Natuurlijke-taalverwerking

Week 2

Overzicht

Recapitulatie

- Doel: computers taal laten begrijpen
- Noodzaak: bepaal syntactische structuur
  - Computationele grammatica
  - Lexicon
  - Parser (ontleder)
- Parse-boom: expliciete syntactische structuur

Recapitulatie (2)

- Context-vrije grammatica's
- $X \rightarrow w$
  - X is een 'non-terminal' symbool
  - w is een reeks van terminal, non-terminal of epsilon symbolen
- Bijvoorbeeld:

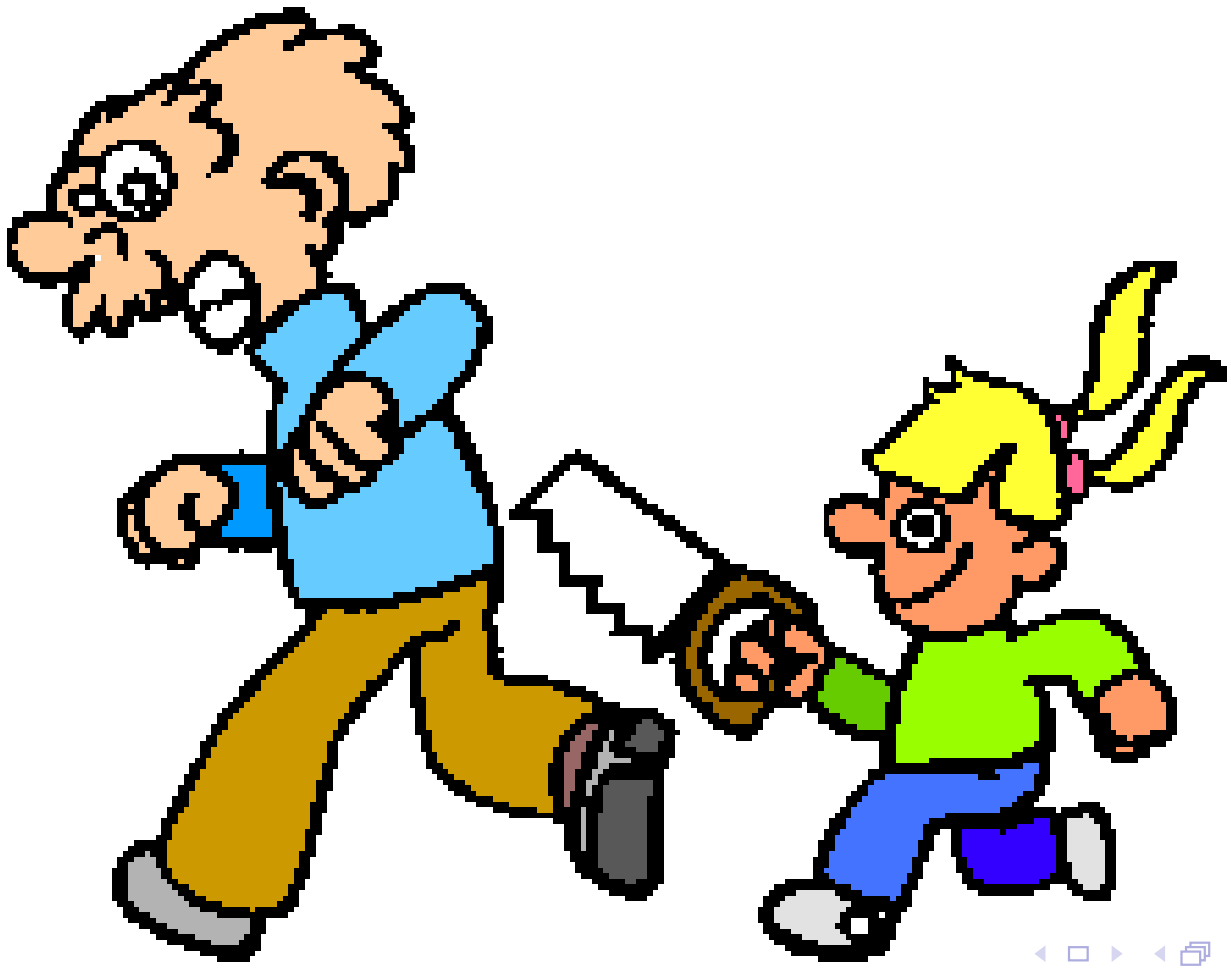
S	$\rightarrow$	NP VP
NP	$\rightarrow$	Det N
Det	$\rightarrow$	*de*
N	$\rightarrow$	*man*
VP	$\rightarrow$	*loopt*

