Prolog

8. Kapitel: parametrisierte DCGs

Dozentin: Wiebke Petersen

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

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

Erweiterung unserer DCG um Plurale

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

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

Erweiterung unserer DCG um Plurale

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

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

Bei der Erweiterung um Plurale

- steigt die Zahl der Regeln,
- werden Regeln schwerer lesbar,
- gehen Generalisierungen verloren.

Erweiterung unserer DCG um Plurale

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

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

Bei der Erweiterung um Plurale

- steigt die Zahl der Regeln,
- werden Regeln schwerer lesbar,
- gehen Generalisierungen verloren.

```
% Grammatikregeln:
s --> s_sing.
s --> s_plur.
s_sing --> np_sing, vp_sing.
s_plur --> np_plur, vp_plur.
np --> np_sing.
np --> np_plur.
np_sing --> det_sing, n_sing.
np_plur --> det_plur, n_plur.
np_plur --> n_plur.
vp_sing --> v_sing, np.
vp_plur --> v_plur, np.

% Lexikon:
det_sing --> [eine].
det_sing --> [die].
det_plur --> [die].
n_sing --> [maus].
n_sing --> [katze].
n_plur --> [maeuse].
n_plur --> [katzen].
v_sing --> [jagt].
v_plur --> [jagen].
```

Ausweg: parametrisierte DCGs

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

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

DCG Regeln können mit zusätzlichen **Parametern** oder **Merkmalen** angereichert werden.

```
% mit Pluralformen
% parametrisierte DCG
% Grammatikregeln:
s --> np(Num), vp(Num).
np(Num) --> det(Num), n(Num).
vp(Num) --> v(Num), np(_).

% Lexikon:
det(sing) --> [eine].
det(_) --> [die].
det(plur) --> [].
n(sing) --> [maus].
n(sing) --> [katze].
n(plur) --> [maeuse].
n(plur) --> [katzen].
v(sing) --> [jagt].
v(plur) --> [jagen].
```

Parametrisierte DCGs

```
1  % Grammatikregeln:
2  s --> np(Num), vp(Num).
3  np(Num) --> det(Num), n(Num).
4  vp(Num) --> v(Num), np(_).
5
6  % Lexikon:
7  det(sing) --> [eine].
8  det(_) --> [die].
9  det(plur) --> [].
10 n(sing) --> [maus].
11 n(sing) --> [katze].
12 n(plur) --> [maeuse].
13 n(plur) --> [katzen].
14 v(sing) --> [jagt].
15 v(plur) --> [jagen].
```

Zeile 2: NP und VP müssen im Numerus übereinstimmen.

Zeile 3: der Numerus der NP wird bestimmt von dem Numerus von Det und N.

Zeile 4: der Numerus der VP wird bestimmt vom Numerus des Verbs. Der Numerus der NP ist egal.

Zeile 8: Synkretismus: Die Wortform „die“ wird im Singular und im Plural verwendet.

Zeile 9: Im Plural kann der Artikel weggelassen werden.

Parametrisierte DCGs

```
1  % Grammatikregeln:
2  s --> np(Num), vp(Num).
3  np(Num) --> det(Num), n(Num).
4  vp(Num) --> v(Num), np(_).
5
6  % Lexikon:
7  det(sing) --> [eine].
8  det(_) --> [die].
9  det(plur) --> [].
10 n(sing) --> [maus].
11 n(sing) --> [katze].
12 n(plur) --> [maeuse].
13 n(plur) --> [katzen].
14 v(sing) --> [jagt].
15 v(plur) --> [jagen].
```

Zeile 2: NP und VP müssen im Numerus übereinstimmen.

Zeile 3: der Numerus der NP wird bestimmt von dem Numerus von Det und N.

Zeile 4: der Numerus der VP wird bestimmt vom Numerus des Verbs. Der Numerus der NP ist egal.

