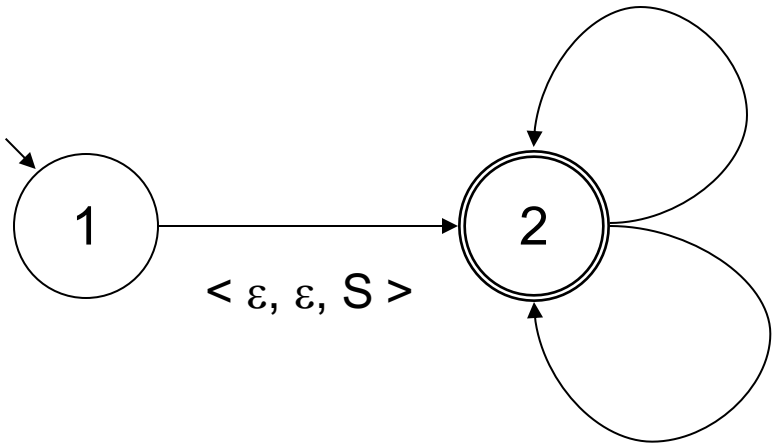
< a, a, ε > *(für jedes Terminalsymbol a)*

<lies, lösche, schreibe>

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>

Inhalte der VL

- Kellerautomat
- Kellerautomat $\leftrightarrow$ CFG äquivalent!
- **Parsen mit CFG / Kellerautomat**
  - **Top-down Parser**
  - Bottom-up Parser (= “Shift-Reduce Parser”)

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*

<u>Anna</u> mag die Katze

Kontextfreier Top-Down Parser

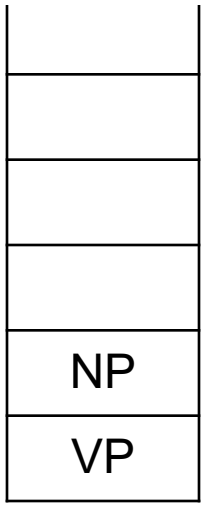
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

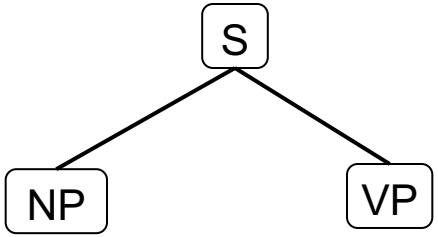
S

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*

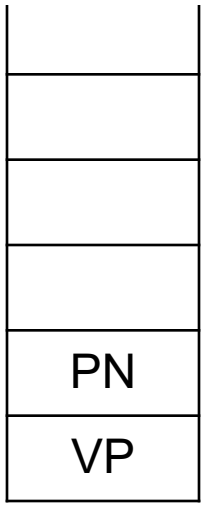


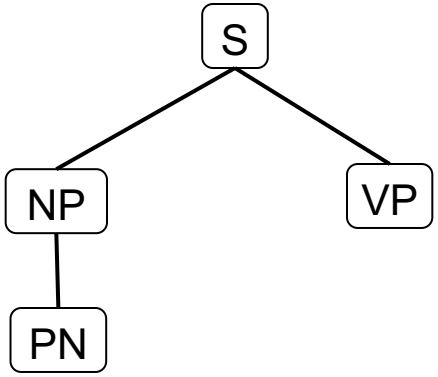


Anna mag die Katze

Kontextfreier Top-Down Parser

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

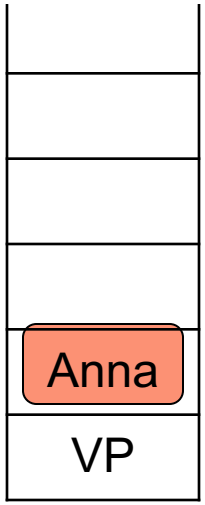


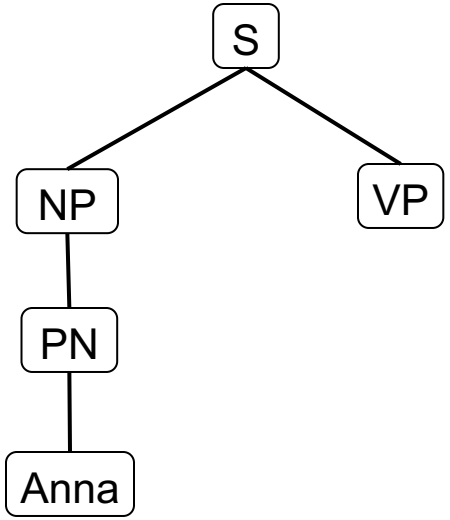


<u>Anna</u> mag die Katze

Kontextfreier Top-Down Parser

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

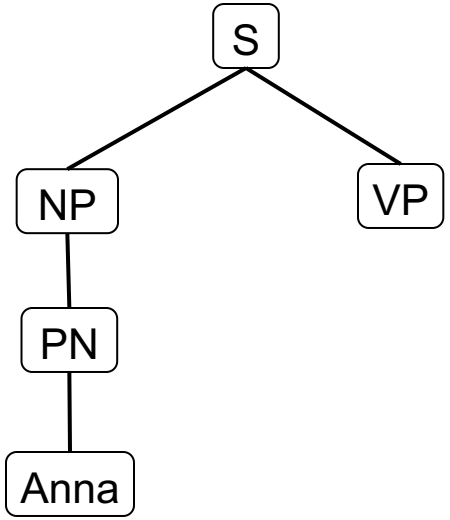




Anna mag die Katze

Kontextfreier Top-Down Parser

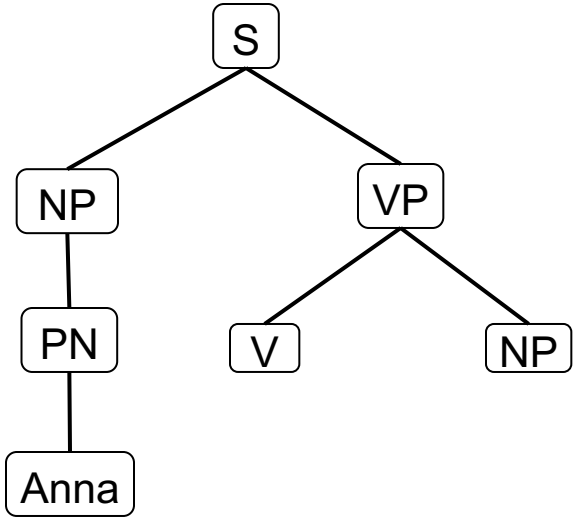
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

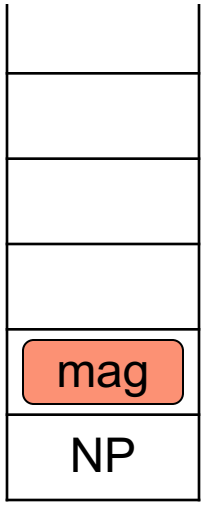
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