Recapitulatie (2 - boom)

S	$\rightarrow$	NP VP
NP	$\rightarrow$	Det N
Det	$\rightarrow$	*de*
N	$\rightarrow$	*man*
VP	$\rightarrow$	*loopt*

Recapitulatie (3)

- Als er meer dan één mogelijke lezing van een zin is: *ambiguïteit*
- Structurele ambiguïteit: *Ik zag de man met de verrekijker.*
- Lexicale ambiguïteit: *Wij zagen vader niet meer.*

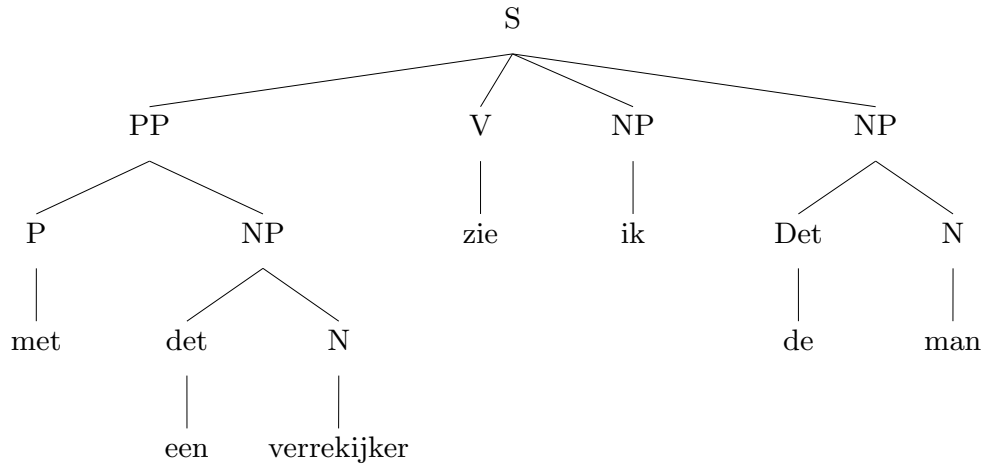


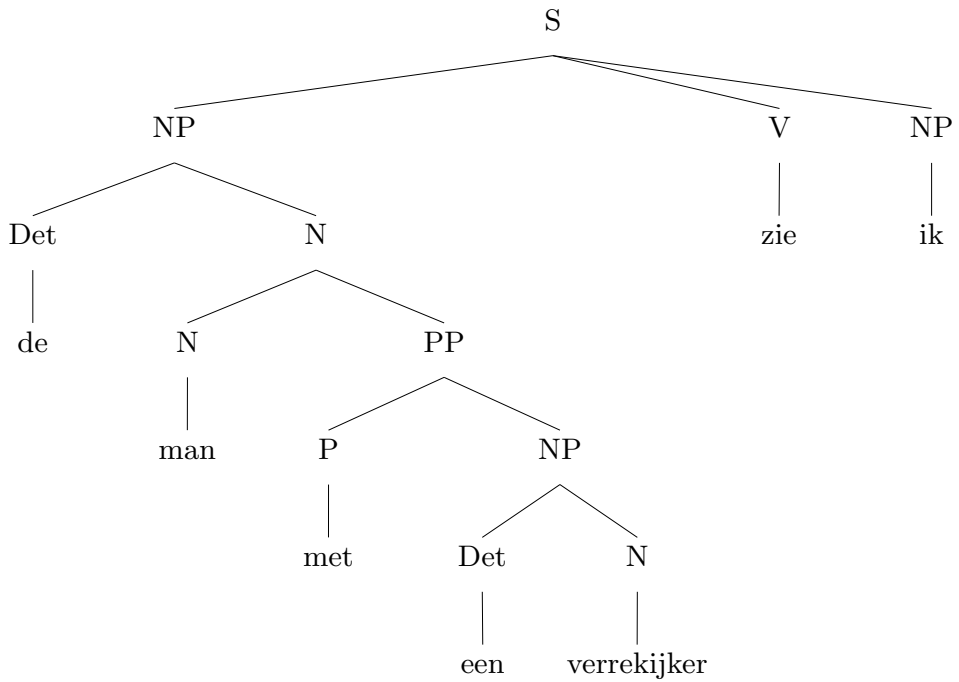
Recapitulatie (3 - PPs)