Zeile 8: Synkretismus: Die Wortform „die“ wird im Singular und im Plural verwendet.

Zeile 9: Im Plural kann der Artikel weggelassen werden.

Wie kann diese Grammatik um Maskuline (z.B. „Hund“) und Neutra (z.B. „Pferd“) erweitert werden?

Erweiterung unserer Grammatik um Genus

Kommen Maskuline und Neutra hinzu, muss auf folgendes geachtet werden:

- die Genuskongruenz zwischen Artikel und Nomen („der Hund“ vs. „♯ das Hund“)
- die Wahl des korrekten Kasus für Subjekt und Objekt („der Hund jagt den Hund“ vs. „♯ der Hund jagt der Hund“)

Das Genus wird als zusätzlicher Parameter modelliert:

```
% Grammatikregeln:
s --> np((nom,Num,_)), vp(Num).
np(KNG) --> det(KNG), n(KNG).
vp(Num) --> v(Num), np((akk,_,_)).
```

```
% Lexikon:
det((nom,sing,mas)) --> [der].
det((akk,sing,mas)) --> [den].
det((_,sing,fem)) --> [die].
det((_,sing,neu)) --> [das].
det((_,plur,_)) --> [die].
det((_,plur,_)) --> [].
n((_,sing,fem)) --> [maus].
n((_,plur,_)) --> [maeuse].
n((_,sing,fem)) --> [katze].
n((_,plur,_)) --> [katzen].
n((_,sing,mas)) --> [hund].
n((_,plur,mas)) --> [hunde].
n((_,sing,neu)) --> [pferd].
n((_,plur,neu)) --> [pferde].
v(sing) --> [jagt].
v(plur) --> [jagen].
```

▸ Übung

parametrisierte DCGs: interne Repräsentation

```
s --> np(Num), vp(Num).
np(Num) --> det(Num), n(Num).
n(sing) --> [maus].
```

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

np(Num, A, C):-
    det(Num, A, B),
    n(Num, B, C).

n(sing, [maus|A], A).
```

▸ Übung

```
p1(P11,...,P1i) -->
    p2(P21,...,P2j),
    ...,
    pn(Pn1,...,Pnk).
```

```
p1(P11,...,P1i,V1,Vn):-
    p2(P21,...,P2j,V1,V2),
    ...,
    pn(Pn1,...,Pnk,Vn-1,Vn).
```

Die Parameter werden intern als zusätzliche Argumente realisiert.

Prolog als Akzeptor, Generator, Parser

```
s --> np, vp.
np --> det, n.
vp --> v, np.
```

```
det --> [die].
n --> [maus].
v --> [jagt].
```

Prolog als Akzeptor, Generator, Parser

```
s --> np, vp.
np --> det, n.
vp --> v, np.
```

```
det --> [die].
n --> [maus].
v --> [jagt].
```

Ein **Akzeptor** ist eine Maschine, die entscheiden kann, ob ein String von einer Grammatik generiert wird.

```
?- s([die,maus,jagt,die,maus],[]).
true.
```

Prolog als Akzeptor, Generator, Parser

```
s --> np, vp.
np --> det, n.
vp --> v, np.
```

```
det --> [die].
n --> [maus].
v --> [jagt].
```

Ein **Akzeptor** ist eine Maschine, die entscheiden kann, ob ein String von einer Grammatik generiert wird.

```
?- s([die,maus,jagt,die,maus],[]).
true.
```

Ein **Generator** ist eine Maschine, die zu einer gegebenen Grammatik, die von der Grammatik generierten Strings ausgeben kann.

```
?- s(X,[]).
X = [die,maus,jagt,die,maus].
```

Prolog als Akzeptor, Generator, Parser

```
s --> np, vp.
np --> det, n.
vp --> v, np.
```

```
det --> [die].
n --> [maus].
v --> [jagt].
```

Ein **Akzeptor** ist eine Maschine, die entscheiden kann, ob ein String von einer Grammatik generiert wird.

