

Definite Clause Grammar II

Übersicht

- ◆ DCG-Formalismus vs. DCG-Standard-Parser
- ◆ DCG zu Prolog-Übersetzung: Light-Version
- ◆ Erweiterungen: Komplexe Nicht-Terminal-Symbole
 - ◆ Modellierung von Kongruenz
 - ◆ Aufbau von Syntaxstrukturen
 - ◆ Abdeckungsgrad von DCGs
- ◆ Akzeptor vs. Parser
- ◆ Eingebettete Prolog-Klauseln
- ◆ Fazit: DCG und Prolog

DCG II - 1

DCG-Formalismus vs. Standard-Parser

DCG als Grammatik-Formalismus

- ◆ *Definite Clause Grammars* erlauben eine systematische Beschreibung der formalen Regularitäten einer Sprache.

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

```
det --> [the].  
det --> [a].
```

Der Standard-DCG-Parser von Prolog

- ◆ In den meisten Prologs gibt es einen Mechanismus, der DCGs automatisch in ein Prolog-Programm übersetzt, das die Sätze der von der DCG beschriebenen Sprache analysieren kann.
- ◆ Dieser Parser hat Probleme mit linksrekursiven Regeln.

DCG II - 2

DCG zu Prolog: Light-Version

Wie werden DCGs zu Prolog-Programmen übersetzt?

- *Im Folgenden: Vereinfachte Version der DCG-Übersetzung*
- ◆ `translate/2` nimmt eine DCG-Regel und übersetzt (compiliert) sie in eine Prolog-Klausel.

```
translate((LHS --> RHS), (Head :- Body)) :-  
  left_hand_side(LHS, Start, End, Head),  
  right_hand_side(RHS, Start, End, Body).
```

```
np --> det, n.
```



```
np(Start, End) :-  
  det(Start, Middle),  
  n(Middle, End).
```

DCG II - 3

DCG zu Prolog: LHS

Wie wird die *Left-Hand-Side* übersetzt?

- ◆ `left_hand_side/4` setzt die *Start*- und *End*-Variable
- ◆ Nicht-Terminal *NT* muss ein Atom sein (`atom/1`)
- ◆ *Head* wird mittels `../2` als komplexer Term zusammengesetzt

```
left_hand_side(NT, Start, End, Head) :-  
  atom(NT),  
  Head =.. [NT,Start,End].
```

```
np --> det, n.
```



```
np(Start, End) :-  
  det(Start, Middle),  
  n(Middle, End).
```

DCG II - 4

DCG zu Prolog: RHS mit Terminal

Wie wird die RHS übersetzt?

- ♦ Falls ein Terminal-Symbol (listenförmig!) vorliegt
 - ▶ Abbruchbedingung
 - ▶ 'C'/3-Aufruf einsetzen mit Terminal-Symbol als 2. Argument

```
right_hand_side([T], Start, End, 'C'(Start,T,End)).
```

```
det --> [der].
```

→

```
det(Start, End) :-
    'C'(Start, der, End).
```

DCG II - 5

DCG zu Prolog: RHS rekursiv

Wie wird die *Right-Hand-Side* (RHS) übersetzt?

- ♦ Falls mehrere Symbole auf der rechten Seite sind
 - ▶ Doppelte Rekursion
 - ▶ Erzeugen der Zwischenposition *Middle*

```
nt --> s1,s2,s3.
```

↕

```
nt --> (s1,(s2,s3)).
```

```
right_hand_side((S1,S2), Start, End, (Body1, Body2)) :-
    right_hand_side(S1, Start, Middle, Body1),
    right_hand_side(S2, Middle, End, Body2).
```

```
np --> det, n.
```

→

```
np(Start, End) :-
    det(Start, Middle),
    n(Middle, End).
```

DCG II - 6

DCG zu Prolog: Light-Version

Wie wird die *Right-Hand-Side* übersetzt?

- ♦ Falls Nicht-Terminal-Symbol
 - ▶ NT-Symbol muss ein Atom sein
 - ▶ Rumpfteil wird als komplexer 2-stelliger Term zusammengesetzt mit `../2`
 - ▶ Abbruchbedingung für die Rekursion

```
right_hand_side(NT, Start, End, Body) :-
    atom(NT),
    Body =.. [NT,Start,End].
```

```
vp --> v.
```

→

```
vp(Start, End) :-
    v(Start, End).
```

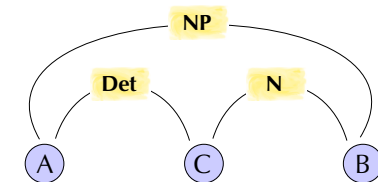
DCG II - 7

DCG zu Prolog: Light-Version

Die rekursive Dekomposition von `right_hand_side/4` fügt die Aufrufe mit den jeweiligen Position korrekt ein:

```
?- translate((np --> det, n), X).
X = np(A,B):-det(A,C),n(C,B) ? ;
no
```

Eine NP besteht zwischen A und B,
falls zwischen A und C ein Det besteht
und zwischen C und B ein N.

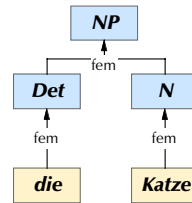


DCG II - 8

Komplexe Nicht-Terminal-Symbole

DCG mit komplexen NT-Symbolen

- ◆ Zusätzliche Information kann repräsentiert und unifiziert werden!
- ◆ Beispiel: **Kongruenz (Agreement)**
 - ▶ Im Deutschen müssen Genus von Artikel und Nomen übereinstimmen!
 - ▶ Merkmale werden hinaufunifiziert!



```

np(Genus) -->
  det(Genus),
  n(Genus).
n(fem) --> [katze].
det(fem) --> [die].
  