Syntactische structuur

- Woordvolgorde;
- naamval;
- getal-, persoons-, en geslachtskenmerken.

Woordvolgorde





Andere syntactische beperkingen

- naamval
  - *Wie kust hij?*
  - *Wie kust hem?*
  - *Hij kust haar*
  - **Hij kust zij*
- getal
  - *De mannen roepen de vrouw*
  - **De mannen roept de vrouw*
- andere formele kenmerken
  - *met bossen overdekt was*
  - *met bossen overdekte was*

Recursie (rechts)

- [*De man met* [*de verrekijker*]]
- [*De man met* [*de verrekijker van* [*de buurman*]]]
- [*De man met* [*de verrekijker van* [*de buurman van* [*mijn dochter*]]]]

Recursie (links)

- [[*De man*] *zijn verrekijker*]
- [[[*De man*] *zijn dochter*] *zijn verrekijker*]
- [[[[*De man*] *zijn dochter*] *zijn buurman*] *zijn verrekijker*]

Recursie (centrum?)

- [*De man die* [*de vrouwen*] *kent*]
- [*De man die* [*de vrouwen die* [*hun kinderen*] *troosten*] *kent*]
- [*De man die* [*de vrouwen die* [*hun kinderen* [*die* [*hun speelgoed*] *kwijt zijn*] *troosten*] *kent*]
- [*De man die* [*de vrouwen die* [*hun kinderen* [*die* [*hun speelgoed dat* [*ik*] *ze gegeven had*] *kwijt zijn*] *troosten*] *kent*]

Overzicht

Parsing as Deduction

$$\boxed{\text{NP} \rightarrow \text{Det N}}$$

- Een **NP** kan worden herschreven als een **Det** gevolgd door een **N**;
- **NP** is waar als **Det** en **N** waar zijn.

```
np :- det, n.
```

Verschillen tussen CFG en Prolog

$$\boxed{\text{NP} \rightarrow \text{Det N}}$$

```
np :- det, n.
```

Waarom kunnen we deze vertaling niet rechtstreeks maken?

Verschillen tussen CFG en Prolog

`np :- det, n.`	$=$	`np :- n, det.`
NP $\rightarrow$ Det N	$\neq$	NP $\rightarrow$ N Det
`np :- det, n.`	$=$	`np :- det, n, n.`
NP $\rightarrow$ Det N	$\neq$	NP $\rightarrow$ Det N N

- **(Woord-)volgorde** speelt een rol
- Aantal **Voorkomens** van een woord/constituent speelt een rol.

CFG in Prolog

de roze olifant ontsnapt

0 de 1 olifant 2 ontsnapt 3

```
word(0,de,1).
word(1,olifant,2).
word(2,ontsnapt,3).
```

CFG in Prolog (2)

Det $\rightarrow$ *de*
N $\rightarrow$ *olifant*
NP $\rightarrow$ Det N

vertalen we als

```
det(P0,P1) :- word(P0,de,P1).
n(P0,P1) :- word(P0,olifant,P1).
np(P0,P2) :- det(P0,P1), n(P1,P2).
```

Kort voorbeeld

```
det(P0,P1) :- word(P0,de,P1).
n(P0,P1) :- word(P0,olifant,P1).
np(P0,P2) :- det(P0,P1), n(P1,P2).
```

