Natuurlijke-Taalverwerking

Week 5

Parsing

Overzicht

- DCG's en links-recursie
- Shift-reduce parsing
- Chart parsing
- Generatie

Links-recursie

```
ouder(geert,jan).
ouder(jan,youri).

voorouder(X,Y) :-
  voorouder(X,Z),
  ouder(Z,Y).

voorouder(X,Y) :-
  ouder(X,Y).
```

```
?- voorouder(geert,youri).
```

Links-recursie (2)

```
ouder(geert,jan).
ouder(jan,youri).

voorouder(X,Y) :-
  ouder(X,Y).

voorouder(X,Y) :-
  voorouder(X,Z),
  ouder(Z,Y).
```

```
?- voorouder(geert,youri).
```

Links-recursie en DCG's

- de bal [in de hoek]
- de bal [met rode stippen [in de hoek]]
- de bal [van de hond [met rode stippen [in de hoek]]]

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

| ?- np([de,bal,in,de,hoek],[]).
        1      1 Call: np([de,bal,in,de,hoek],[]) ?
        [...]
        4      2 Call: n([bal,in,de,hoek],[]) ?
        5      3 Call: n([bal,in,de,hoek],_3740) ?
        6      4 Call: n([bal,in,de,hoek],_4300) ?
        7      5 Call: n([bal,in,de,hoek],_4860) ?
```

Links-recursie en DCG's

Erben en Gerard
Erben en Gerard of Jan
Erben en Marianne of Gerard en Renate

```
np --> np, [en], np.
np --> np, [of], np.
```

Links-recursie en DCG's

de schaaten
Erben's schaatsen
Erben's vader's schaatsen

```
np --> det, n.
det --> np, ['\'s'].
```

Links-recursie en DCG

Als we de normale Prolog bewijsprocedure (top-down, backtracking) gebruiken, zijn links-recursieve DCG-regels problematisch.

- Optie 1: Pas grammatica aan.
- Optie 2: Pas ontleedalgoritme aan.

Eliminatie van links-recursie

