Prolog

7. Kapitel: Definite Clause Grammar (DCG)

Dozentin: Wiebke Petersen

Kursgrundlage: Learn Prolog Now (Blackburn, Bos, Striegnitz)

Zusammenfassung Kapitel 6

- Wir haben die zentralen Listenprädikate `append/3`, `delete/3` und `reverse/2` kennengelernt.
- Wir haben gelernt, Akkumulatoren für Listenelemente zu verwenden, um effizienter Prädikate zu definieren.
- Wir haben Differenzlisten eingeführt, die die Konkatenation von Listen in einem Schritt ermöglichen.

- **Keywords**: `append/3`, `delete/3`, `reverse/2`, Akkumulatorlisten, Differenzlisten.
- **Wichtig**: Rekursive Prädikatsaufrufe sind ineffizient und sollten auf ein Minimum beschränkt werden
  (vgl. `naiverev/2` mit `reverse/2` und `append/3` mit `append_dl/3`).
- **Ausblick Kapitel 7**: Definite Clause Grammars (DCG's)

Sprachbeschreibung durch Angabe einer Grammatik

Definition (formale Grammatik)

*Eine **formale Grammatik** besteht aus*

- *einem Alphabet von Terminalsymbolen Σ (den Symbolen der beschriebenen Sprache)*
- *einer Menge von Nichtterminalsymbolen N (unseren Hilfssymbolen)*
- *einem Startsymbol $S \in N$*
- *einer Menge von Regeln der Form $\alpha \rightarrow \beta$, wobei α und β Ketten von Symbolen aus $\Sigma \cup N$ sind.*

- Grammatiken sind endliche Regelsysteme.
- Die Menge aller Ketten, die von einer Grammatik generiert werden, bilden die von der Grammatik beschriebene formale Sprache (das sind alle Ketten von Terminalsymbolen, die man aus dem Startsymbol durch sukzessive Regelanwendung gewinnen kann).

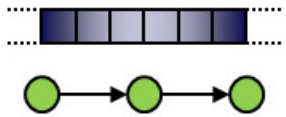
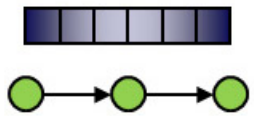
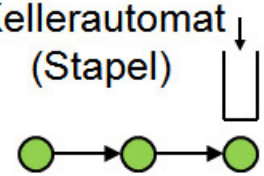
kontextfreie Grammatiken

Eine formale Grammatik in der jede linke Regelseite aus genau einem Nichtterminalsymbol besteht heißt **kontextfrei**.

Beispiel:
$G = (\{\text{S,NP,VP,N,V,Det}\}, \{\text{eine, die, keine, katze, maus, jagt klaut}\}, \text{S}, P)$

$$P = \left\{ \begin{array}{lllllllll} \text{S} & \rightarrow & \text{NP VP} & \text{VP} & \rightarrow & \text{V NP} & \text{NP} & \rightarrow & \text{Det N} \\ \text{Det} & \rightarrow & \text{eine} & \text{N} & \rightarrow & \text{katze} & \text{V} & \rightarrow & \text{jagt} \\ \text{Det} & \rightarrow & \text{die} & \text{N} & \rightarrow & \text{maus} & \text{V} & \rightarrow & \text{klaut} \\ \text{Det} & \rightarrow & \text{keine} & & & & & & \end{array} \right\}$$

Chomskyhierarchie & Automaten (grober Überblick)

Sprache	*Automat*	*Grammatik*	*Erkennung*	*Abhängigkeit*
rekursiv aufzählbar	Turing Maschine 	unbeschränkt Baa → ε	unentscheidbar	beliebig
kontext-sensitiv	linear gebunden 	kontext-sensitiv γAδ → γβδ	NP-vollständig 	überkreuzt 
kontext-frei	Kellerautomat (Stapel) 	kontextfrei C → bABa	polynomiell 	eingebettet 
regulär	endlicher Automat 	regulär A → bA	linear 	strikt lokal 

Grammatik 1 ohne Listen

```
% Grammatikregel:
s(W1,W2,W3,W4,W5):-
     det(W1),
     n(W2),
     v(W3),
     det(W4),
     n(W5).

% Lexikon:
det(eine).
det(die).
det(keine).
n(maus).
n(katze).
v(jagt).
v(klaut).
```

Grammatik 1 ohne Listen

```
% Grammatikregel:
s(W1,W2,W3,W4,W5):-
     det(W1),
     n(W2),
     v(W3),
     det(W4),
     n(W5).

% Lexikon:
det(eine).
det(die).
det(keine).
n(maus).
n(katze).
v(jagt).
v(klaut).
```

Vorteile

- effiziente Implementierung

Nachteile

- Es werden keinerlei Generalisierungen erfasst.
- Pro Satzlänge wird eine eigenes Satzprädikat `s(W1,...,Wn)` benötigt.
- Rekursive Strukturen können nicht erfasst werden (z.B. „Die alte, schöne, hungrige, ..., verrückte, gemeine katze klaut die maus'").

Grammatik 2 mit Listen und append/3 am Ende

```
% Grammatikregeln,
% append/3 am Ende:
s(L3):-
     np(L1),
     vp(L2),
     append(L1,L2,L3).
np(L3):-
     det(L1),
     n(L2),
     append(L1,L2,L3).
vp(L3):-
     v(L1),
     np(L2),
     append(L1,L2,L3).

% Lexikon:
det([eine]).
det([die]).
det([keine]).
n([maus]).
n([katze]).
v([jagt]).
v([klaut]).
```

Grammatik 2 mit Listen und append/3 am Ende

```
% Grammatikregeln,
% append/3 am Ende:
s(L3):-
     np(L1),
     vp(L2),
     append(L1,L2,L3).
np(L3):-
     det(L1),
     n(L2),
     append(L1,L2,L3).
vp(L3):-
     v(L1),
     np(L2),
     append(L1,L2,L3).

% Lexikon:
det([eine]).
det([die]).
det([keine]).
n([maus]).
n([katze]).
v([jagt]).
v([klaut]).
```

Vorteile

- Grammatische Generalisierungen werden erfasst
- Direkte Übertragung einer kontextfreien Grammatik in dieses Format ist möglich

Nachteile

- Sehr ineffizient:
  - es werden zunächst beliebige NPs und VPs generiert,
  - anschließend werden sie konkateniert,
  - dann wird geprüft, ob sie gleich der zu überprüfenden Kette sind.