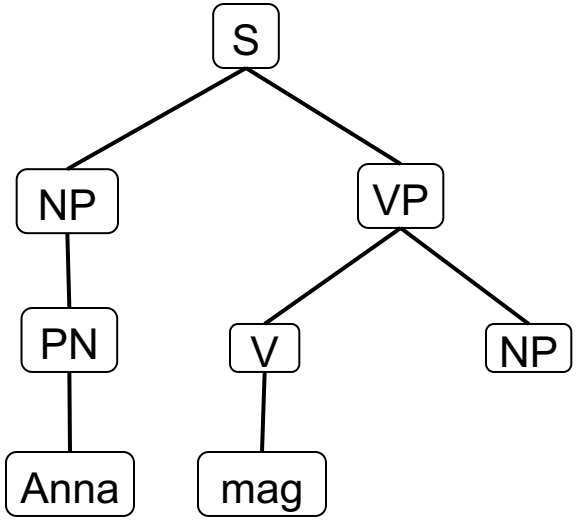


Anna mag die Katze

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

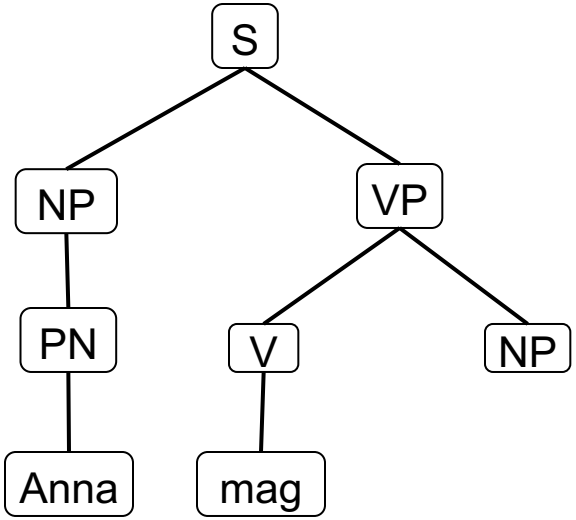
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