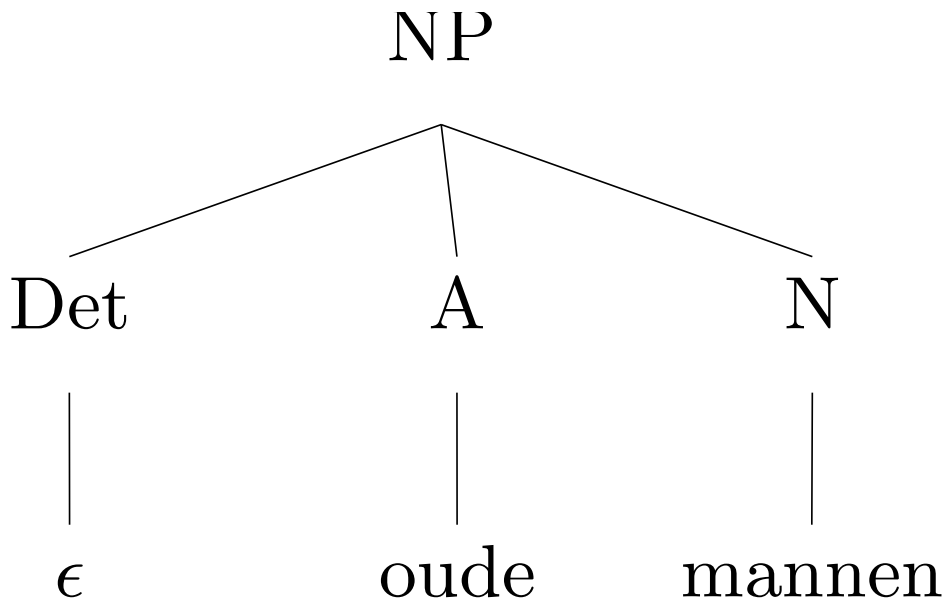
```
word(0,de,1).
word(1,olifant,2).
```

Meer voorbeelden

```
s(P0,P2) :- np(P0,P1), vp(P1,P2).
np(P0,P3) :- det(P0,P1), a(P1,P2), n(P2,P3).
vp(P0,P1) :- v(P0,P1).
vp(P0,P2) :- v(P0,P1), np(P1,P2).
det(P0,P1) :- word(P0,de,P1).
det(P0,P1) :- word(P0,het,P1).
det(P0,P0).
n(P0,P1) :- word(P0,man,P1).
n(P0,P1) :- word(P0,huis,P1).
v(P0,P1) :- word(P0,verlaat,P1).
v(P0,P1) :- word(P0,schreeuwt,P1).
```

Epsilon-regels

$$\text{NP} \rightarrow \text{Det A N}$$
$$\text{Det} \rightarrow \epsilon$$




```
det(P0,P0).
```

Prolog-lijsten als string-posities

de olifant ontsnapt

0 de 1 olifant 2 ontsnapt 3

```
0 = [de,olifant,ontsnapt]
1 = [olifant,ontsnapt]
2 = [ontsnapt]
3 = [ ]
```

```
word([de,olifant,ontsnapt],de,[olifant,ontsnapt]).
word([olifant,ontsnapt],olifant,[ontsnapt]).
word([ontsnapt],ontsnapt,[]).
```

Herken het patroon

```
word([de,olifant,ontsnapt],de,[olifant,ontsnapt]).
word([olifant,ontsnapt],olifant,[ontsnapt]).
word([ontsnapt],ontsnapt,[]).
```

```
word([Word|Words],Word,Words).
```

My first parser

```
parse(Words) :-
  s(Words,[]).

word([Word|Words],Word,Words).

%% ...  en verder de grammaticaregels:
s(P0,P2) :- np(P0,P1), vp(P1,P2).
np(P0,P2) :- det(P0,P1), n(P1,P2).
vp(P0,P1) :- v(P0,P1).
vp(P0,P2) :- v(P0,P1), np(P1,P2).
det(P0,P1) :- word(P0,de,P1).
det(P0,P1) :- word(P0,het,P1).
det(P0,P0).
n(P0,P1) :- word(P0,man,P1).
n(P0,P1) :- word(P0,huis,P1).
v(P0,P1) :- word(P0,verlaat,P1).
v(P0,P1) :- word(P0,schreeuwt,P1).
```

Overzicht

De DCG pijl

$$\boxed{\text{NP} \rightarrow \text{Det N}}$$

wordt in Prolog geschreven als

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

en wordt door Prolog vertaald in

```
np(P0,P2) :- det(P0,P1), n(P1,P2).
```

Prolog `term_expansion` vertaalt code bij het inlezen in andere code...