Grammatik 3 mit Listen und append/3 am Anfang

```
% Grammatikregeln,
% append/3 am Anfang:
s(L3):-
     append(L1,L2,L3),
     np(L1),
     vp(L2).
np(L3):-
     append(L1,L2,L3),
     det(L1),
     n(L2).
vp(L3):-
     append(L1,L2,L3),
     v(L1),
     np(L2).

% Lexikon:
det([eine]).
det([die]).
det([keine]).
n([maus]).
n([katze]).
v([jagt]).
v([klaut]).
```

Grammatik 3 mit Listen und append/3 am Anfang

```
% Grammatikregeln,
% append/3 am Anfang:
s(L3):-
     append(L1,L2,L3),
     np(L1),
     vp(L2).
np(L3):-
     append(L1,L2,L3),
     det(L1),
     n(L2).
vp(L3):-
     append(L1,L2,L3),
     v(L1),
     np(L2).

% Lexikon:
det([eine]).
det([die]).
det([keine]).
n([maus]).
n([katze]).
v([jagt]).
v([klaut]).
```

Vorteile

- Grammatische Generalisierungen werden erfasst
- Direkte Übertragung einer kontextfreien Grammatik in dieses Format ist möglich

Nachteile

- Nicht zur Generierung geeignet. Loopt sobald beim backtracking auf append/3 in der np-Klausel, die die NP innerhalb der VP generiert, das erste Mal eine Präfixliste generiert wird, die länger ist als 1.
- Immer noch sehr ineffizient:
  - es werden zunächst beliebige Zerlegungen des Satzes generiert,
  - anschließend wird geprüft, ob die erste Teilkette eine NP und die zweite eine VP bildet

Grammatik 4 mit Differenzlisten

```
% Grammatikregeln:
s(A,C):-
     np(A,B),
     vp(B,C).
np(A,C):-
     det(A,B),
     n(B,C).
vp(A,C):-
     v(A,B),
     np(B,C).

% Lexikon:
det([eine|A],A).
det([die|A],A).
det([keine|A],A).
n([maus|A],A).
n([katze|A],A).
v([jagt|A],A).
v([klaut|A],A).
```

Grammatik 4 mit Differenzlisten

```
% Grammatikregeln:
s(A,C):-
     np(A,B),
     vp(B,C).
np(A,C):-
     det(A,B),
     n(B,C).
vp(A,C):-
     v(A,B),
     np(B,C).

% Lexikon:
det([eine|A],A).
det([die|A],A).
det([keine|A],A).
n([maus|A],A).
n([katze|A],A).
v([jagt|A],A).
v([klaut|A],A).
```

▸ Übung

Vorteile

- Grammatische Generalisierungen werden erfasst
- Direkte Übertragung einer kontextfreien Grammatik in dieses Format ist möglich
- Sehr effizient

Nachteile

- Differenzlisten sind schwierig zu lesen.
- Die Verwendung der vielen Variablen ist unübersichtlich und führt leicht zu Fehlern.

Definite Clause Grammar (DCG)

```
% Grammatikregeln:
s --> np, vp.
np --> det, n.
vp --> v, np.

% Lexikon:
det --> [eine].
det --> [die].
det --> [keine].
n --> [maus].
n --> [katze].
v --> [jagt].
v --> [klaut].
```

- Prolog ermöglicht die kompakte Darstellung von Phrasenstrukturgrammatiken als Definite Clause Grammars.
- DCG-Klauseln werden intern umgewandelt in Klauseln mit Differenzlisten (siehe vorherige Grammatik 4).
- Bei der DCG-Darstellung handelt es sich nur um "notational sugar".
  `?- listing(s/2).`

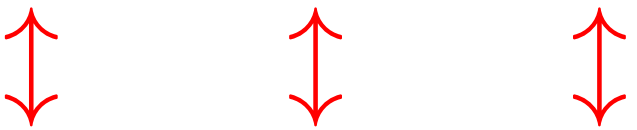
Definite Clause Grammar (DCG)

- Definite Clause Grammars in Prolog entsprechen in ihrer Form den kontextfreien Phrasenstrukturregeln.
- DCGs werden beim Laden (*consult*) eines Prolog- Programms mit Differenzlisten annotiert.

kontextfreie Phrasen-strukturgrammatik	DCG (extern)	DCG (intern)
S → NP VP	`s --> np, vp.`	`s(A,C):- np(A,B), vp(B,C).`
NP → Det N	`np --> det, n.`	`np(A,C):- det(A,B), n(B,C).`
VP → V NP	`vp --> v, np.`	`vp(A,C):- v(A,B), np(B,C).`
Det → eine	`det --> [eine].`	`det([eine|A],A).`
N → katze	`n --> [katze].`	`n([katze|A],A).`
V → jagt	`v --> [jagt].`	`v([jagt|A],A).`
...	...	...

Vergleiche DCGs mit append_dl/3

```
                                  s   (A,C)   :-
        np(A,B), vp(B,C).
```



```
append_dl((A,B),    (B,C),    (A,C)   ).
```

Präfix Suffix Konkatenation

DCG zu Klauseln

Regeln:

```
p1 --> p2,...,pn.
```

```
p1(V1,Vn) :- p2(V1,V2),...,pn(Vn-1,Vn).
```

DCG zu Klauseln

Regeln:

```
p1 --> p2,...,pn.
```

```
p1(V1,Vn) :- p2(V1,V2),...,pn(Vn-1,Vn).
```