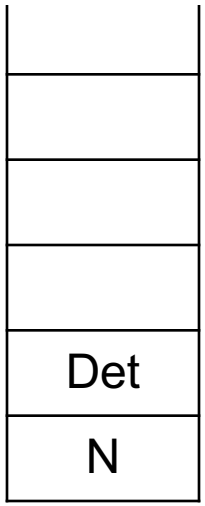
PN → *Anna*

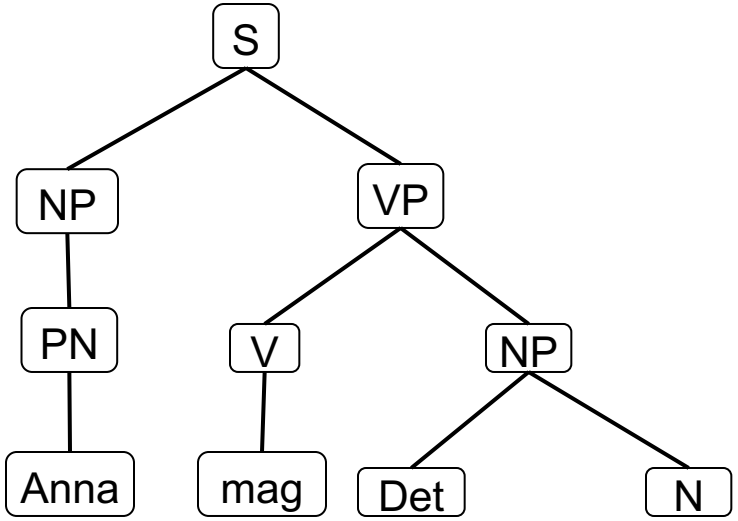


Anna mag die Katze

Kontextfreier Top-Down Parser

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

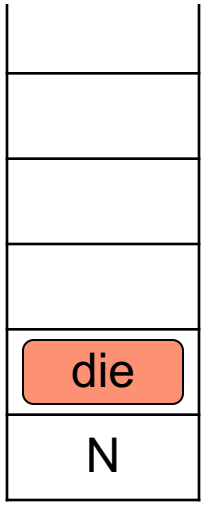


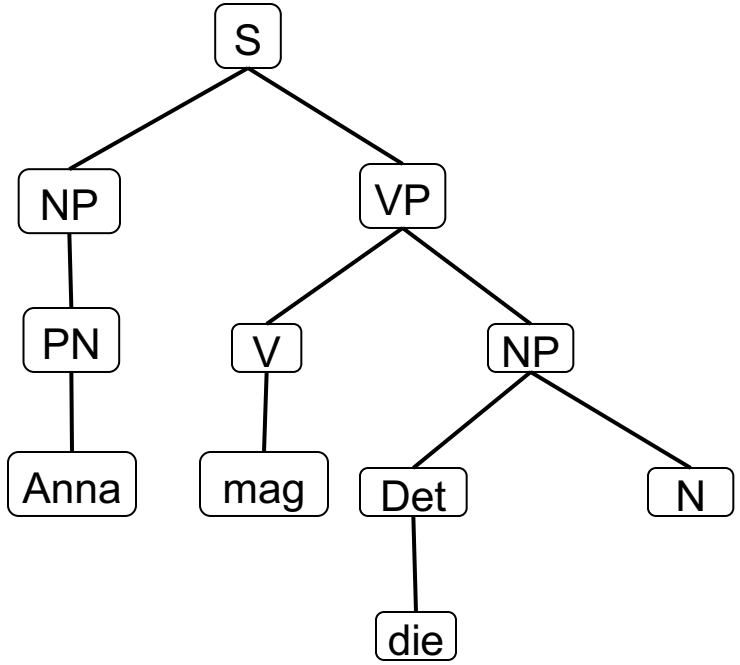


Anna mag <u>die</u> Katze

Kontextfreier Top-Down Parser

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

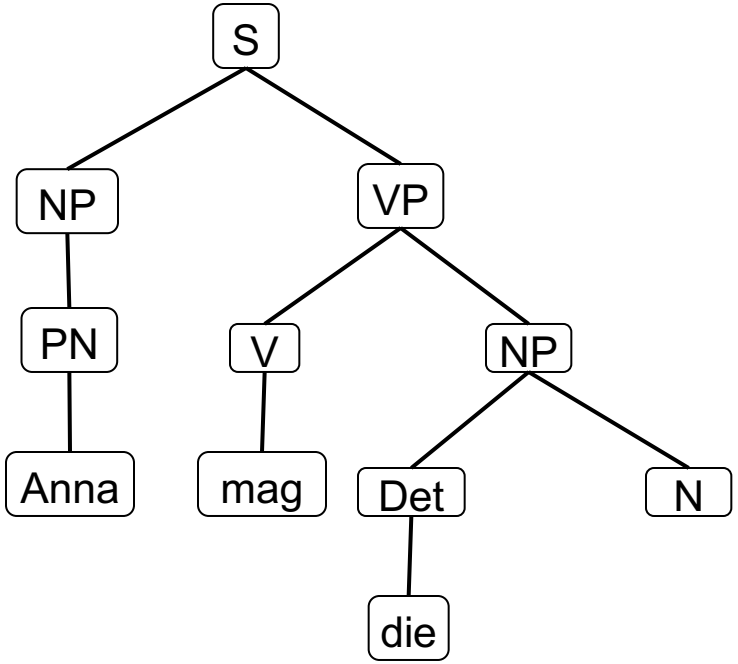




Anna mag die Katze

Kontextfreier Top-Down Parser

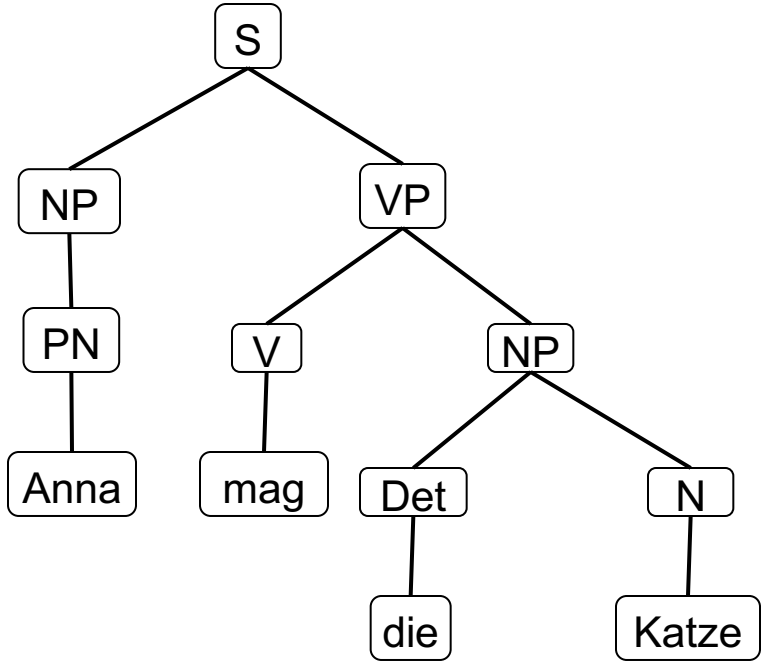
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die ***Katze***

Kontextfreier Top-Down Parser

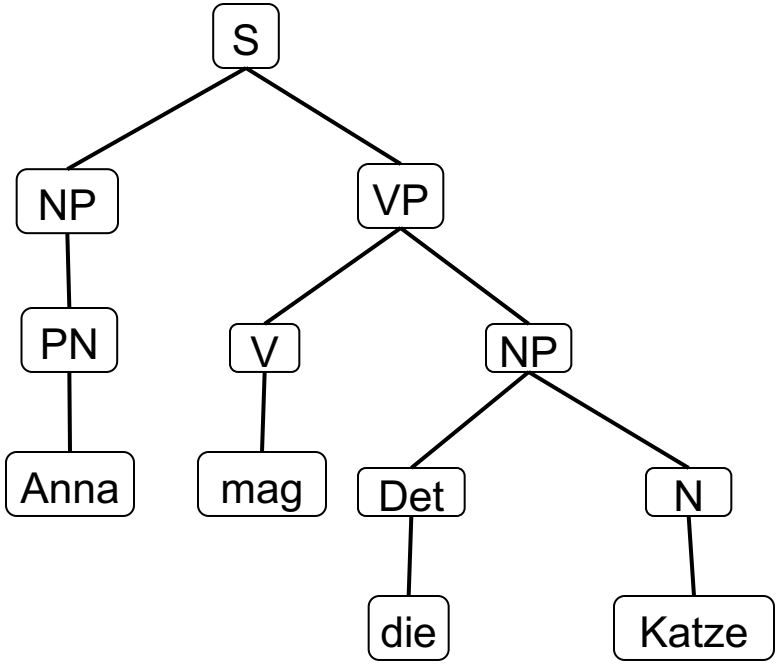
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



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*

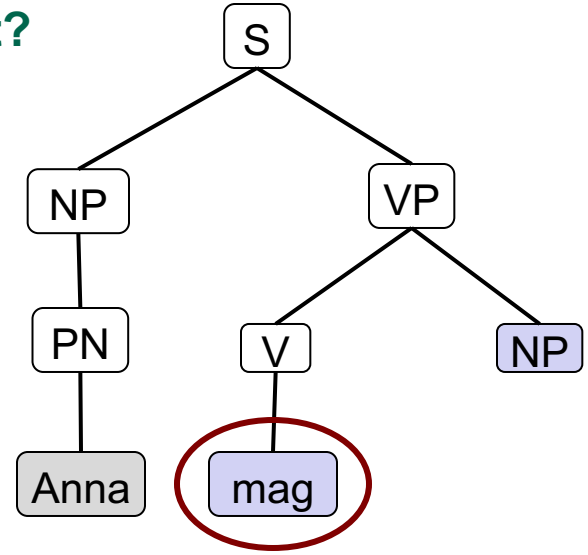


*Anna mag die Katze_*

Verbrauchte, aktive, aktueller Knoten

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

Brauchen wir den Stack überhaupt?



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

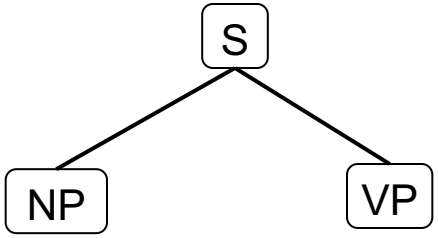
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

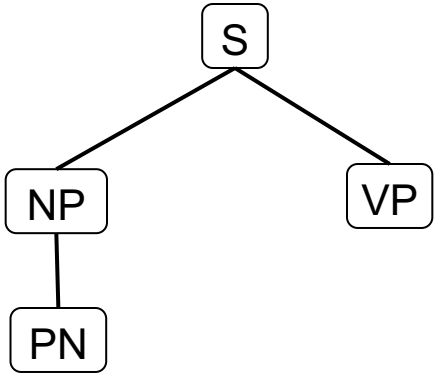
PN → *Anna*



<u>Anna</u> mag die Katze

Kontextfreier Top-Down Parser

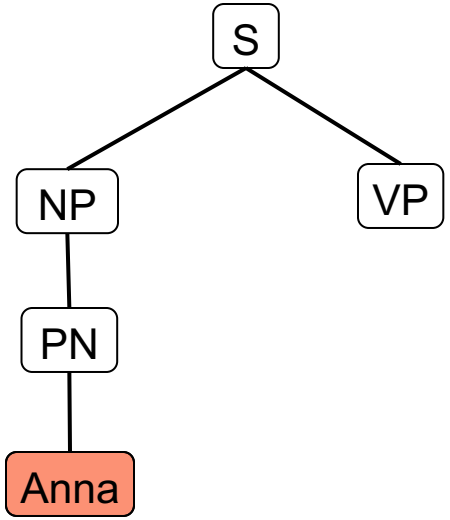
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