```
n --> n_lex, pps.
pps --> pp, pps.
pps --> [].
```

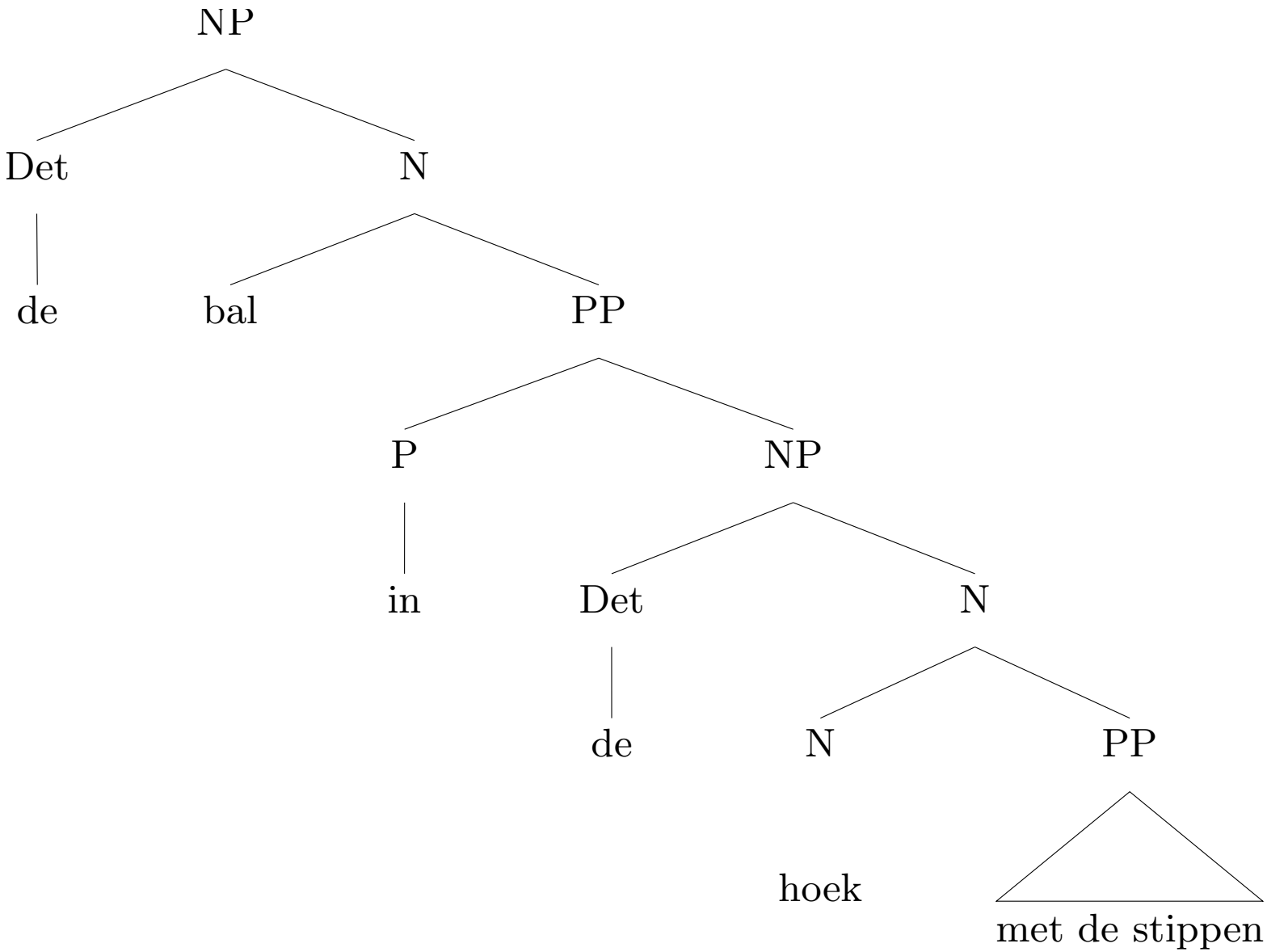
Eliminatie van links-recursie

Gegeven een grammatica met links-recursieve regels, vindt een *equivalente* grammatica zonder links-recursieve regels.

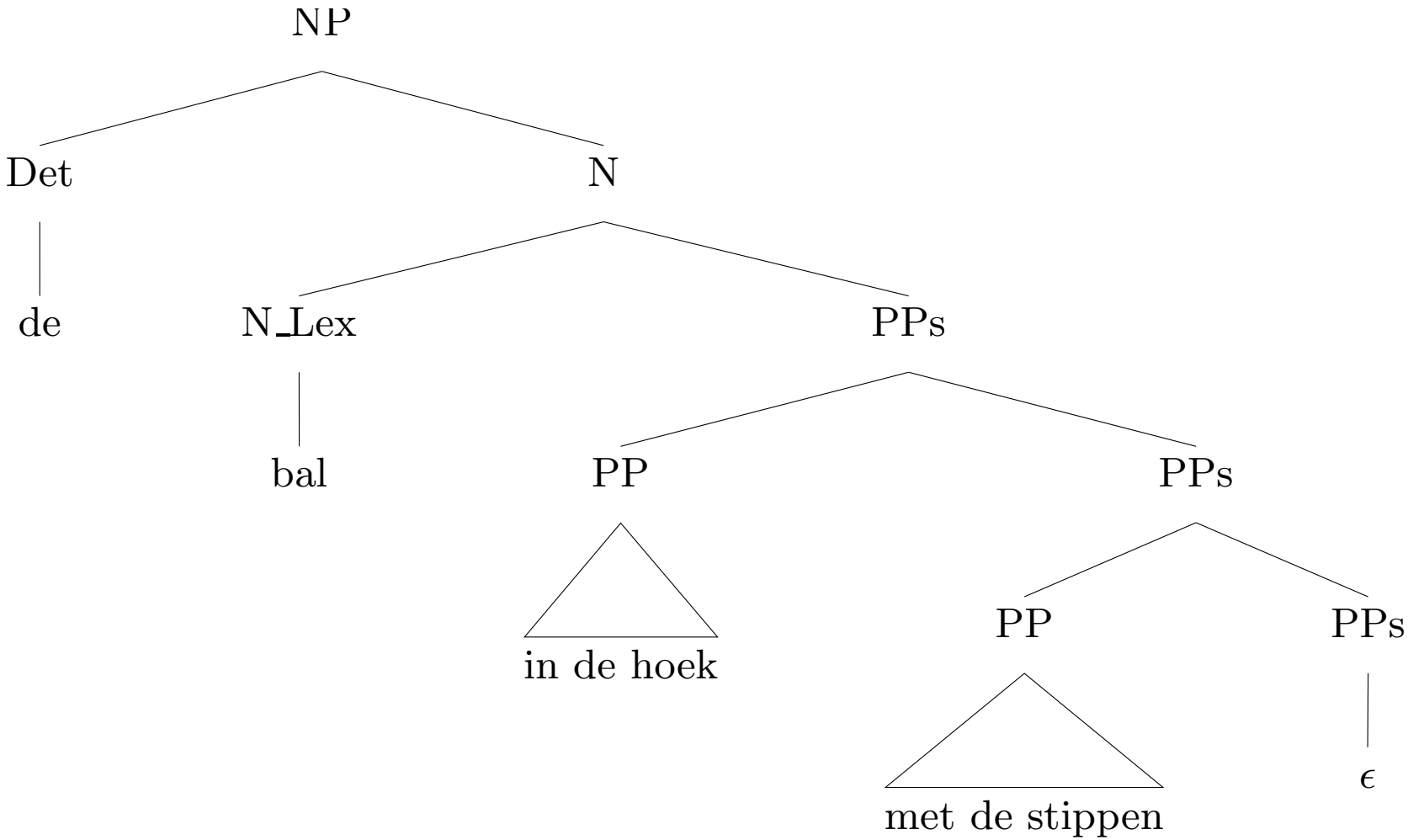
Er zijn echter verschillende vormen van equivalentie:

- *weakly equivalent*: grammatica's genereren dezelfde verzameling zinnen
- *strongly equivalent*: grammatica's genereren dezelfde verzameling zinnen met parsebomen

Boom links-recursieve grammatica



Boom herschreven grammatica



Eliminatie van links-recursie

Pas ook de parseboom aan!

Eliminatie van links-recursie

- Een grammatica met links-recursie kan automatisch worden omgezet in een grammatica zonder links-recursie
