DCG II

Heute:

- DCGs mit extra Argumenten
- DCGs mit Agreementmerkmalen
- Parsebäume
- Extra Tests

Extra Argumente

Beispiele:

`x - -> y.`	$\longrightarrow$	`x(A,B):- y(A,B).`
`s - -> np, vp.`	$\longrightarrow$	`s(A,B):- np(A,C), vp(C,B).`
`x(foo, ba) - -> y.`	$\longrightarrow$	`x(foo,ba,A,B) :- y(A,B).`
`s(f) - -> np, vp.`	$\longrightarrow$	`s(f,A,B):- np(A,C), vp(C,B).`
`x - -> y(foo,a,b).`	$\longrightarrow$	`x(A,B):- y(foo,a,b,A,B).`
`x(X) - -> y(X).`	$\longrightarrow$	`x(A,B,C):- y(A,B,C).`
`x(_) - -> y(_).`	$\longrightarrow$	`x(A,B,C):- y(D,B,C).`

In der Prolog internen Darstellung von DCGs, sind die letzten zwei Argumente immer die Differenz-Listen. Zusätzliche Argumente werden davor geschrieben.

DCGs mit Agreementmerkmalen

Man möchte die Grammatik vom letzten Mal

```
s - -> np, vp.
np - -> det, n.
vp - -> v, np.
vp - -> v.
det - -> [the].
det - -> [a].
n - -> [woman].
n - -> [man].
v - -> [shoots].
```

mit Pronomen erweitern. So dass z.B. auch die folgenden Sätze grammatisch sind:

1. 'She shoots him'

2. 'He shoots her'

Erster Versuch

```
s - -> np, vp.          n - ->[man].
np - -> det, n.         n - ->[woman].
vp - -> v, np.          pro - -> [he].
vp - -> v.              pro - -> [she].
det - -> [the].         pro - -> [him].
det - -> [a].           pro - -> [her].
v - -> [shoots].        np - -> pro.
```

```
-?  s([she,shoots,him],[]).
yes
```

Aber:

```
-?  s([she,shoots,he],[]).

yes
```

Zweiter Versuch

```
s - -> np_subject, vp.
np_subject - -> det, n.
np_object - -> det, n.
np_subject - -> pro_subject.
np_object - -> pro_object.
vp - -> v, np_object.
vp - -> v.

det - -> [the].
det - -> [a].
n - -> [woman].
n - -> [man].
v - -> [shoots].

pro_subject - -> [he].
pro_subject - -> [she].
pro_object - -> [him].
pro_object - -> [her].
```

Zweiter Versuch

Jetzt erlaubt Prolog nur richtige Sätze.

-? s([she,shoots,him],[]).

yes

-? s([she,shoots,he],[]).

no

-? s([her,shoots,him],[]).

no

Eine (viel!) bessere Lösung

```
s - -> np(subject), vp.
np(_) - -> det, n.
np(X) - -> pro(X).
vp - -> v, np(object).
vp - -> v.
det - -> [the].
det - -> [a].
n - -> [woman].
n - -> [man].
v - -> [shoots].
pro(subject) - -> [he].
pro(subject) - -> [she].
pro(object) - -> [him].
pro(object) - -> [her].
```

Eine bessere Lösung

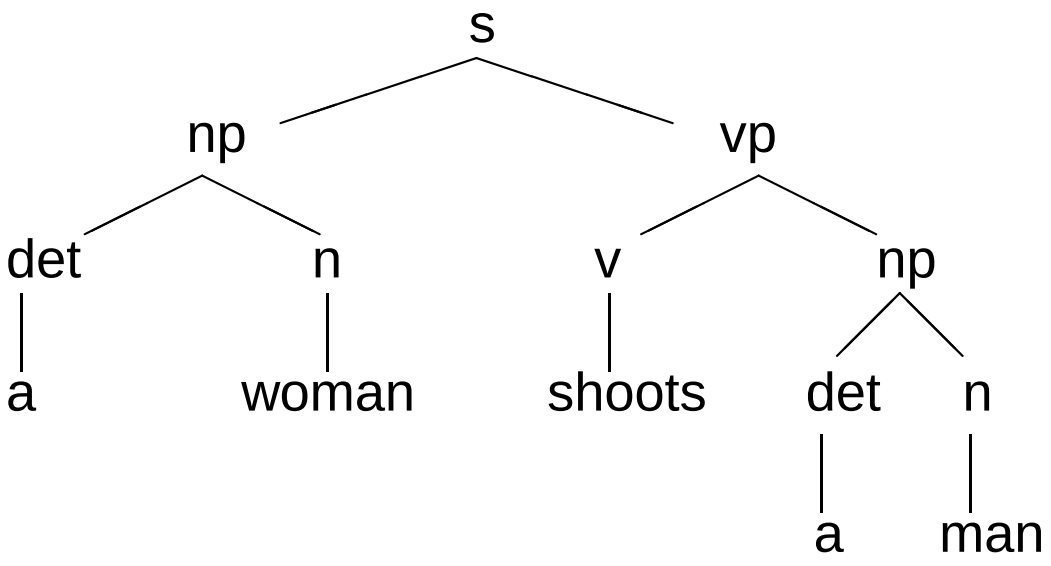
np(X) - -> pro(X).

intern: np(A,B,C):- pro(A,B,C).

np(_) - -> det, n.