<u>Anna</u> mag die Katze

Kontextfreier Top-Down Parser

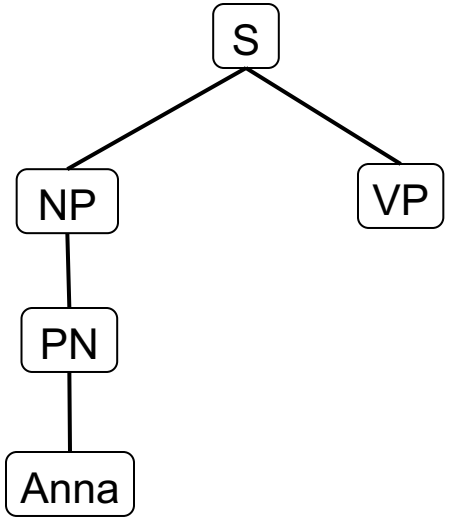
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

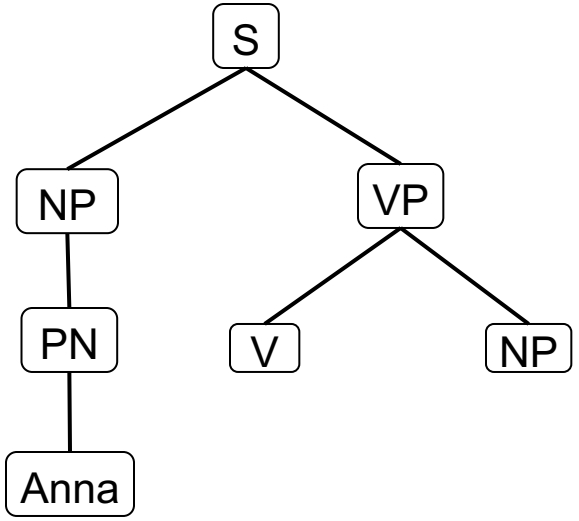
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

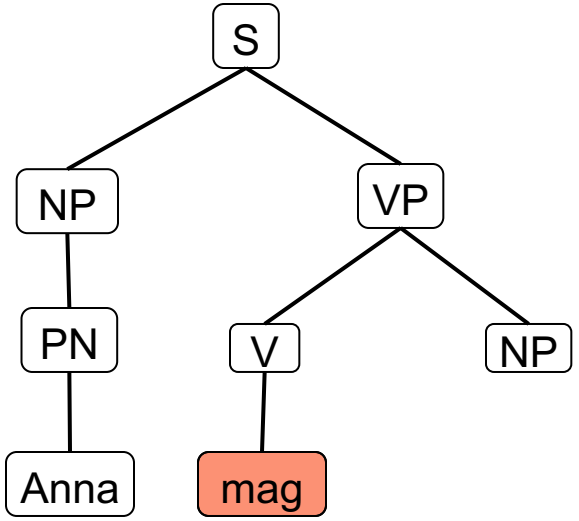
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



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

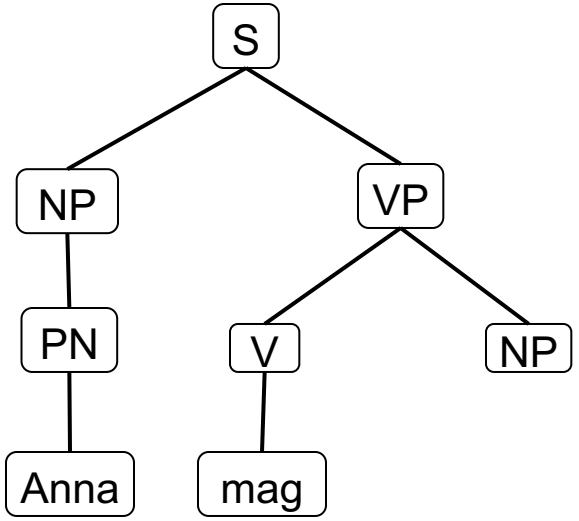
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

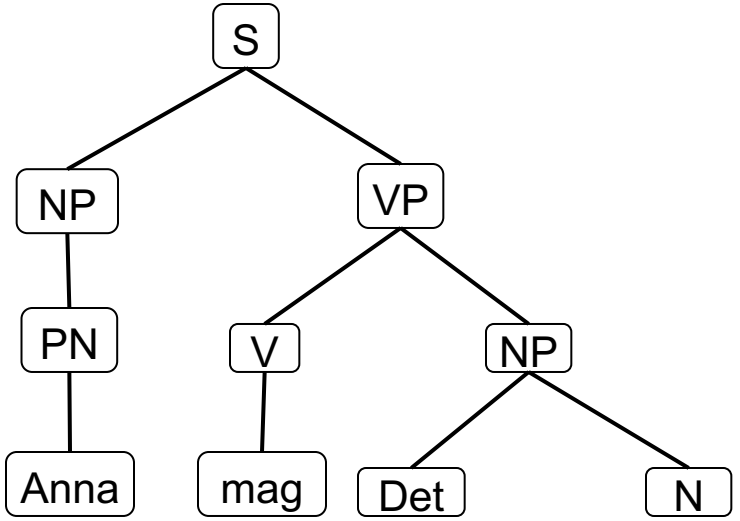
PN → *Anna*



Anna mag <u>die</u> Katze

Kontextfreier Top-Down Parser

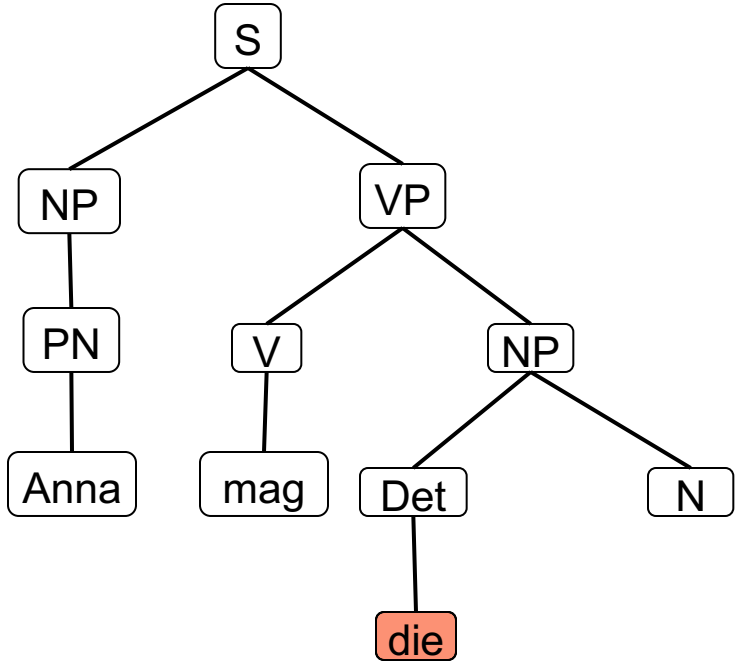
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