Lexikoneinträge:

```
p --> [Atom].
```

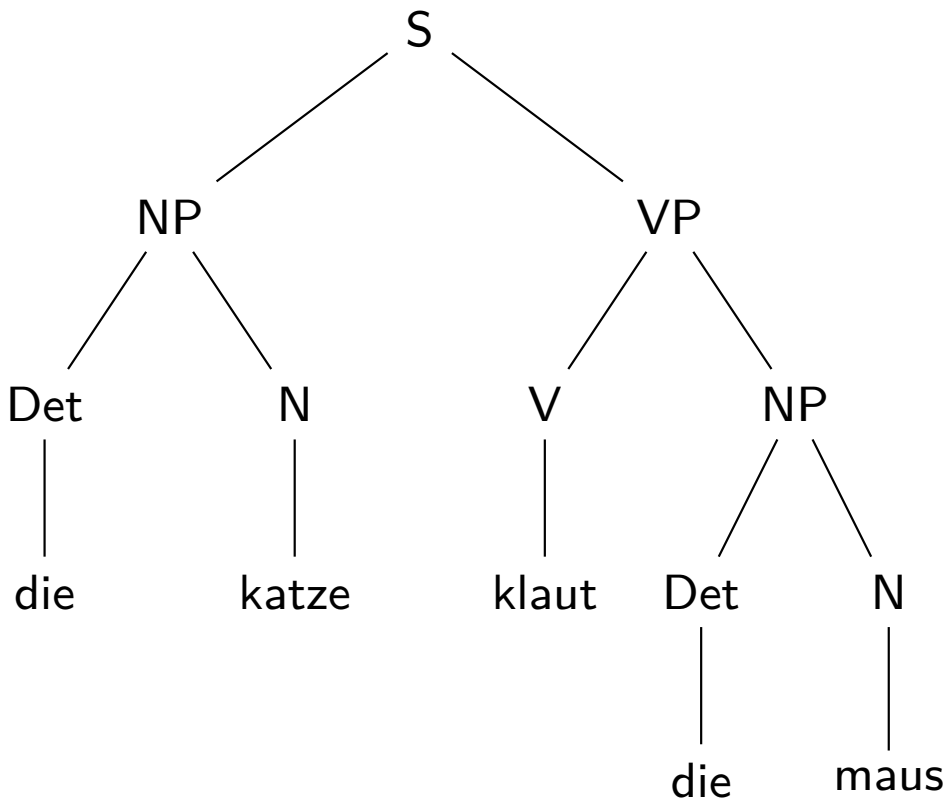
```
p([Atom|T],T).
```

▸ Übung

Prologs Parsingstrategie

Linksableitung:

Ableitungsbaum:



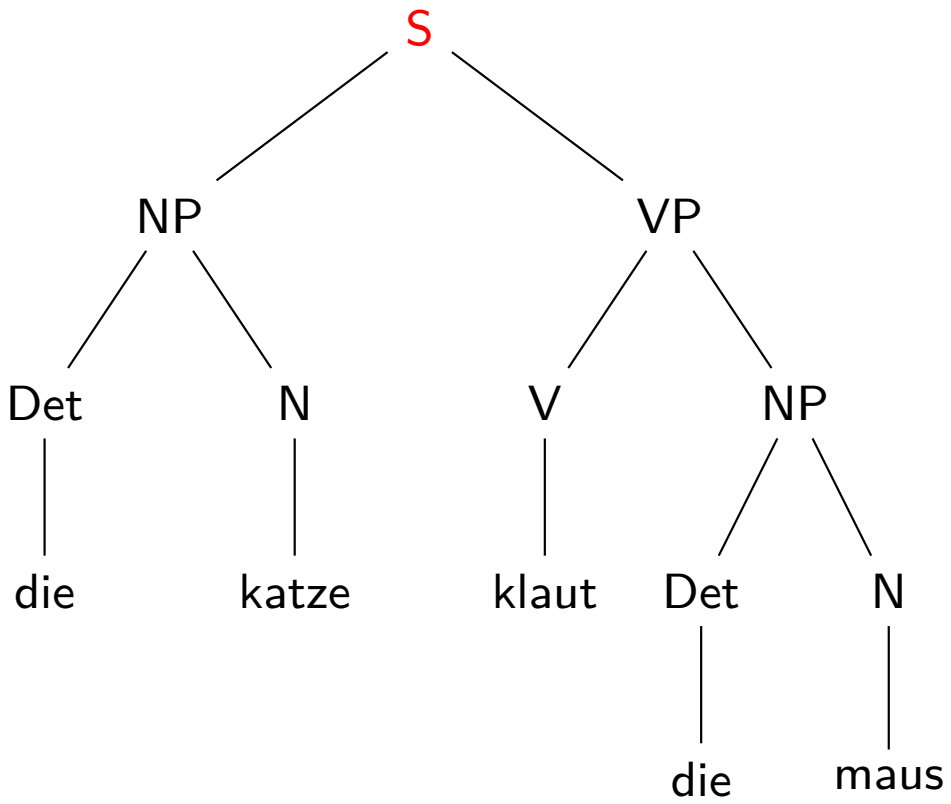
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S

Ableitungsbaum:



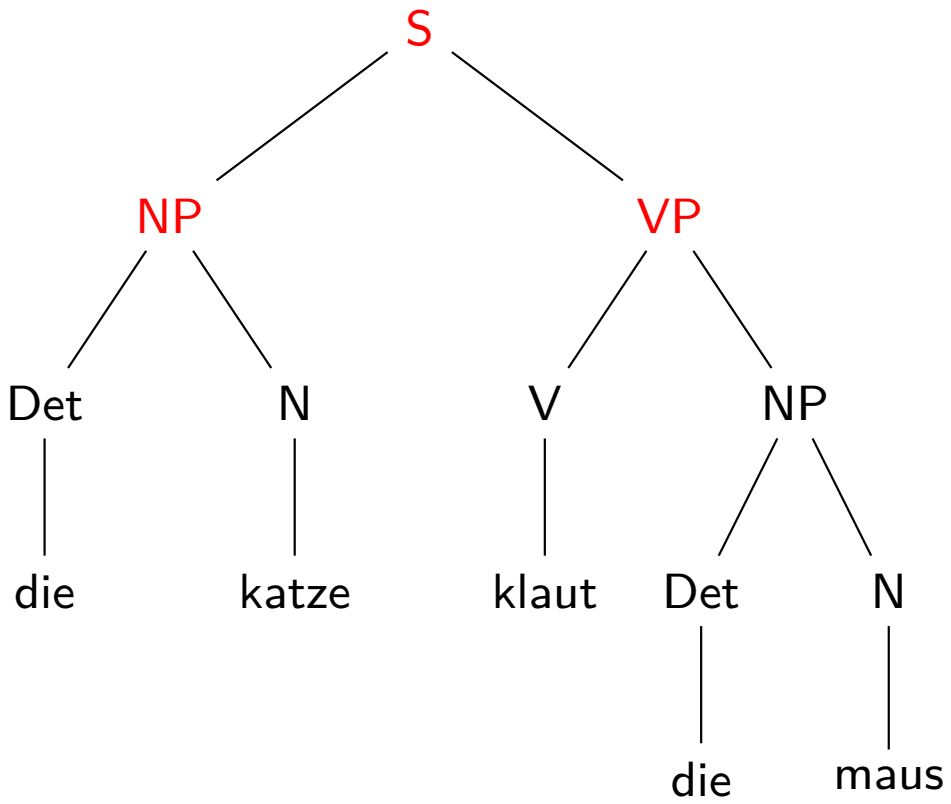
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
⊢ NP VP

Ableitungsbaum:



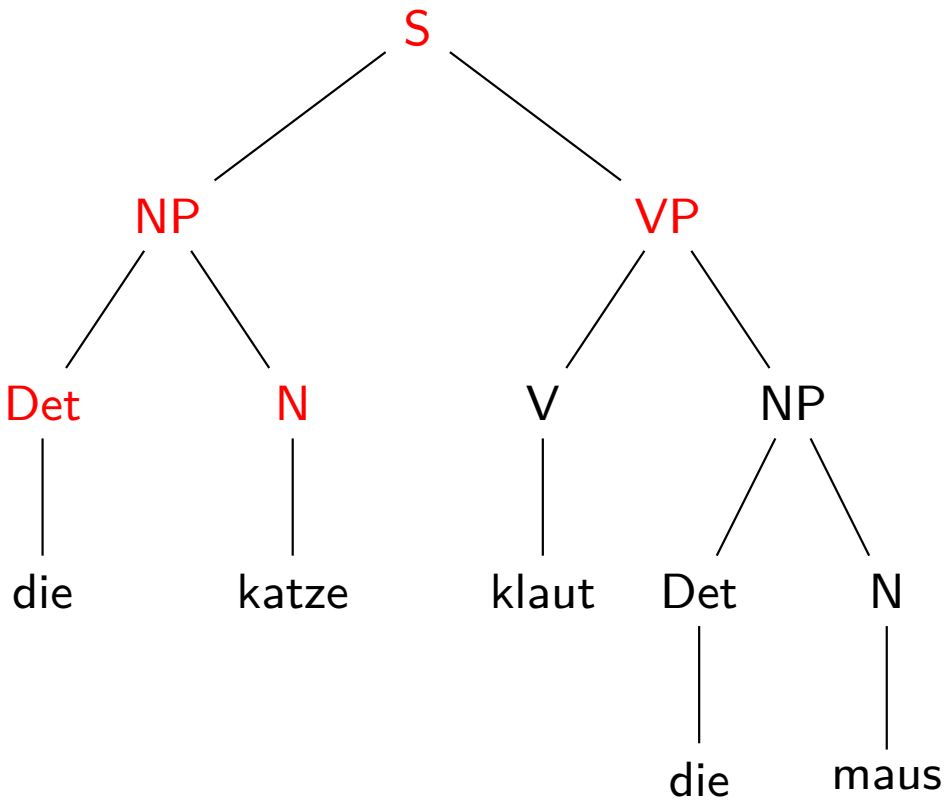
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP

Ableitungsbaum:



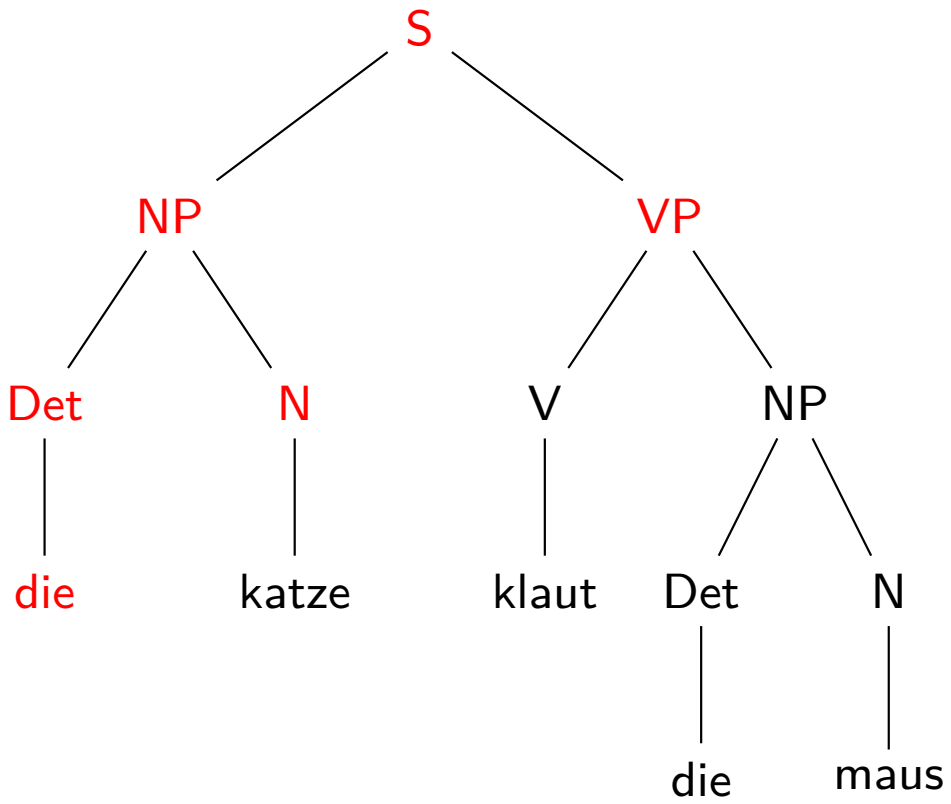
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP

Ableitungsbaum:



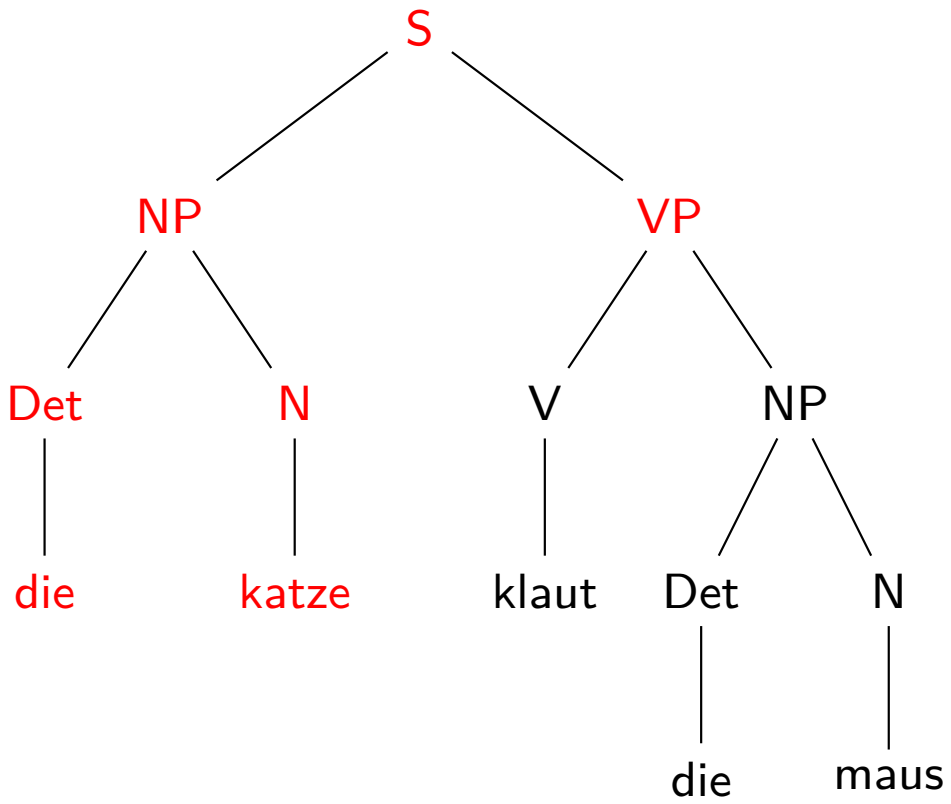
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP
- ⊢ die katze VP