  - strongly equivalent
- Maar aantal regels kan explosief toenemen
- Regels worden onbegrijpelijk

Parse-strategieën

- Prolog (DCG) zoekt **top-down**, **depth-first**, **links-rechts**
- Alternatieven:
  - **Bottom-up**: Begin bij de input (woorden), werk naar de startcategorie (S) toe
  - **Breadth-first**: Onderzoek alle mogelijke manieren om zinsdelen te vormen parallel
  - **rechts-links**, **hoofd-gedreven**
- Alternatieve methode's zijn (vaak) robuuster en/of efficiënter

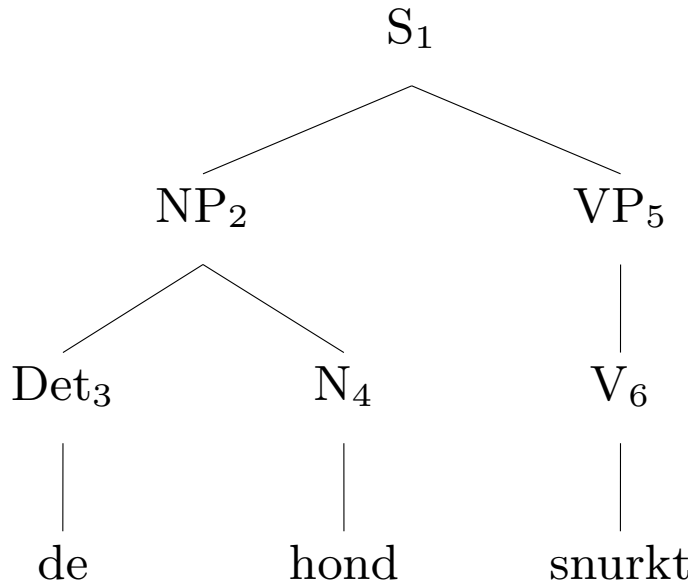
Alternatieve Parse-strategieën

- Vereist een scheiding tussen **regels (data)** en **parser (algoritme)**;
- grammatica-regels als Prolog-data (zie ook unificatie-grammatica):