```
?- s([die,maus,jagt,die,maus],[]).
true.
```

Ein **Generator** ist eine Maschine, die zu einer gegebenen Grammatik, die von der Grammatik generierten Strings ausgeben kann.

```
?- s(X,[]).
X = [die,maus,jagt,die,maus].
```

Ein **Parser** ist eine Maschine, die zu einer gegebenen Grammatik und einem String angibt, ob der String von der Grammatik generiert wird und wenn ja, einen Ableitungsbaum zu dem String ausgibt.

Prolog als Akzeptor, Generator, Parser

```
s --> np, vp.
np --> det, n.
vp --> v, np.
```

```
det --> [die].
n --> [maus].
v --> [jagt].
```

Ein **Akzeptor** ist eine Maschine, die entscheiden kann, ob ein String von einer Grammatik generiert wird.

```
?- s([die,maus,jagt,die,maus],[]).
true.
```

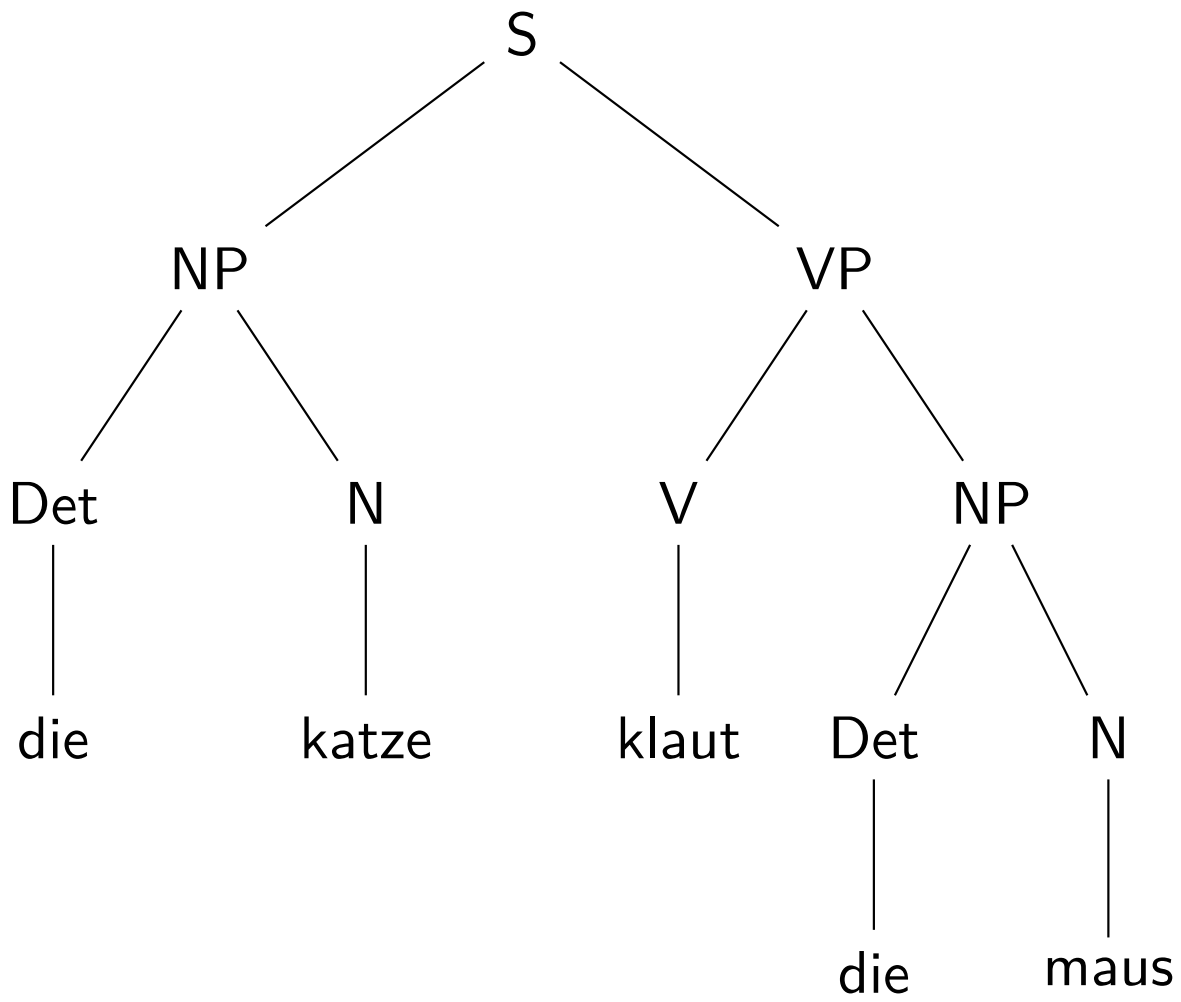
Ein **Generator** ist eine Maschine, die zu einer gegebenen Grammatik, die von der Grammatik generierten Strings ausgeben kann.