Ableitungsbaum:



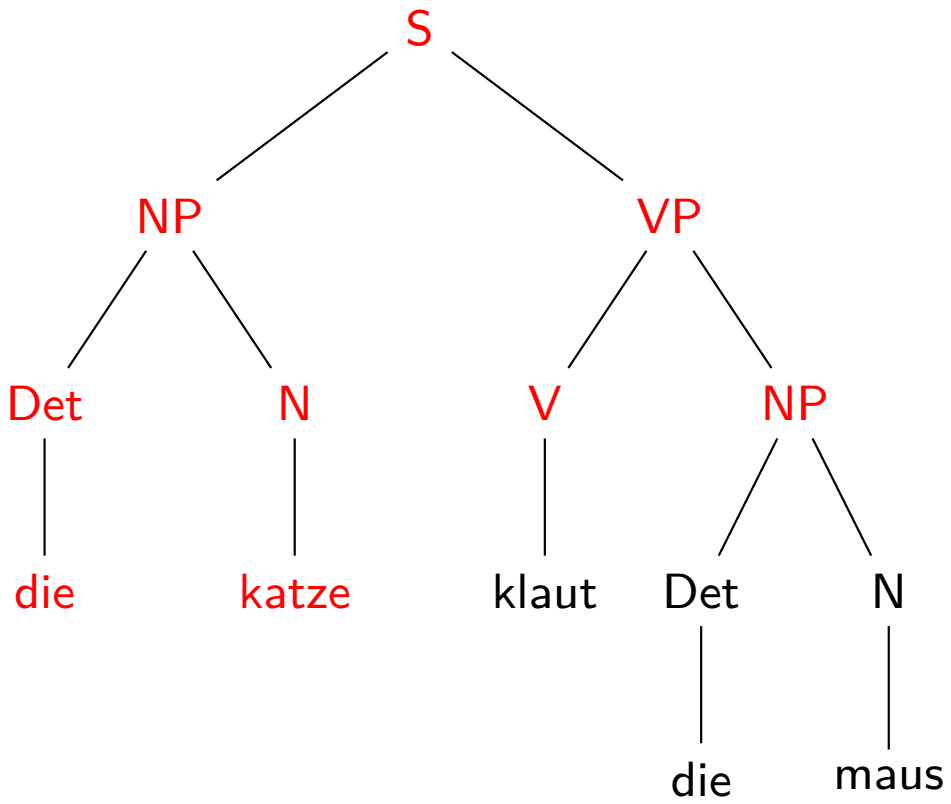
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP
- ⊢ die katze VP
- ⊢ die katze V NP

Ableitungsbaum:



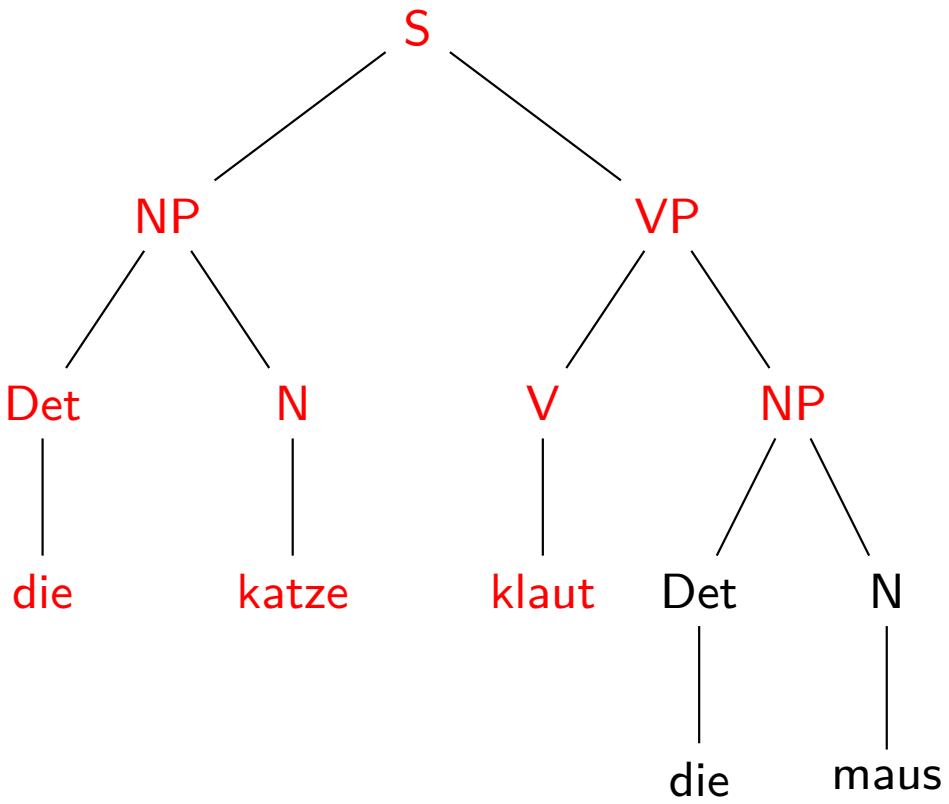
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP
- ⊢ die katze VP
- ⊢ die katze V NP
- ⊢ die katze klaut NP

Ableitungsbaum:



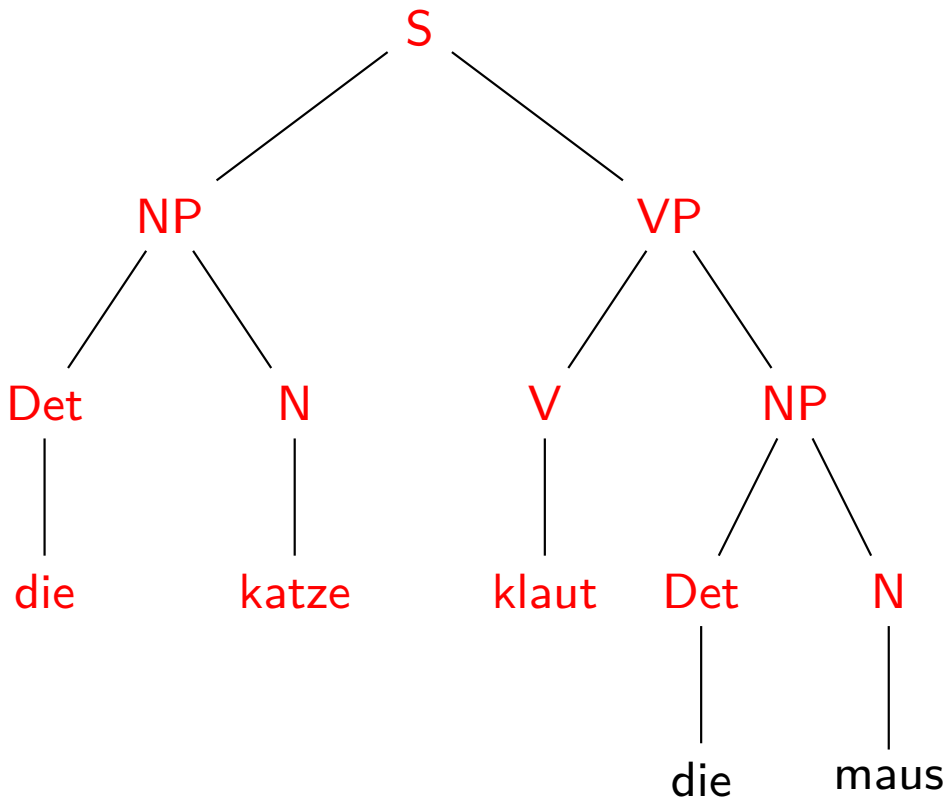
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP
- ⊢ die katze VP
- ⊢ die katze V NP
- ⊢ die katze klaut NP
- ⊢ die katze klaut Det N