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

Parsebäume



Darstellung in Prolog:

```
s(np(det(a),n(woman)),
    vp(v(shoots),np(det(a),n(man))))
```

Parsebäume Bauen

n - -> [woman]. baut den Parsebaum n(woman)

$\Rightarrow$ n(n(woman)) - -> [woman].

intern: n(n(woman), [woman||A], A).

s - -> np,vp. baut den Parsebaum s(NP-Parsebaum, VP-Parsebaum)

$\Rightarrow$ s(s(NP-Baum,VP-Baum)) - -> np(NP-Baum), vp(VP-Baum).

intern: s(s(NP,VP),A,B):- np(NP,A,C), vp(VP,C,B).

Parsen mit extra Argumenten

```
s(s(NP,VP)) - -> np(NP), vp(VP).
np(np(DET,N)) - -> det(DET), n(N).
vp(vp(V,NP)) - -> v(V), np(NP).
vp(vp(V)) - -> v(V).
det(det(the)) - -> [the].
det(det(a)) - -> [a].
n(n(woman)) - -> [woman].
n(n(man)) - -> [man].
v(v(shoots)) - -> [shoots].
```

-? s(T,[a,woman,shoots],[]).

Call: (6) s(_G292, [a, woman, shoots], [])

Call: (7) np(_G353, [a, woman, shoots], _G362)

Call: (8) det(_G356, [a, woman, shoots], _G365)

Exit: (8) det(det(a), [a, woman, shoots], [woman, shoots])

Call: (8) n(_G357, [woman, shoots], _G367)

Exit: (8) n(n(woman), [woman, shoots], [shoots])

Exit: (7) np(np(det(a), n(woman)), [a, woman, shoots], [shoots])

Call: (7) vp(_G354, [shoots], [])

Call: (8) v(_G363, [shoots], [])

Exit: (8) v(v(shoots), [shoots], [])

Exit: (7) vp(vp(v(shoots)), [shoots], [])

Exit: (6) s(s(np(det(a), n(woman)), vp(v(shoots))),
[a, woman, shoots], [])

T = s(np(det(a), n(woman)), vp(v(shoots)))

Aufgaben

1. Erweitere die DCG von Folien 7 mit Regeln die Pluralpronomen erlauben. Die neue Grammatik soll z.B. Sätze wie ’They shoot the woman’ und ’A woman shoots them’ akzeptieren.
2. Mache aus der DCG von Aufgabe 1 einen Parser, indem du in einem zusätzlichen Argument den Parsebaum aufbaust.

Die Sprache $a^n b^n c^n$

```
s(Count) - -> as(Count), bs(Count), cs(Count).

as(0) - -> [].
as(succ(X)) - -> [a], as(X).

bs(0) - -> [].
bs(succ(X)) - -> [b], bs(X).

cs(0) - -> [].
cs(succ(X)) - -> [c], cs(X).
```

DCGs mit extra Argumenten sind ausdrucksstärker als CFGs. (Die Sprache $a^n b^n c^n$ ist nicht kontextfrei.)

Die Sprache $a^n b^n c^n$

-? s(Count,Wort,[]).

Count=0

Wort=[]

;

Count=succ(0)

Wort=[a,b,c]

; Count=succ(succ(0))

Wort=[a,a,b,b,c,c]

;

...

Extra Tests

Extra Tests erlauben den Aufruf von beliebigen Prologprädikaten auf der rechten Seite von DCG Regeln.

```
a --> b, c, {prädikat}.
```
intern:
```
a(A,B):- b(A,C), c(C,B), prädikat.
```

```
a --> b, {prädikat}, c.
```
intern:
```
a(A,B):- b(A,C), prädikat, c(C,B).
```

Die Sprache $a^n b^n c^n$ mit extra Tests

```
s(Count) - -> as(Count), bs(Count), cs(Count).

as(0) - -> [].
as(Count) - -> [a], {NewCount is Count - 1},
               as(NewCount).

bs(0) - -> [].
bs(Count) - -> [b], {NewCount is Count - 1},
               bs(NewCount).

cs(0) - -> [].
cs(Count) - -> [c], {NewCount is Count - 1},
               cs(NewCount).
```

Achtung: Diese DCG kann nur zum Generieren verwendet werden.

Zum Erkennen nicht, da dann die Variable Count nicht instantiiert ist.

Extra Tests

as(Count) - -> [a], {NewCount is Count - 1} , as(NewCount).

intern: as(Count,A,B):- 'C'(A,a,C), NewCount is Count - 1,
as(NewCount,C,B).

-? s(3,L,[]).

L = [a,a,a,b,b,b,c,c,c].

yes

Trennung von Grammatik und Lexikon

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

det - -> [W], lex(W,det).
n - -> [W], lex(W,n).
v - -> [W], lex(W,v).
```

```
lex(the,det).
lex(a,det).
lex(woman,n).
lex(man,n).
lex(shoots,v).
```

Zusammenfassung

Heute haben wir gesehen

- wie extra Argumente und extra Tests die Ausdrücksfähigkeit von DCGs erweitern
- wie man Parsebäume mit DCGs realisieren kann

Nächste Woche Freitag (24.11.) Cut und Negation.

Übungsaufgaben: Die Übungen sind auf der Webseite.