```
% rule(Mother_Category,List_Daughters)

% s --> np(Agr), vp(Agr).
rule(s,[np(Agr),vp(Agr)]).

% lex(Category,Word).

% np(sg) --> [kim].
lex(np(sg),kim).
```

Top-down ontleden

S
NP VP
Det N VP
de N VP
de hond VP
de hond V
de hond snurkt



DCG = top-down

```
1.  s([de, hond, snurkt], []).
2.  np([de, hond, snurkt], P1), vp(P1, []).
3.  det([de, hond, snurkt], P2), n(P2, P1), vp(P1, []).
4.  n([hond, snurkt], P1), vp(P1, []).
5.  vp([snurkt], []).
6.  v ([snurkt], [])
7.
```

Top-down ontleden

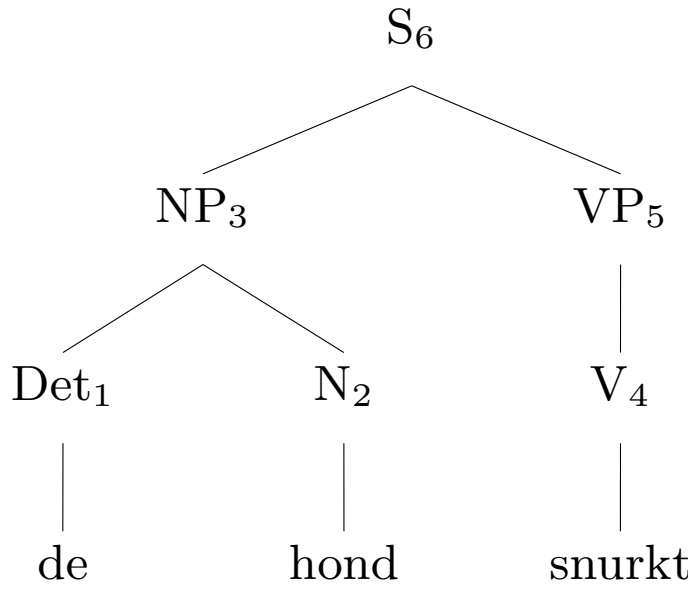
```
parse(Cat,[Word|Tail],Tail) :-
  lex(Cat,Word).
parse(Cat,P0,P) :-
  rule(Cat,Daughters),
  parse_daughters(Daughters,P0,P).

parse_daughters([],List,List).
parse_daughters([D1|Daughters],P0,P) :-
  parse(D1,P0,P1),
  parse_daughters(Daughters,P1,P).

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

Bottom-up ontleding

- Begin bij de input (woorden),
- Bepaal de categorieën van de woorden,
- Combineer categorieën tot zinsdelen,
- Combineer zinsdelen tot grotere zinsdelen
- **Geen problemen met links-recursieve regels!**

Bottom-up ontleding

de hond snurkt
Det *hond snurkt*
Det N *snurkt*
NP *snurkt*
NP V
NP VP
S



Shift-reduce algoritme

- **Stack**: Een stapel van reeds gevonden zinsdelen,
- **Shift**: Verwijder het meest linkse element W van de input, en zet de catogorie C van W op de stack.
- **Reduce**: Vervang $C_1..C_n$ op de stack door C_0 mits er een regel $C_0 \rightarrow C_1..C_n$ bestaat.