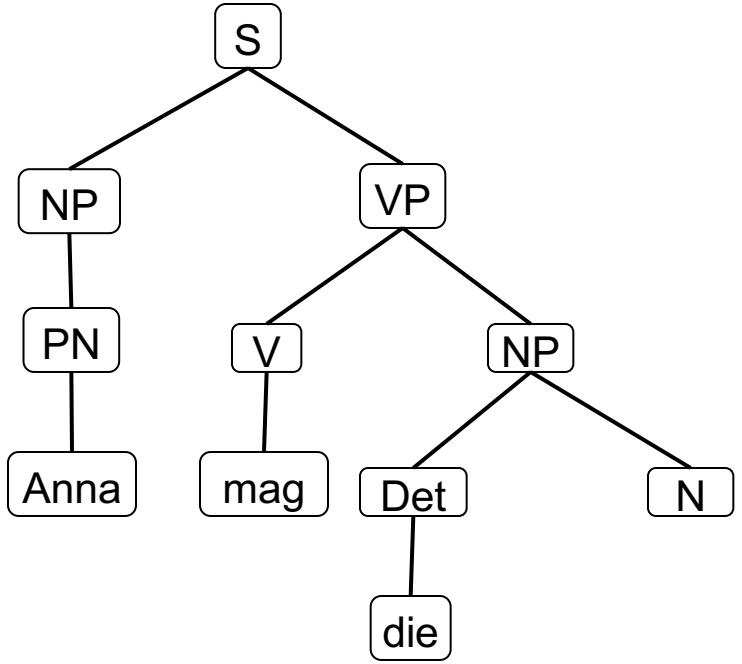
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Kontextfreier Top-Down Parser

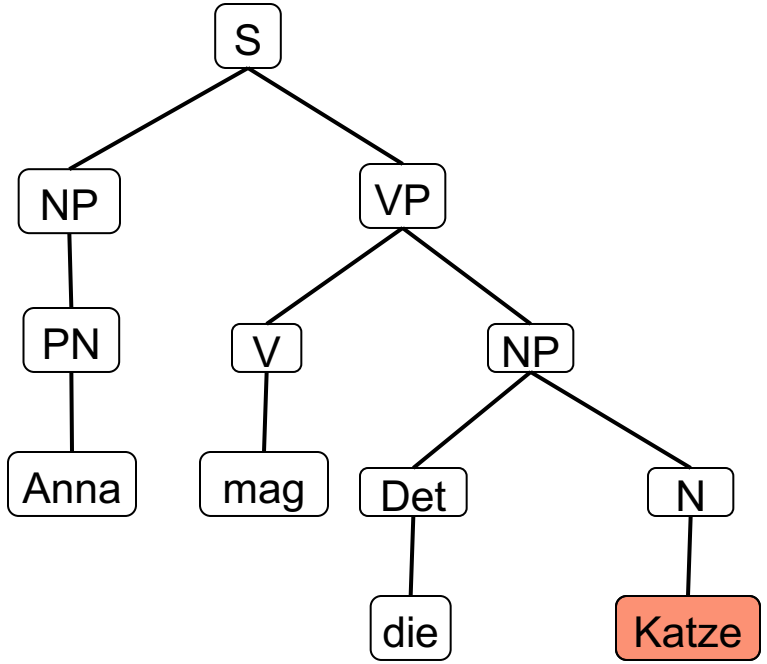
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die ***Katze***

Kontextfreier Top-Down Parser

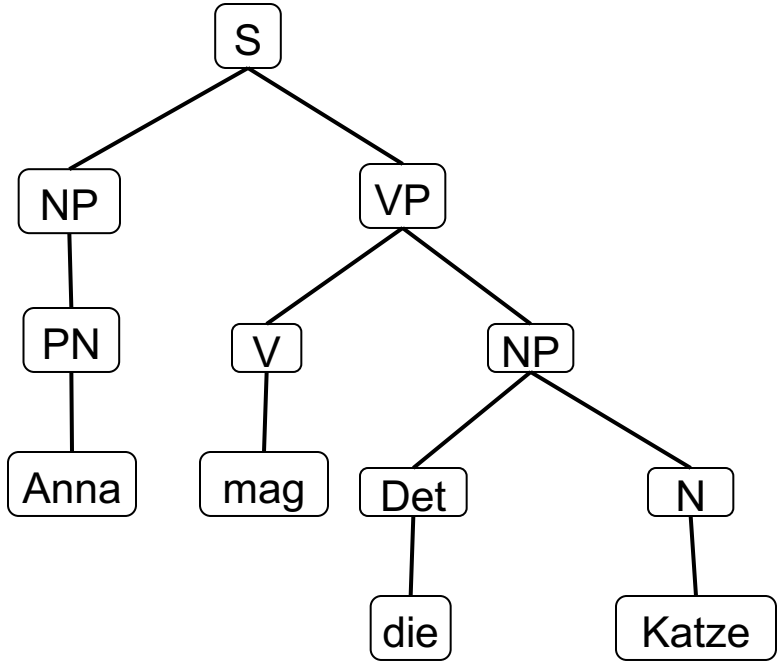
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



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

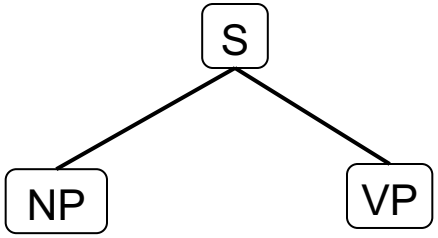
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