```
?- s(X,[]).
X = [die,maus,jagt,die,maus].
```

Ein **Parser** ist eine Maschine, die zu einer gegebenen Grammatik und einem String angibt, ob der String von der Grammatik generiert wird und wenn ja, einen Ableitungsbaum zu dem String ausgibt.

Prolog ist ein Akzeptor und ein Generator, aber auch ein Parser?

Ableitungsbaum repräsentiert als komplexer Prologterm




```
T = s(np(det(die), n(katze)), vp(v(klaut), np(det(die), n(maus))))
```

Prolog als Parser

Idee: Der Ableitungsbaum wird als zusätzlicher Parameter behandelt.

```
% Grammatikregeln:
s(s(NP,VP)) --> np(NP), vp(VP).
np(np(Det,N)) --> det(Det), n(N).
vp(vp(V,NP)) --> v(V), np(NP).

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

?- s(T,[die, katze, klaut, die, maus], []).
T = s(np(det(die), n(katze)), vp(v(klaut), np(det(die), n(maus))))
```

▸ Übung

kontextsensitive Sprache $a^n b^n c^n$

Die kontextsensitive Sprache $a^n b^n c^n$ wird von der Grammatik
S $\rightarrow \epsilon$, S $\rightarrow$ A B S C, B A $\rightarrow$ A B, A $\rightarrow$ a, B $\rightarrow$ b, C $\rightarrow$ c
generiert.
Diese Sprache lässt sich nicht direkt als DCG schreiben.

kontextsensitive Sprache $a^n b^n c^n$

Die kontextsensitive Sprache $a^n b^n c^n$ wird von der Grammatik
S $\rightarrow \epsilon$, S $\rightarrow$ A B S C, B A $\rightarrow$ A B, A $\rightarrow$ a, B $\rightarrow$ b, C $\rightarrow$ c
generiert.
Diese Sprache lässt sich nicht direkt als DCG schreiben.

Wir können aber Parameter einsetzen, die das Zählen der a's, b's und c's übernehmen.

```
s --> ablock(Count),bblock(Count),cblock(Count).

ablock(0) --> [].
ablock(succ(Count)) --> [a],ablock(Count).

bblock(0) --> [].
bblock(succ(Count)) --> [b],bblock(Count).

cblock(0) --> [].
cblock(succ(Count)) --> [c],cblock(Count).
```

▸ Übung

Extraziele

Mithilfe von **Extrazielen** können wir auf die umständlichere successor-Notation für die Zähler verzichten (Nachteil: Generierung nicht mehr möglich).