Ableitungsbaum:



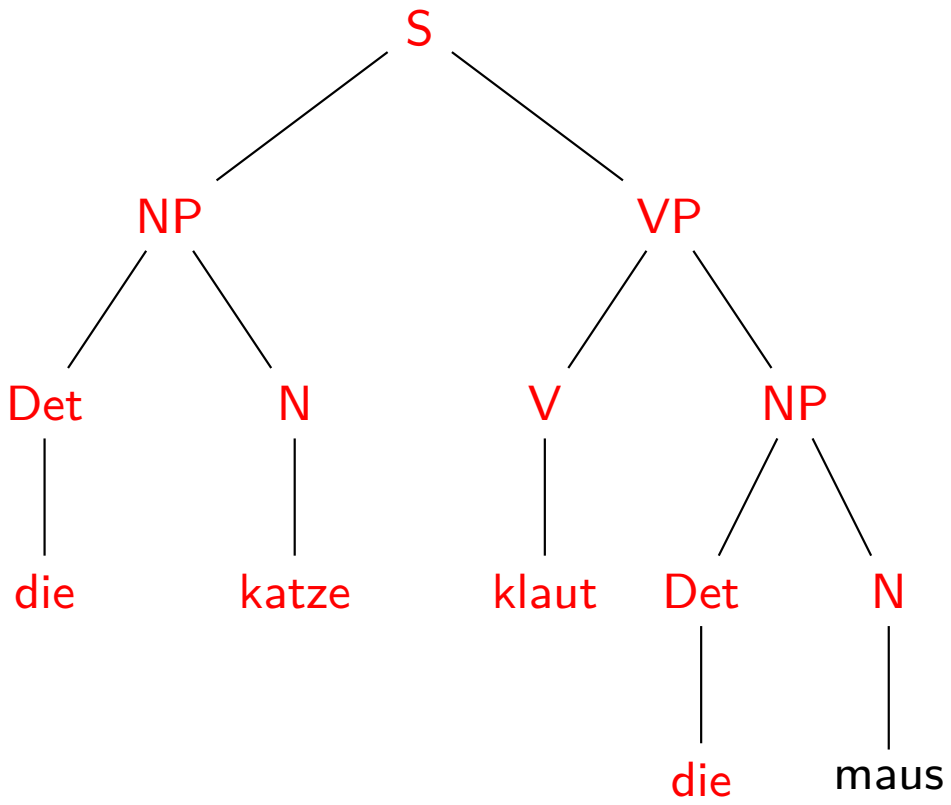
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP
- ⊢ die katze VP
- ⊢ die katze V NP
- ⊢ die katze klaut NP
- ⊢ die katze klaut Det N
- ⊢ die katze klaut die N

Ableitungsbaum:



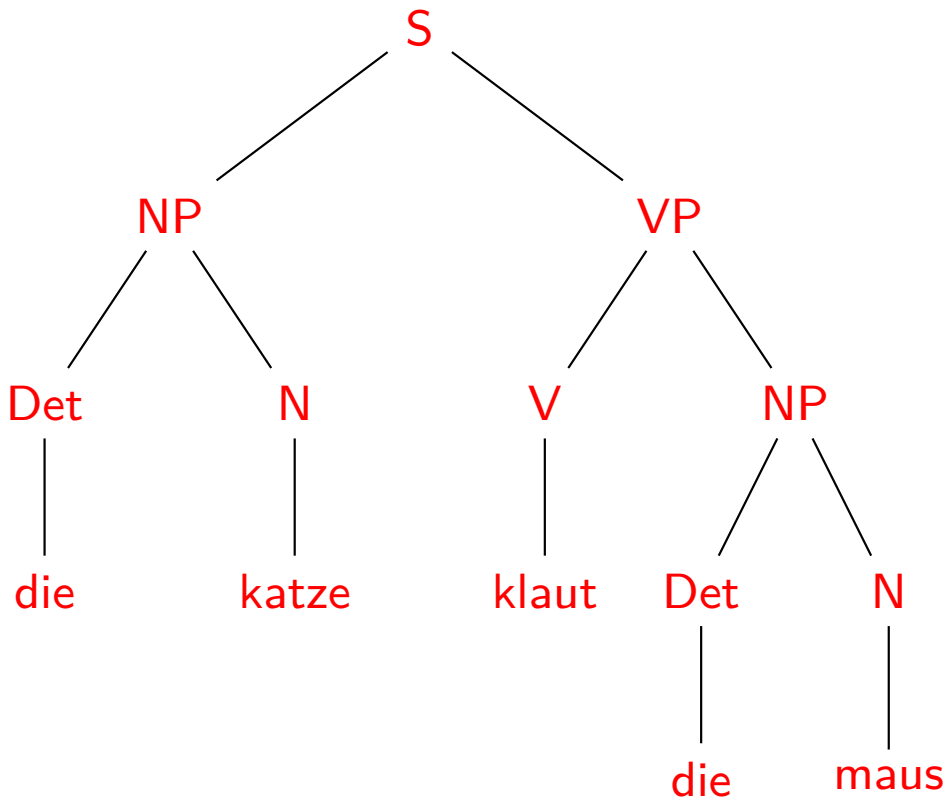
top-down, left-to-right, depth-first

Prologs Parsingstrategie

Linksableitung:

S
- ⊢ NP VP
- ⊢ Det N VP
- ⊢ die N VP
- ⊢ die katze VP
- ⊢ die katze V NP
- ⊢ die katze klaut NP
- ⊢ die katze klaut Det N
- ⊢ die katze klaut die N
- ⊢ die katze klaut die maus