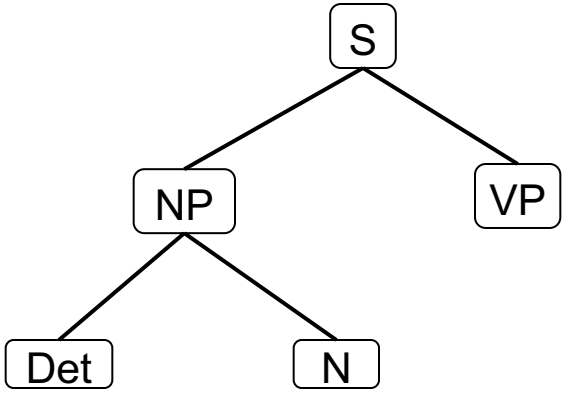
PN → *Anna*



<u>Anna</u> mag die Katze

Nicht-Determinismus in der Regelanwendung

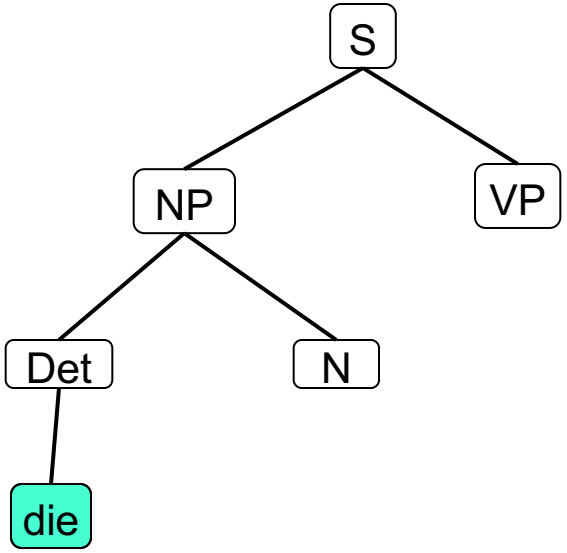
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



<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, Subjekt-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.

Kontextfreier Top-Down Parser

S → NP VP

NP → Det N

NP → PN

VP → VP PP

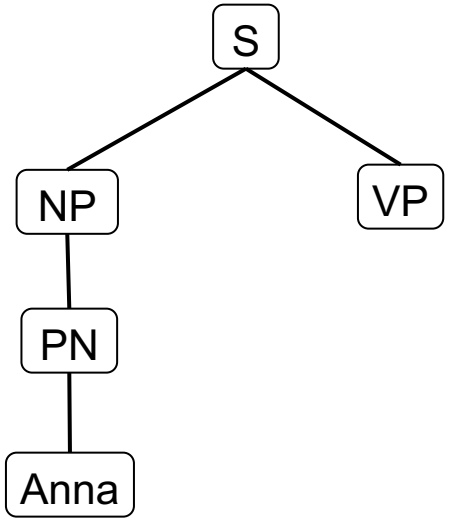
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

PN → *Anna*



Inhalte der VL

- Kellerautomat
- Kellerautomat $\leftrightarrow$ CFG: äquivalent!
- **Parsen mit CFG / Kellerautomat**
  - Top-down Parser
  - **Bottom-up Parser (= “Shift-Reduce Parser”)**

Alternative: Bottom-Up Parser

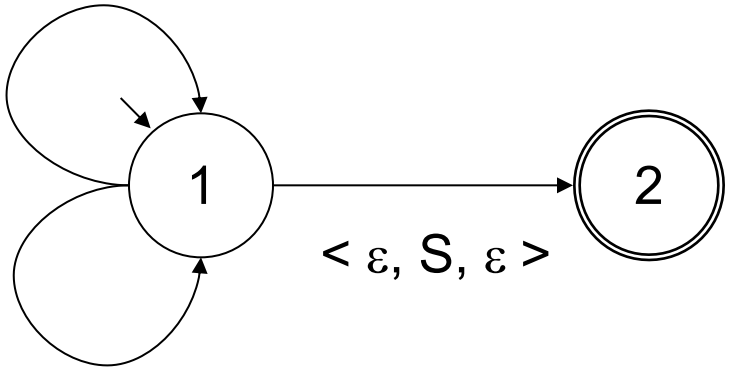
- Lies Symbole der Eingabekette und lege sie in den Stack.
- Wenn die **obersten n** Symbole im Stack umgekehrt gelesen die **rechte Seite** einer Grammatikregel **A** $\rightarrow$ **u** bilden, **ersetze** sie im Stack durch A.
- Wenn die Eingabe abgearbeitet ist und der Stack nur noch das Startsymbol 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.

<lies, lösche, schreibe>

Schema für den Shift-Reduce-Parser

u^R steht für "u rückwärts"

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



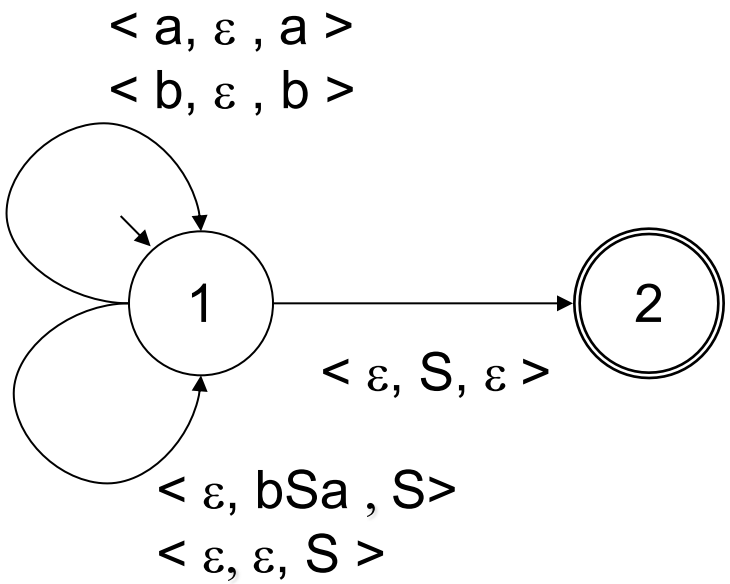
< ε, u^R, A > *(für jede Regel A → u)*

<lies, lösche, schreibe>

Beispiel: $a^n b^n$

S → aSb

S → ε

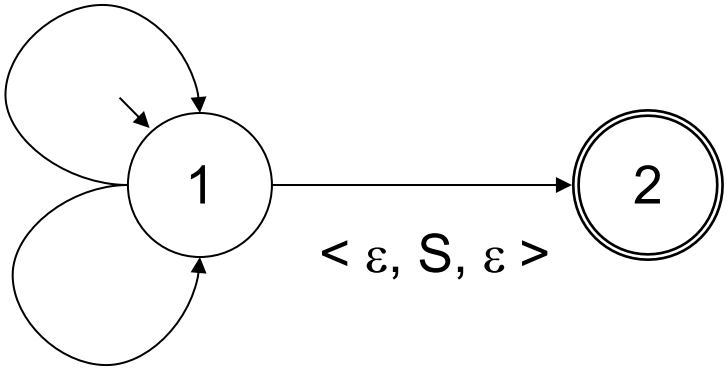


<lies, lösche, schreibe>

Shift-Reduce-Parser

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

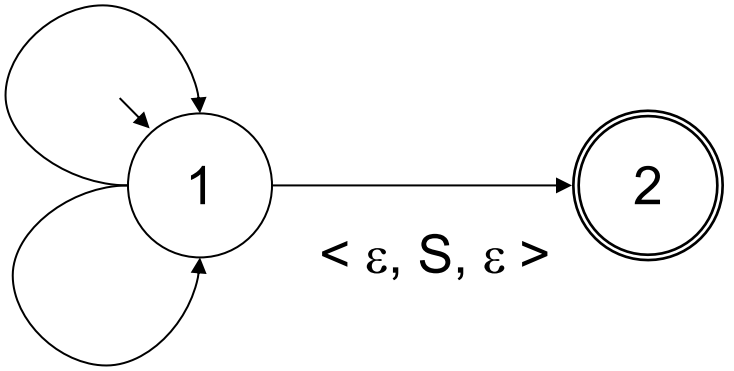
< a, ε, a > *(für jedes Terminalsymbol a)*



