

Definite Clause Grammars I

Heute:

- kontextfreie Grammatiken
- und ihre Verarbeitung in Prolog

Grammatiken

Eine Menge von Regeln, die beschreiben, wie grammatikalische Sätze aussehen.

$S \Rightarrow NP VP$

$NP \Rightarrow Det N$

$VP \Rightarrow V NP$

$VP \Rightarrow V$

$Det \Rightarrow a$

$Det \Rightarrow the$

$N \Rightarrow woman$

$N \Rightarrow man$

$V \Rightarrow shoots$

Kontextfreie Grammatiken für natürliche Sprachen

Terminal: die Wörter einer Sprache. Z.B. *a*, *wizard*, *fly*, ...

Nichtterminale: die syntaktischen Kategorien. Z.B. *S*, *NP*, *VP*, ...

Regeln: Kontextfreie Regeln haben auf der linken Seite immer genau ein Nichtterminal. Auf der rechten Seite können entweder ein oder mehrere (nichtterminale und terminale) Symbole stehen oder das leere Wort ϵ .

Zum Beispiel:

$S \Rightarrow NP VP$ und

$N \Rightarrow man$ und

$NP \Rightarrow \epsilon$ sind kontextfreie Regeln.

$V NP \Rightarrow V Det N$ ist keine kontextfreie Regel.

Kontextfreie Regeln als Rewrite Regeln

Das linke Symbol kann durch das was rechts steht ersetzt werden.

$S \Rightarrow NP VP$	$Det \Rightarrow a$
$NP \Rightarrow Det N$	$Det \Rightarrow the$
$VP \Rightarrow V NP$	$N \Rightarrow woman$
$VP \Rightarrow V$	$N \Rightarrow man$
	$V \Rightarrow shoots$

Eine Ableitungen:

$S \rightarrow NP VP \rightarrow Det N VP \rightarrow the N VP \rightarrow the woman VP \rightarrow the woman V \rightarrow the woman shoots$

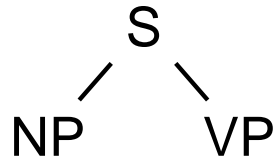
Ein Satz ist grammatisch, wenn er (angefangen von einem gegebenen Startsymbol) abgeleitet werden kann.

Kontextfreie Regeln zum “Bäumebauen”

Regeln sind “Baupläne” für Teilbäume.

Zum Beispiel:

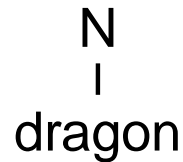
$S \Rightarrow NP VP$



$VP \Rightarrow V$

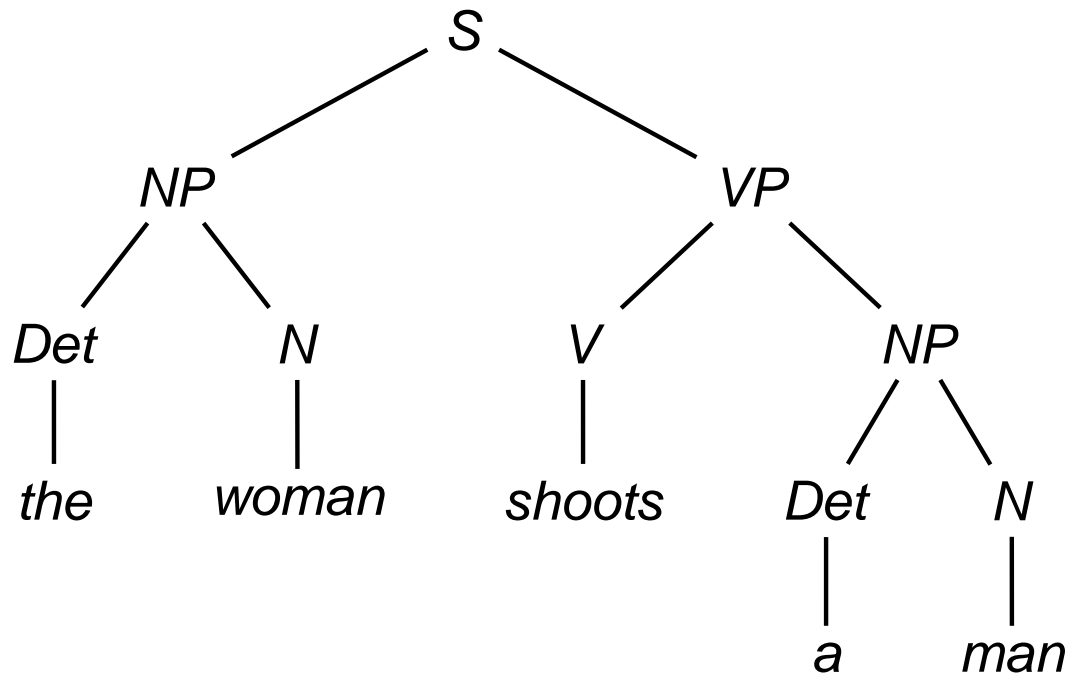


$N \Rightarrow dragon$



Parsebäume

Ein Parsebaum für den Satz *the woman shoots a man*:



$S \Rightarrow NP VP$

$NP \Rightarrow Det N$

$VP \Rightarrow V NP$

$VP \Rightarrow V$

$Det \Rightarrow a$

$Det \Rightarrow the$

$N \Rightarrow woman$

$N \Rightarrow man$

$V \Rightarrow shoots$

Jeder lokale Teilbaum ist durch eine Regel der Grammatik lizenziert.

Grammatische und ungrammatische Sätze

- Ein Satz ist grammatisch (bezüglich einer Grammatik), wenn man für diesen Satz einen Parsebaum bauen kann.
- Man sagt dann auch, dass dieser Satz zur Sprache, die von dieser Grammatik generiert wird, gehört.

Erkenner vs. Parser

- Ein Erkenner (Recognizer) ist ein Programm, das erkennt, ob ein Satz grammatisch ist oder nicht.
- Ein Parser ist ein Programm, das erkennt, ob ein Satz grammatisch ist oder nicht und das außerdem den Parsebaum zurückgibt, falls der Satz grammatisch ist.

Eine kontextfreie Grammatik als Prologprogramm

- Wir repräsentieren (Teil-)Sätze als Listen von Atomen.
z.B. [a, woman, shoots] oder [the, man]
- Wir repräsentieren Grammatikregeln als Prolog-Regeln und Fakten.

$S \Rightarrow NP VP$: eine Liste von Wörtern ist eine S -Liste, wenn sie in eine NP -Liste und eine VP -Liste zerlegt werden kann.

`s(L) :- append(L1, L2, L), np(L1), vp(L2).`

$N \Rightarrow woman$: [woman] ist eine N -Liste.

`n([woman]).`

Ein Erkenner

```
s(Z) :- append(X,Y,Z), np(X), vp(Y).
```

```
np(Z) :- append(X,Y,Z), det(X), n(Y).
```

```
vp(Z) :- append(X,Y,Z), v(X), np(Y).
```

```
vp(Z) :- v(Z).
```

```
det([the]).
```

```
det([a]).
```

```
n([woman]).
```

```
n([man]).
```

```
v([shoots]).
```

Was man damit machen kann

```
?- s([the,woman,shoots,a,man]).
```

```
yes
```

```
?- s([the,shoots,woman]).
```

```
no
```

```
?- s(X).
```

```
X = [the,woman,shoots,the,woman] ;
```

```
X = [the,woman,shoots,the,man] ;
```

```
X = [the,woman,shoots,a,woman] ;
```

```
X = [the,woman,shoots,a,man] ;
```

```
X = [the,woman,shoots]
```

```
...
```

```
yes
```

```
?- np([the,man]).
```

```
yes
```

und so funktioniert:

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

- Append zerlegt die Liste in [] und [the,woman,shoots,a,man].
- Es wird getestet, ob [] eine NP ist. Schlägt fehl, also backtracking.
- Append liefert die nächste Zerlegung [the] und [woman,shoots,a,man].
- [the] ist auch keine NP also backtracking.
- Die nächste Zerlegung ...

s(Z) :- append(X,Y,Z), np(X), vp(Y).

np(Z) :- append(X,Y,Z), det(X), n(Y).

vp(Z) :- append(X,Y,Z), v(X), np(Y).

vp(Z) :- v(Z).

det([the]).

det([a]).

n([woman]).

n([man]).

v([shoots]).

Differenzlisten

[a , woman , shoots]

wird repräsentiert durch

[a , woman , shoots] , []

oder

[a , woman , shoots , a , man] , [a , man]

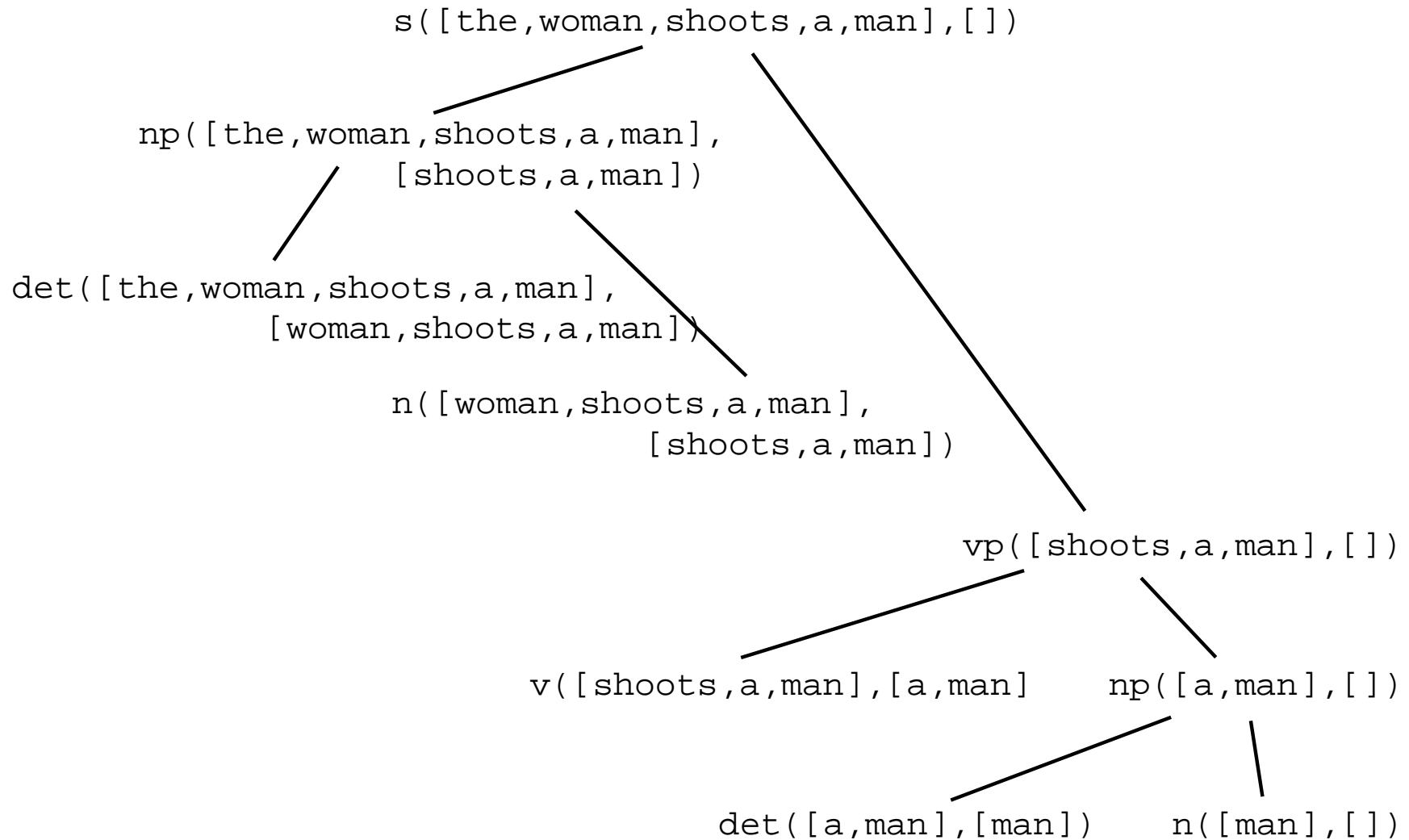
oder

[a , woman , shoots , ploggle , woggle] , [ploggle , woggle]

oder

...

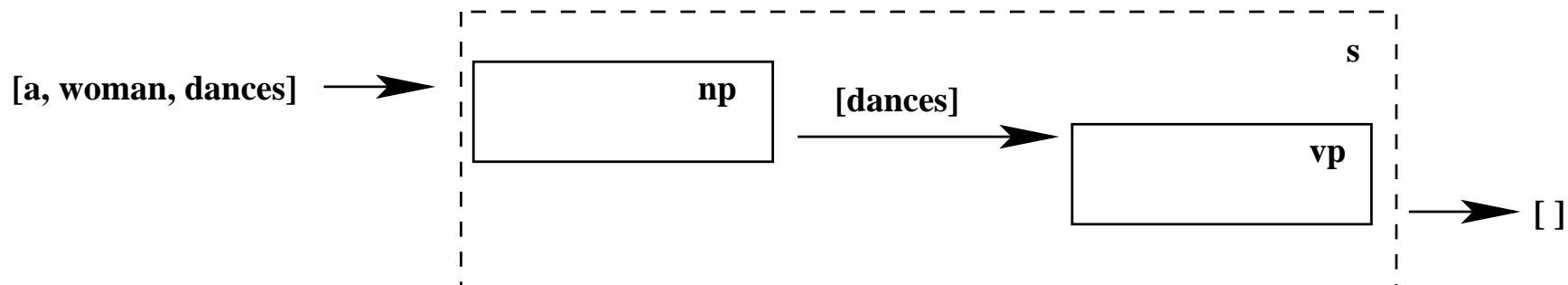
Differenzlisten zur Darstellung von Phrasen



Erkenner mit Differenzlisten: Idee

- Für jede Kategorie ein Prädikat, das eine (Wort-)Liste als Eingabe nimmt, von vorne eine Sequenz der Kategorie wegstreicht und dann den Rest zurückgibt.