Ableitungsbaum:



top-down, left-to-right, depth-first

Beispiel

```
s -> np, vp.

s([die,katze,klaut,die,maus],[]):-
          np([die,katze,klaut,die,maus],[klaut,die,maus]),
          vp([klaut,die,maus],[]).

np -> det, n.

np([die,katze,klaut,die,maus],[klaut,die,maus]):-
          det([die,katze,klaut,die,maus],[katze,klaut,die,maus]),
          n([katze,klaut,die,maus],[klaut,die,maus]).

det -> [die].

det([die,katze,klaut,die,maus],[katze,klaut,die,maus]).

n -> [katze].

n([katze,klaut,die,maus],[klaut,die,maus]).
```

Beispiel (Fortsetzung)

```
vp -> v, np.

vp([klaut,die,maus],[]):-
          v([klaut,die,maus],[die,maus]),
          np([die,maus],[]).

v -> [klaut].

v([klaut,die,maus],[die,maus]).

np -> det,n.

np([die,maus],[]):-
          det([die,maus],[maus]),
          n([maus],[]).

det -> [die].

det([die,maus],[maus]).

n -> [maus].

n([maus],[]).
```

Rechtslineare Grammatiken: Wiederholung

- Eine Grammatik ist rechtslinear, wenn jede ihrer Regeln die Form

  $A \rightarrow a\ B$

  oder die Form

  $A \rightarrow a$

  hat (a steht für ein Terminal und A und B für Nichtterminale).

Rechtslineare Grammatiken: Wiederholung

- Eine Grammatik ist rechtslinear, wenn jede ihrer Regeln die Form

  $A \rightarrow a\ B$

  oder die Form

  $A \rightarrow a$

  hat (a steht für ein Terminal und A und B für Nichtterminale).
- Zu jeder Sprache, die von einer rechtslinearen Grammatik generiert wird, gibt es einen endlichen Automaten, der die Sprache akzeptiert und umgekehrt.

Rechtslineare Grammatiken: Wiederholung

- Eine Grammatik ist rechtslinear, wenn jede ihrer Regeln die Form

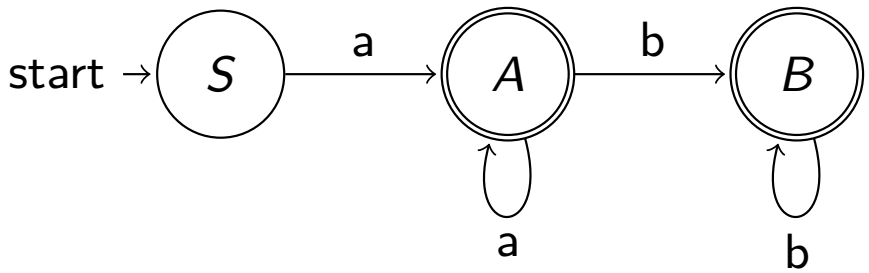
  $A \rightarrow a\ B$

  oder die Form

  $A \rightarrow a$

  hat (a steht für ein Terminal und A und B für Nichtterminale).
- Zu jeder Sprache, die von einer rechtslinearen Grammatik generiert wird, gibt es einen endlichen Automaten, der die Sprache akzeptiert und umgekehrt.

Beispiel: Sprache a^+b^*



$S \rightarrow a\ A$
$S \rightarrow a$
$A \rightarrow a\ A$
$A \rightarrow b\ B$
$A \rightarrow a$
$A \rightarrow b$
$B \rightarrow b\ B$
$B \rightarrow b$

Rechtslineare Grammatiken: Wiederholung

- Eine Grammatik ist rechtslinear, wenn jede ihrer Regeln die Form

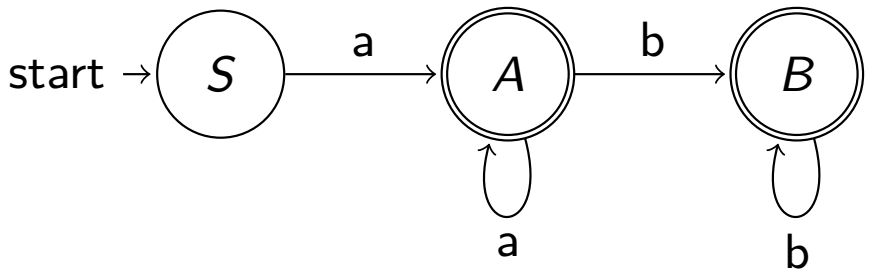
  $A \rightarrow a\ B$

  oder die Form

  $A \rightarrow a$

  hat (a steht für ein Terminal und A und B für Nichtterminale).
- Zu jeder Sprache, die von einer rechtslinearen Grammatik generiert wird, gibt es einen endlichen Automaten, der die Sprache akzeptiert und umgekehrt.

Beispiel: Sprache a^+b^*



$S \rightarrow a\ A$
$S \rightarrow a$
$A \rightarrow a\ A$
$A \rightarrow b\ B$
$A \rightarrow a$
$A \rightarrow b$
$B \rightarrow b\ B$
$B \rightarrow b$

- Eine Sprache ist regulär, wenn sie von einer rechtslinearen Grammatik generiert bzw. von einem endlichen Automaten akzeptiert wird.

Rechtslineare Grammatiken: Beispiel

Die Sprache a^+b^* wird von folgender rechtslinearer Grammatik generiert:
$S \to aS$, $S \to bB$, $B \to \epsilon$, $B \to bB$ (Vorsicht: nicht im strengen Sinne rechtslinear wg. $B \to \epsilon$).
Modellierung als DCG:

```
s --> [a], s.
s --> [a], bblock.
bblock --> [].
bblock --> [b], bblock.
```

Rechtslineare Grammatiken: Beispiel

Die Sprache a^+b^* wird von folgender rechtslinearer Grammatik generiert: $S \to aS$, $S \to bB$, $B \to \epsilon$, $B \to bB$ (Vorsicht: nicht im strengen Sinne rechtslinear wg. $B \to \epsilon$).
Modellierung als DCG:

```
s --> [a], s.
s --> [a], bblock.
bblock --> [].
bblock --> [b], bblock.
```

externe Repräsentation:

```
bblock --> [].
bblock --> [b], bblock.
```

interne Repräsentation:

```
bblock(A, A).
bblock([b|A], B) :-
    bblock(A, B).
```

▸ Übung

Rekursion: Erweiterung um Adjektive

„eine kleine, kleine, ... Maus“

```
np --> det, adjs, n.
adjs --> [].
adjs --> adj, adjs.
adj --> [kleine].
det --> [eine].
n --> [maus].
```

```
?- np(X,[]).
X = [eine,maus] ? ;
X = [eine,kleine,maus] ? ;
X = [eine,kleine,kleine,maus] ? ;
X = [eine,kleine,kleine,kleine,maus] ? ;
X = [eine,kleine,kleine,kleine,kleine,maus] ? ;
X = [eine,kleine,kleine,kleine,kleine,kleine,maus] ? ;
X = [eine,kleine,kleine,kleine,kleine,kleine,kleine,maus] ? ;
X = [eine,kleine,kleine,kleine,kleine,kleine,kleine,kleine,maus] ? ;
...
```

Linksrekursion

Erweiterung um Konjunktionen:
„Die Maus jagt die Katze **und** die Katze klaut die Maus.“

```
s --> s, conj, s.
s --> np, vp.
conj --> [und].
```

Loopt bei jedem Satz.