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

Shift-Reduce-Parser

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



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

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

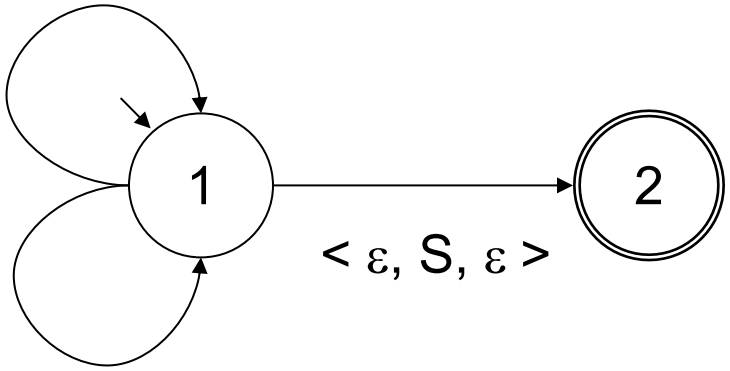
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

Anna

Anna mag die Katze

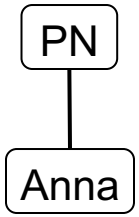
Shift-Reduce-Parser

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



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

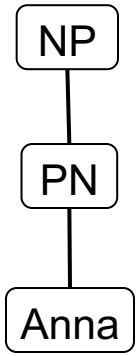
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Shift-Reduce-Parser

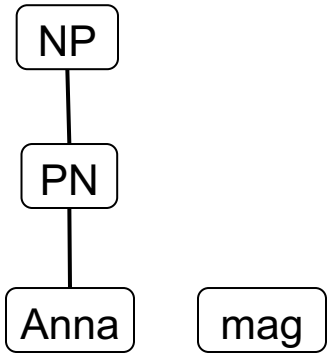
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Shift-Reduce-Parser

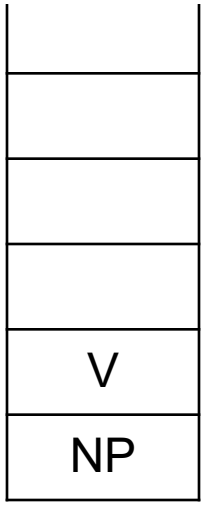
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*

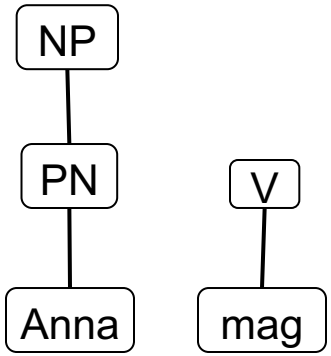


Anna mag <u>die</u> Katze

Shift-Reduce-Parser

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

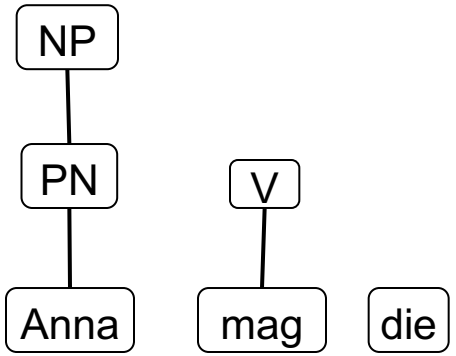




Anna mag die Katze

Shift-Reduce-Parser

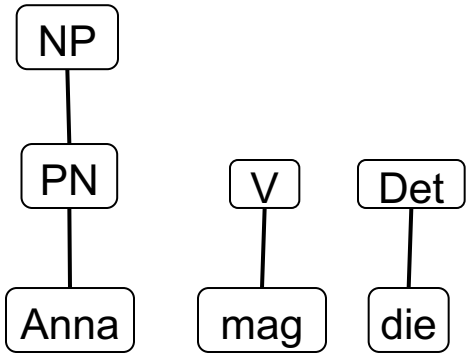
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



Anna mag die Katze

Shift-Reduce-Parser

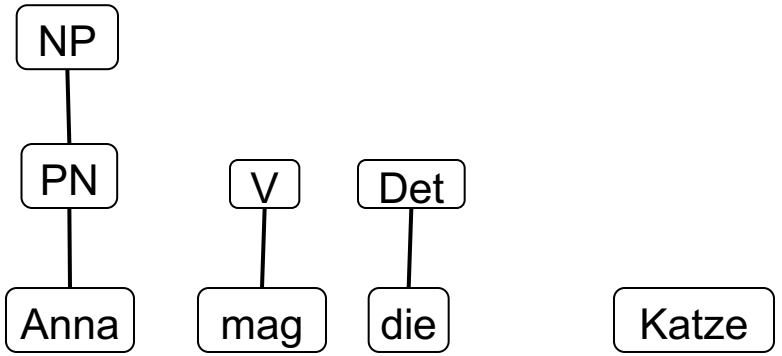
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