- Wenn das Prädikat `np/2` z.B. die Liste `[a, man, snores]` als Eingabe bekommt, soll es `[snores]` zurückgeben.
- Wenn wir von einer Eingabe-Liste zunächst eine NP wegstreichen können und danach noch eine VP wegstreichen können, dann haben wir einen Satz erkannt.

`s(X, Z) :- np(X, Y), vp(Y, Z).`



Ein Erkenner mit Differenzlisten

`s(X,Z) :- np(X,Y), vp(Y,Z).`

`np(X,Z) :- det(X,Y), n(Y,Z).`

`vp(X,Z) :- v(X,Y), np(Y,Z).`

`vp(X,Z) :- v(X,Z).`

`det([the|W],W).`

`det([a|W],W).`

`n([woman|W],W).`

`n([man|W],W).`

`v([shoots|W],W).`

Mögliche Anfragen:

`?- s([a,woman,shoots],[])`

`?- s(L,[])`

Aufgaben

1. Erweitert den Differenzlisten-Erkenner so, dass Nomen auch von Präpositionalphrasen modifiziert werden können. D.h. er soll NPs wie z.B. `[the, man, under, the, table]` akzeptieren.
2. Erweitert den Differenzlisten-Erkenner so, dass (beliebig viele) Sätze durch `[und]` miteinander verknüpft werden können. Z.B. soll `[the, man, dances, and, the, woman, smiles, and, the, dog, barks]` akzeptiert werden. Tipp: Hier ist der Unterschied zwischen deklarativer und prozeduraler Bedeutung wieder wichtig.