De DCG pijl (2)

$$\begin{array}{l} \mathrm{N} \rightarrow \textit{olifant} \\ \mathrm{Det} \rightarrow \epsilon \end{array}$$

wordt in Prolog geschreven als

```
n --> [olifant].
det --> [].
```

en wordt door Prolog vertaald in

```
n(P0,P1) :- 'C'(P0,olifant,P1).
det(P0,P1) :- P0 = P1.
```

gegeven de volgende definitie

```
'C'([Word|P],Word,P).
```

Enkelvoud/Meervoud

	Dumbo slaapt	olifanten slapen
fout:	Dumbo slapen	olifanten slaapt

$S \rightarrow NPsg\ VPsg$	$VPsg \rightarrow Vsg$
$S \rightarrow NPpl\ VPpl$	$VPpl \rightarrow Vpl$
$NPsg \rightarrow Detsg\ Nsg$	
$NPpl \rightarrow Detpl\ Npl$	
$Vsg \rightarrow$ slaapt	$Vpl \rightarrow$ slapen

- Dit werkt
- Maar wordt snel onoverzichtelijk (naamval, persoon, . . .)
- Zeer veel regels
- Generalizatie gemist

DCG's met variabelen

```
s --> np(Num), vp(Num).
np(Num) --> det(Num), N(Num).
vp(Num) --> v(Num).
v(sg) --> [slaapt].
n(sg) --> [olifant].
v(pl) --> [slapen].
n(pl) --> [olifanten].
```

DCG's met variabelen (2)

```
s --> np(N), vp(N).
```

wordt door Prolog vertaald als

```
s(P0,P2) :- np(N,P0,P1), vp(N,P1,P2).
```

- Variabelen voor string-posities worden altijd achteraan toegevoegd.

Woordenboeken

```
v(sg) --> [slaapt].
v(pl) --> [slapen].
v(inf) --> [slapen].
v(prt) --> [geslapen].
v(sg) --> [sliep].
.....
n(sg) --> [olifant].
n(pl) --> [olifanten].
```

Woordenboeken (2)

- Voor ieder werkwoord moeten dezelfde vormen worden gespecificeerd. (*Paradigma*).
- Dit vereist veel regels.
- **Woordenboek en regels scheiden**:
  - Woordenboek kan worden **geïmporteerd** uit een algemeen electronisch woordenboek
  - **Veranderingen in de grammatica** hebben minder invloed op woordenboek
  - Woordenboek kan **herbruikt** worden.

Scheiden Woordenboek en Grammatica

```
verb(slaapt,slapen,sliep,sliepen,geslapen).

v(sg) --> [Word], {verb(Word,_,_,_,_)}.
v(sg) --> [Word], {verb(_,_,Word,_,_)}.
v(pl) --> [Word], {verb(_,Word,_,_,_)}.
v(pl) --> [Word], {verb(_,_,_,Word,_)}.
...
```

Accolade's

- Soms wil je 'gewone' Prolog-code tussen je regels vlechten.
- Accolade-notatie staat dit toe.

```
v(sg) --> [W], {verb(W,_,_,_,_)}.
```

wordt door Prolog vertaald als:

```
v(sg,P0,P1) :- 'C'(P0,W,P1), verb(W,_,_,_,_).
```

Grammatica schrijven

Ik slaap.	*Mij slaap
Hij slaapt.	*Hem slaap
Ik zie hem.	*Ik zie hij
Hij ziet mij.	*Hij ziet ik
Jan ziet mij	
Ik zie Jan	

- *Ik* en *hij* kan alleen als onderwerp gebruikt worden (*nominatief*)
- *mij* en *hem* kan in alle posities, behalve onderwerp, gebruikt worden (*accusatief*),
- *Jan* kan in alle NP-posities gebruikt worden.