```
s --> ablock(Count),bblock(Count),cblock(Count).

ablock(0) --> [].
ablock(NewCount) --> [a],ablock(Count), {NewCount is Count + 1}.
bblock(0) --> [].
bblock(NewCount) --> [b],bblock(Count), {NewCount is Count + 1}.
cblock(0) --> [].
cblock(NewCount) --> [c],cblock(Count), {NewCount is Count + 1}.
```

Externe Repräsentation:

```
ablock(NewCount) --> [a],ablock(Count), {NewCount is Count + 1}.
```

Interne Repräsentation:

```
ablock(B, [a|A], D) :-
    ablock(C, A, E),
    B is C+1,
    D=E.
```

Zusammenfassung Kapitel 8

- Wir haben Parameter kennengelernt und diese eingesetzt,
  - um grammatische Constraints wie z.B. Kongruenz zu erfassen,
  - um mithilfe eines Zählers die kontextsensitive Sprache $a^n b^n c^n$ zu modellieren.
- Wir haben gesehen, wie wir DCGs mit Extrazielen anreichern können. Dies ist möglich, da DCGs nur *notational sugar* sind.
- Mit Parametern und Extrazielen ist es möglich die Grenzen von kontextfreien Grammatiken zu verlassen.

- **Keywords**: Parameter, Extraziele
- **Wichtig**: Durch Parameter und Extraziele sind DCGs mächtiger als kontextfreie Grammatiken.
- **Ausblick Kapitel 9**: Terme und Operatoren

Übung: externe/interne Notation

Übertragen Sie die folgenden DCGs in die interne Prolognotation:

```
1 % mit Parametern
2 a(t) --> b(t), c.
3 b(t) --> [ha].
4 c --> d(s,r), [hu].
```

```
1 % mit Extrazielen
2 a --> b, c, {mag(popeye,spinat)}.
3 b(t) --> [ha], {wizart(t),sailor(popeye)}.
```

zurück

Übung: Grammatik

1. Erweitern Sie Ihre Grammatik um möglichst viele der folgenden Punkte (Vorsicht, bei manchen dieser Punkte müssen Sie das Lexikon sehr stark erweitern, gehen Sie daher schrittweise vor):
   - Maskuline und Neutra, Singular- und Pluralformen
   - transitive Verben, die ein Komplement im Dativ verlangen (z.B. „helfen“)
   - ditransitive Verben
   - Adjektive (Vorsicht, die Flexion von Adjektiven ist für Maskuline und Neutra von dem Determinationstyp abhängig: „ein schönes Pferd“, „das schöne Pferd“)
   - lokative Präpositionalphrasen wie z.B. „unter dem Tisch“, „neben dem Tisch“, „auf dem Tisch“ (es müssen nur syntaktisch wohlgeformte Sätze gebildet werden, dabei ist es egal, ob sie semantisch sinnvoll sind)
   - Eigennamen
   - Pronomen
   - Konjunktionen

   Schreiben Sie sie als parametrisierte DCG. zurück
2. Verändern Sie Ihre Grammatik so, dass Sie sie zum Parsen einsetzen können. Es soll also zu jedem Satz der Ableitungsbaum gebildet werden (hierzu müssen Sie einen zusätzlichen Parameter für den Ableitungsbaum verwenden). zurück

Übung: Separieren des Lexikons

- Lesen sie Abschnitt 8.2.1 in *Learn Prolog Now!*
- Separieren sie, wie in dem Abschnitt beschrieben, ihre Grammatik von ihrem Lexikon.

Übung: kontextsensitive Sprachen

1. Vergleichen Sie die beiden Grammatiken für die Sprache $a^n b^n c^n$ von Folie 11 und 12.
   - Wie können Sie fragen, ob die Strings *aaabbbccc* und *aabbbcc* von den Grammatiken generiert werden?
   - Können Sie mit den Grammatiken auch Strings generieren? Welche Argumente müssen jeweils bei der Anfrage instantiiert sein?
2. Schreiben Sie eine DCG, die die Sprache $a^n b^m c^n d^m$ akzeptiert.

zurück