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

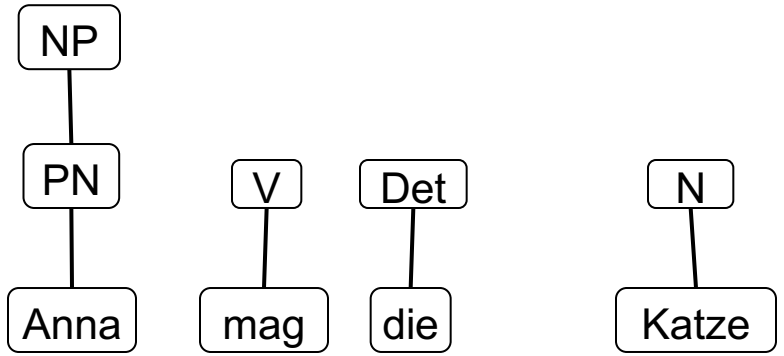
VP → V NP

Det → *die*

N → *Katze*

V → *mag*

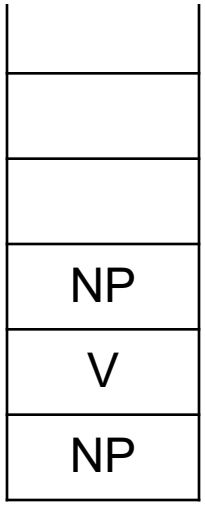
PN → *Anna*

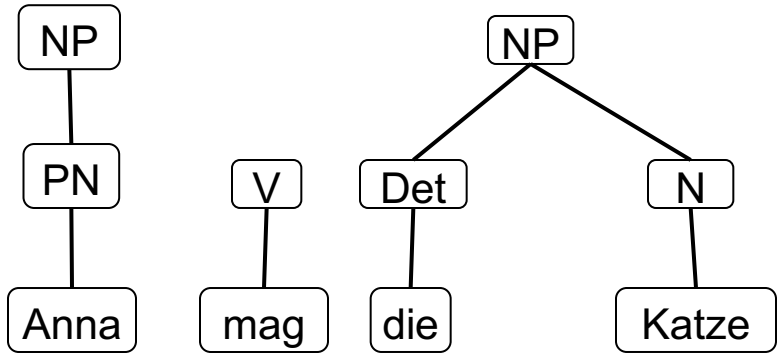


*Anna mag die Katze_*

Shift-Reduce-Parser

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

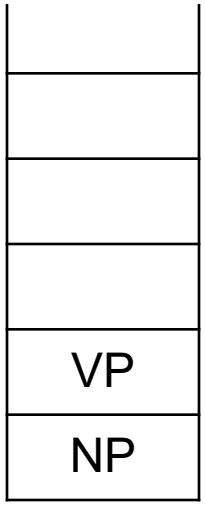


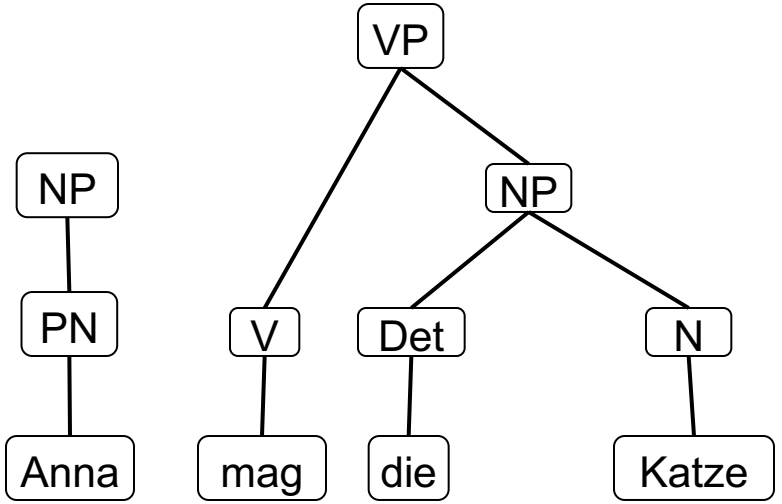


*Anna mag die Katze_*

Shift-Reduce-Parser

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

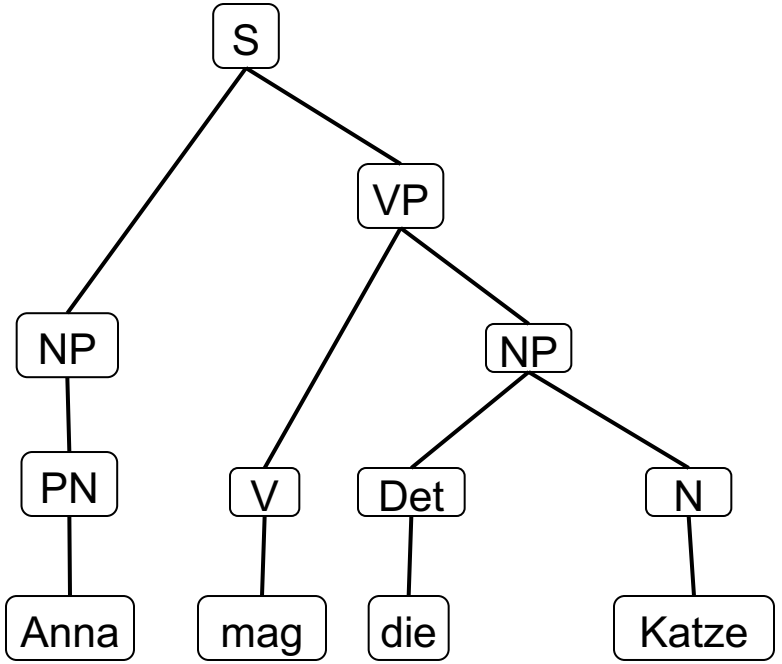




*Anna mag die Katze_*

Shift-Reduce-Parser

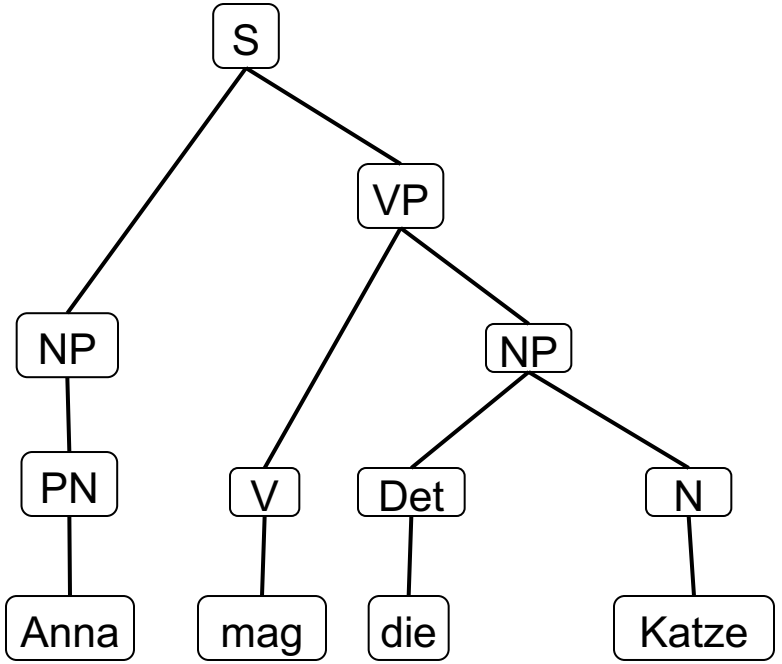
S → NP VP
NP → Det N
NP → PN
VP → V NP
Det → *die*
N → *Katze*
V → *mag*
PN → *Anna*



*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.
  - (w^R steht für „die Symbolfolge w rückwärts“)

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 darstellen).

- 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.

Was Sie nach dieser Vorlesung können sollten

- Was ist ein Kellerautomat, wie unterscheidet er sich von einem einfachen Automaten?
- Wie verhalten sich CFG und Kellerautomat zueinander?
- Grammatik auf einen Kellerautomaten abbilden können
- Parsen mit CFG / Kellerautomat
  - Top-down Parser
  - Bottom-up Parser (= “Shift-Reduce Parser”)