Grammatica zonder naamvallen

$$
\begin{aligned}
\texttt{s} &\rightarrow \texttt{np vp} \\
\texttt{vp} &\rightarrow \texttt{v np} \\
\texttt{np} &\rightarrow \texttt{ik} \\
\texttt{np} &\rightarrow \texttt{mij} \\
\texttt{np} &\rightarrow \texttt{hij} \\
\texttt{np} &\rightarrow \texttt{hem} \\
\texttt{np} &\rightarrow \texttt{Jan}
\end{aligned}
$$

Grammatica met naamvallen

$$
\begin{aligned}
\texttt{s} &\rightarrow \texttt{npnom vp} \\
\texttt{vp} &\rightarrow \texttt{v npacc} \\
\texttt{npnom} &\rightarrow \texttt{ik} \\
\texttt{npnom} &\rightarrow \texttt{hij} \\
\texttt{npnom} &\rightarrow \texttt{Jan} \\
\texttt{npacc} &\rightarrow \texttt{mij} \\
\texttt{npacc} &\rightarrow \texttt{hem} \\
\texttt{npacc} &\rightarrow \texttt{Jan}
\end{aligned}
$$

DCG met naamvallen

```
s --> np(nom), vp.
vp --> v, np(acc).
np(nom) --> [ik].
np(acc) --> [mij].
np(nom) --> [hij].
np(acc) --> [hem].
np(_) --> [jan].
```

DCG met naamvallen

```
s --> np(nom), vp.
vp --> v, np(acc).
np(nom) --> [W], {pronoun(W,_)}.
np(acc) --> [W], {pronoun(_,W)}.
np(_) --> [W],{proper_name(W)}.

pronoun(ik,mij).
pronoun(hij,hem).
proper_name(jan).
```

Context-free versus context-sensitive

- De taal $a^n b^n$ kan beschreven worden met een CFG, terwijl een CFG de taal $a^n b^n c^n$ niet kan beschrijven.
- Zijn DCGs krachtiger?

Uitwerking

$a^m b^n c^o$

```
s --> as, bs, cs.

as --> []
as --> [a], as.

bs --> []
bs --> [b], bs.

cs --> []
cs --> [c], cs.
```

$a^m b^n c^o$ (2)

```
s --> symbols(a), symbols(b), symbols(c).

symbols(_) --> [].
symbols(S) --> [S], symbols(S).
```

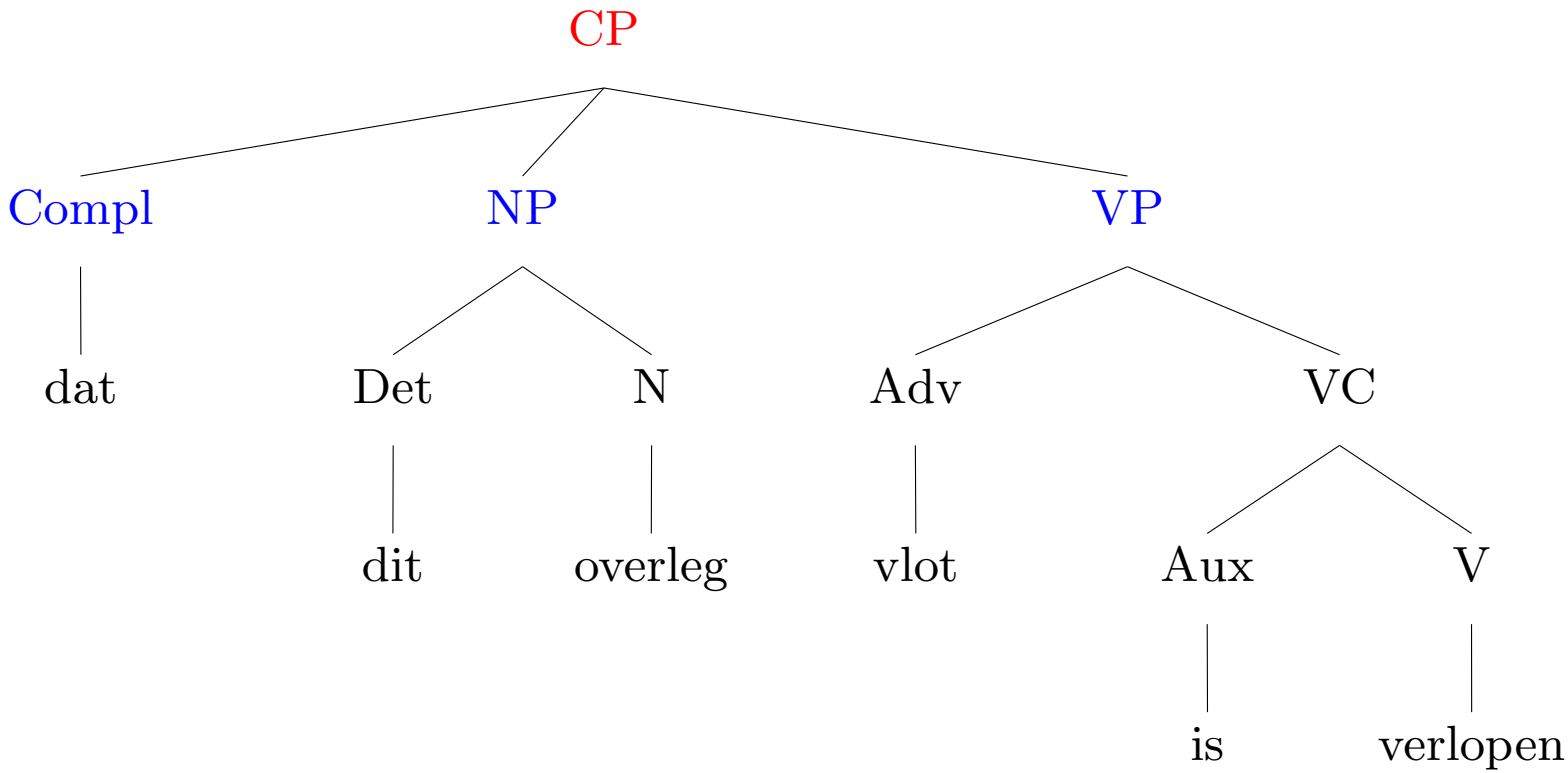
$a^n b^n c^n$ in een DCG

```
s --> symbols(Sem,a), symbols(Sem,b), symbols(Sem,c).

symbols(end,_) --> [].
symbols(s(Sem),S) --> [S], symbols(Sem,S).
```