D(efinite) C(lause) G(rammar)s

DCG Notation

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

interne Darstellung

```
s(X,Z) :- np(X,Y), vp(Y,Z).  
np(X,Z) :- det(X,Y), n(Y,Z).  
vp(X,Z) :- v(X,Y), np(Y,Z).  
vp(X,Z) :- v(X,Z).  
det([the|W],W).  
det([a|W],W).  
n([woman|W],W).  
n([man|W],W).  
v([shoots|W],W).
```

Manche Prologimplementationen übersetzen terminale Regeln mit Hilfe eines Funktors 'C':

```
n --> [woman]          'C'(A, woman, B)
```

Rekursive Grammatikregeln

$S \Rightarrow NP VP$

$NP \Rightarrow Det N$

$VP \Rightarrow V NP$

$VP \Rightarrow V$

$Det \Rightarrow a$

$Det \Rightarrow the$

$N \Rightarrow woman$

$N \Rightarrow man$

$V \Rightarrow shoots$

Rekursive Grammatikregeln

$S \Rightarrow NP VP$

$NP \Rightarrow Det N$

$VP \Rightarrow V NP$

$VP \Rightarrow V$

$Det \Rightarrow a$

$Det \Rightarrow the$

$N \Rightarrow woman$

$N \Rightarrow man$

$V \Rightarrow shoots$

$S \Rightarrow S Conj S$

$Conj \Rightarrow and$

$Conj \Rightarrow or$

$Conj \Rightarrow but$

Wieviele verschiedene Sätze generiert/akzeptiert diese Grammatik?

und als DCG - Version 1

s --> s conj s.

s --> np, vp.

np --> det, n.

vp --> v, np.

vp --> v.

det --> [the].

det --> [a].

n --> [woman].

n --> [man].

v --> [shoots].

conj --> [and].

conj --> [or].

conj --> [but].

Endlosschleife bei s([a, woman, shoots], [])

und als DCG - Version 2

s --> np, vp.

s --> s conj s.

np --> det, n.

vp --> v, np.

vp --> v.

det --> [the].

det --> [a].

n --> [woman].

n --> [man].

v --> [shoots].

conj --> [and].

conj --> [or].

conj --> [but].

Endlosschleife bei s ([woman, shoot], [])

und als DCG - Version 3

s --> simple_s.

s --> simple_s conj s.

simple_s --> np, vp.

np --> det, n.

vp --> v, np.

vp --> v.

det --> [the].

det --> [a].

n --> [woman].

n --> [man].

v --> [shoots].

conj --> [and].

conj --> [or].

conj --> [but].

Formale Sprachen

- Formale Sprachen sind Mengen von Strings.
- Z.B. die Sprache $a^n b^n$. Das ist die Menge aller Strings die aus gleich langen Blöcken von as und bs bestehen.
- Kontextfreie formale Sprachen kann man natürlich auch durch DCGs repräsentieren.

```
s --> [].
```

```
s --> left, s, right.
```

```
left --> [a].
```

```
right --> [b].
```


Zusammenfassung: DCGs

- DCGs sind eine Notation für kontext-freie Grammatiken.
- Intern werden DCGs in ein Prologprogramm übersetzt, das man zum Erkennen und Generieren verwenden kann.
- Deswegen muss man beim Schreiben von DCGs immer auch an ihre prozedurale Bedeutung denken.

Nächste Woche: Agreement-Features in DCGs.

Übungsaufgaben: Das Übungsblatt ist auf der Web-Seite. Abgabe ist am nächsten Freitag.