```



```

np(Genus, A, B) :-
  det(Genus, A, C),
  n(Genus, C, B).
n(fem, A, B) :- 'C'(A, katze, B).
det(fem, A, B) :- 'C'(A, die, B).
  
```

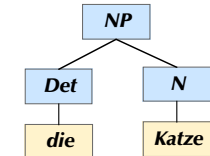
Konstruktion eines Syntaxbaums

DCG mit komplexen NT-Symbolen: Syntaxstrukturen

- ◆ In der LHS jeder Grammatikregel wird die der Regel entsprechende syntaktische Datenstruktur im Zusatzargument aufgebaut.

```

n(n(katze)) --> [katze].
np(np(Det,N)) --> det(Det), n(N).
det(det(die)) --> [die].
  
```



- ◆ Die Grammatiksymbole wie S, NP, Det werden dabei gerne mehrdeutig verwendet:
 - ◆ als Parseprädikate, Datenstrukturen und Variablen
 - ▶ Man könnte auch jeweils andere Bezeichner verwenden!

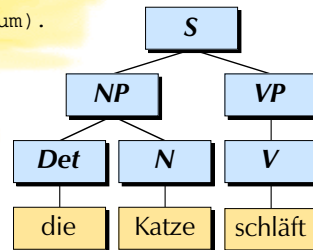
Parsen mit komplexen Symbolen

```

s(s(NP,VP)) --> np(NP,nom,Num), vp(VP,Num).
np(np(Det,N),Kas,Num) -->
  det(Det,Gen,Kas,Num), n(N,Gen,Kas,Num).
vp(v(V),Num) --> v(V,Num).
  
```

```

n(n(katze),fem,Kas,sg) --> [katze].
v(v(schlaeft),sg) --> [schlaeft].
v(v(schlafen),pl) --> [schlafen].
det(det(die),fem,nom,sg) --> [die].
det(det(der),fem,gen,sg) --> [der].
...
  
```



```

?- phrase(s(Baum), [die,katze,schlaeft]).
Baum = s(np(det(die),n(katze)),vp(v(schlaeft)))
  
```

Akzeptor – Parser

Ein Programm zur syntaktischen Analyse

- ◆ nimmt eine Kette von Wörtern entgegen.
- ◆ beurteilt, ob die Eingabe gemäss den Regeln einer Grammatik zulässig ist.

Akzeptoren

- ◆ antworten nur mit »zulässig« bzw. »nicht zulässig«.

Parser

- ◆ geben bei zulässigen Eingaben zusätzlich die mögliche(n) syntaktische(n) Struktur(en) der Eingabekette aus.

Mächtigkeit von DCG

Komplexe Nicht-Terminal-Symbole beeinflussen die mathematischen Eigenschaften des Formalismus.

- Die modellierte Sprache kann ausserhalb der Klasse der kontextfreien Sprachen liegen!

Grammatik für $\{a^n b^n c^n \mid n \geq 1\}$

```
s --> as(N), bs(N), cs(N).  
as(1) --> [a].  
as(succ(A)) --> [a], as(A).  
bs(1) --> [b].  
bs(succ(B)) --> [b], bs(B).  
cs(1) --> [c].  
cs(succ(C)) --> [c], cs(C).
```

DCG II – 13

Anfragen

```
?- phrase(s, [a,b,c]).  
yes  
  
?- phrase(s, [a,a,b,c]).  
no
```

Eingebettete Prolog-Klauseln

Prädikatsaufrufe innerhalb geschweifeter Klammern

- werden bei der Übersetzung von DCGs unverändert übernommen.
- Dadurch können beliebige Prolog-Programme in die Grammatik eingebettet werden.

```
lex(cat, n).  
lex(dog, n).  
lex(cow, n).
```

Beispiel

Anstelle von lexikalischen DCG-Regeln wird ein Lexikon aus lex/2-Fakten verwendet mit Lexem und lexikalischer Kategorie als Argumenten.

```
n(n(N)) -->  
[N],  
{lex(N, n)}.
```



```
n(n(A), B, C) :-  
'C'(B, A, C),  
lex(A, n).
```

DCG II – 14

Eingebettete Prolog-Klauseln

Vorteile der Prolog-Einbettung

- manchmal kann die Effizienz gesteigert werden
 - Es darf auch der cut/0 eingesetzt werden. (Sogar ohne geschweifte Klammern!)
- kontextsensitive Sprachen können erkannt werden

Nachteile der Prolog-Einbettung

- keine deklarative Spezifikation der Grammatik
- Grammatiken werden schnell unübersichtlich
- Reine DCGs können als Formalismus auch von anderen Programmiersprachen/Grammatikkompilierern verarbeitet werden; mit eingebettetem Prolog wird eine Prolog-Abhängigkeit geschaffen

DCG II – 15

Fazit Prolog und DCGs

Vorteile von DCGs

- nützlich zum schnellen Spezifizieren/Ausprobieren einer Mini-Grammatik
- in Prolog ist ein einfacher Top-Down-Parser bereits eingebaut
- als reiner Formalismus unabhängig von Umsetzung in Parser-Programm
- DCG-Grammatiken sind in Prolog via Interpreter als Parser-Programm verfügbar

Nachteile von Standard-Prolog-DCG-Verarbeitung

- »Aufhängen« bei linksrekursiven Grammatiken
- eher ineffizientes Verfahren
 - bei mehrdeutigen Grammatiken werden unter Umständen Teile des Satzes mehrmals analysiert
 - lexikalische Regeln werden ineffizient für grossen Lexika

DCG II – 16