Overzicht

Bomen


CP
Compl
NP
VP
dat
Det
N
Adv
VC
dit
overleg
vlot
Aux
V
is
verlopen

Definitie

- Een boom bestaat uit
  - Een moederknoop
  - en 0,1,2, of meer dochters
- Dochters zijn
  - zelf weer bomen
  - of woorden.

Bomen in Prolog

- `boom(Moeder,Lijst)`
- `boom(n,[overleg])`
- `boom(det,[dit])`
- `boom(np,[boom(det,[dit]),boom(n,[overleg])])`
- `b(np,[b(det,[w(dit)]),b(n,[w(overleg)])])`

Bomen in Prolog (2)

```
np --> det, n.
det --> [dit].
n --> [overleg].

np(b(np,[Det,N])) --> det(Det), n(N).

det(b(det,[w(dit)])) --> [dit].
n(b(det,[w(overleg)])) --> [overleg].

det(b(det,[w(W)])) --> [W], {determiner(W)}.
n(b(det,[w(W)])) --> [W], {noun(W)}.
```

Bomen in Prolog (3)

```
?- parse([de,hond,heeft,gesnurkt]).
  s ----- np ---- det --- de
            ---- n ----- hond
    ----- aux --- heeft
    ----- vp ---- vc ---- aux --- epsilon
                      ---- v ----- gesnurkt
```

Bomen in Prolog (4)

```
hilde$ sicstus -l opgave2
| ?- parse([de,hond,heeft,gesnurkt]).
  s ----- np ---- det --- de
            ---- n ----- hond
    ----- aux --- heeft
    ----- vp ---- vc ---- aux --- epsilon
                      ---- v ----- gesnurkt

  s ----- np ---- det --- de
            ---- n ----- hond
    ----- aux --- heeft
    ----- vp ---- vc ---- v ----- gesnurkt
                      ---- aux --- epsilon
yes
| ?-
```

Bomen in Hdrug

```
hilde$ export PATH=$PATH:/net/aps/64/bin
hilde$ export \
  NTV=/net/aps/64/src/Alpino/Hdrug/Applications/NTV1
hilde$ hdrug -flag grammar opgave2.pl -l $NTV/start
```

Bomen in Hdrug (2)