Linksrekursion

Erweiterung um Konjunktionen:
„Die Maus jagt die Katze **und** die Katze klaut die Maus.“

```
s --> s, conj, s.
s --> np, vp.
conj --> [und].
```

Loopt bei jedem Satz. ⇒ Verbesserung:

```
s --> np, vp.
s --> s, conj, s.
conj --> [und].
```

Loopt bei jedem ungrammatischen Satz.

Linksrekursion

Erweiterung um Konjunktionen:
„Die Maus jagt die Katze **und** die Katze klaut die Maus.“

```
s --> s, conj, s.
s --> np, vp.
conj --> [und].
```

Loopt bei jedem Satz. ⇒ Verbesserung:

```
s --> np, vp.
s --> s, conj, s.
conj --> [und].
```

Loopt bei jedem ungrammatischen Satz. ⇒ Verbesserung:

```
s --> simple_s.
s --> simple_s, conj, s.
simple_s --> np, vp.
conj --> [und].
```

Diese Technik zur Vermeidung von Linksrekursion nennt man *left-corner transform*.

Zusammenfassung DCGs

DCGs bieten eine sehr einfache, direkte Methode zur Formulierung von kontextfreien Grammatiken in Prolog.

Allerdings

- haben wir bisher keine einfache Möglichkeit, Grammatiken um häufige, generelle Phänomene wie Kongruenz zwischen Konstituenten zu erweitern (z.B. KNG-Kongruenz zwischen Adjektiv und Nomen),
- haben wir keine Möglichkeit mächtigere Grammatiken als kontextfreie zu formulieren.

$\Rightarrow$ parametrisierte DCGs ermöglichen diese Erweiterungen.

Zusammenfassung Kapitel 7

- Wir haben gesehen, dass Grammatiken, die append/3 einsetzen sehr ineffizient sind.
- Differenzlisten ermöglichen die Implementierung effizienter Grammatiken.
- DCGs bieten eine sehr einfache, direkte Methode zur Formulierung von kontextfreien Grammatiken in Prolog.
- DCGs werden intern in Grammatiken mit Differenzlisten übersetzt.

- **Keywords**: kontextfreie Grammatiken, DCGs, (Links-)Rekursion
- **Wichtig**: Vermeide Linksrekursion in DCGs
- **Ausblick Kapitel 8**: parametrisierte DCGs

Übung: Grammatiken in Prolog

- Formulieren Sie für jede der vier Grammatiken die Anfrage, mit der Sie abfragen können, ob der Satz „Eine Katze klaut eine Maus“ von der Grammatik generiert wird.
- Stellen Sie Ihre Anfrage im Tracemodus.
- Formulieren Sie für jede der vier Grammatiken die Anfrage mit der Sie alle Sätze, die von der Grammatik generiert werden, ausgegeben bekommen (Vorsicht, s3 loopt).
- Können Sie vorhersagen, welcher Satz jeweils zuerst generiert wird?

▸ zurück

Übung: externe/interne Notation

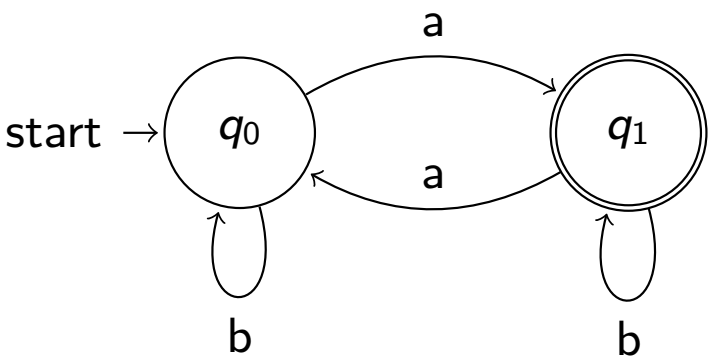
Übertragen Sie die folgenden DCGs in die interne Prolognotation:

```
1  a --> b, c.
2  b --> c.
3  c --> [].
4  c --> [ha,hu].
5  d --> a, b, c, d.
6  d --> a, [hu], c. % Zusatzaufgabe
```

▸ zurück

Übung: rechtslineare Grammatik für formale Sprache

1. Geben Sie eine rechtslineare Grammatik an, die die Sprache generiert, die von folgendem endlichen Automaten akzeptiert wird:



2. Schreiben Sie die Grammatik als DCG in Prolog.
3. Welche Sprache wird von der Grammatik generiert, welche Eigenschaften haben alle Wörter der Sprache?

zurück

Übung: DCG formale kontextfreie Sprache

1. Übertragen Sie die folgende kontextfreie Sprache in eine Prolog DCG. Welche Sprache wird von der Grammatik generiert?
   S $\rightarrow$ A, S, B
   S $\rightarrow \epsilon$
   A $\rightarrow$ a
   B $\rightarrow$ b
2. Geben Sie eine kontextfreie Grammatik an, die die Sprache generiert, die aus allen Palindromen über dem Alphabet $\{c, d\}$ besteht.

Übung: Erweiterte Grammatik

Erweiteren Sie die Grammatik mit den Regeln
$S \rightarrow NP\ VP$, $NP \rightarrow Det\ N$, $VP \rightarrow V\ NP$,
$Det \rightarrow$ eine|die|keine, $N \rightarrow$ katze|maus, $V \rightarrow$ jagt|klaut
um Regeln, für

- Adjektive (klein, schnell,...) und
- Konjunktionen (und, oder)

Implementieren Sie Ihre Grammatik als DCG in Prolog. Speichern Sie Ihre Grammatik in einer eigenen Datei. Wir werden sie in den kommenden Wochen erweitern und zusätzliche Prädikate definieren, die die Arbeit mit der Grammatik vereinfachen.

Erweitern Sie ihre Grammatik zusätzlich um

- den Eigennamen „otto“ und/oder
- die Pluralformen „mäuse“ und „katzen“.

Bearbeiten Sie Aufgabe 3 aus der ‘Practical Session’ zu Kapitel 7 aus “Learn Prolog Now!” (Zusatzaufgabe).