Shift-reduce Algoritme

stap	invoer	stapel	actie
1.	*de hond snurkt*	[ ]	*shift*
2.	*hond snurkt*	[Det]	*shift*
3.	*snurkt*	[Det N]	*reduce*
4.	*snurkt*	[NP]	*shift*
5.		[NP V]	*reduce*
6.		[NP VP]	*reduce*
7.		[S]	

Shift-reduce in Prolog

```
%% sr(Input,Stack0,Stack)
sr([],L,L).
sr(Input,Stack0,Stack) :-
    reduce(Stack0,Stack1),
    sr(Input,Stack1,Stack).
sr(Input,Stack0,Stack) :-
    shift(Input,Rest,Cat),
    sr(Rest,[Cat|Stack0],Stack).

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

Shift/reduce

```
shift([Wrd|Input],Input,Cat) :-
    lex(Cat,Wrd).

%% reduce([vp,np|Stack],[s|Stack]).  % s -> np, vp
%% reduce([n,det|Stack],[np|Stack]). % np -> det, n

reduce(Stack0,[Cat|Rest]) :-
  rule(Cat,Dtrs),
  reverse(Dtrs,RDtrs),
  append(RDtrs,Rest,Stack0).
```

Voordelen van shift-reduce algoritme

- Links-recursie is geen probleem.
- Omvang van de stack is nooit groter dan aantal woorden in de input.
- Parsing termineert zolang de grammatica geen cyclische regels bevat.

$$\begin{array}{|l} \hline np \rightarrow n \\ n \rightarrow np \\ \hline \end{array}$$

Maar . . .

$$\boxed{\text{det} \rightarrow \epsilon}$$

- Epsilon-regels voegen een categorie toe aan de stack, zonder dat dit correspondeert met een element van de input;
- omvang van de stack is niet langer kleiner of gelijk aan de input;
- epsilon-regels kunnen ertoe leiden dat de parsing niet termineert.

SR en Epsilon's

		String	Stack	actie	regel
1	honden	blaffen	[]	sh	
2	honden	blaffen	[det]	sh	
3	honden	blaffen	[det,det]	sh	
...					

Eliminatie van Epsilons

- Het effect van epsilon-regels kan ook altijd door regels zonder epsilon bereikt worden,
- Iedere grammatica met epsilons heeft een equivalente grammatica zonder epsilons.

$$\frac{\mathsf{np} \rightarrow \mathsf{det}\ \mathsf{n} \qquad\qquad \mathsf{det} \rightarrow \epsilon}{\mathsf{np} \rightarrow \mathsf{n}}$$

Eliminatie van Epsilons

Voor alle regels $C \rightarrow \epsilon$ en
alle regels $M \rightarrow C_1 ... C_i, \mathbf{C}, C_j ... C_n,$
voeg toe $M \rightarrow C_1 ... C_i, C_j ... C_n.$

Eliminatie van Unaire regels

- Vergelijkbare constructie

Eliminatie van Epsilons en Unaire regels

- Kan altijd voor CFG
- Voor DCG lastiger:
  - `x(A) --> x(s(A)).`
  - `y(s(A)) --> y(A).`

Parse-strategieën

- Prolog (DCG) zoekt **top-down**, **depth-first**, **links-rechts**
- Alternatieven:
  - **Bottom-up**: Begin bij de input (woorden), werk naar de startcategorie (S) toe
  - **Breadth-first**: Onderzoek alle mogelijke manieren om zinsdelen te vormen parallel
  - **rechts-links**, **hoofd-gedreven**
- Alternatieve methode's zijn (vaak) robuuster en/of efficiënter

Zoeken

- Prolog (en onze parsers tot nu toe): depth-first met backtracking
- Alternatief: breadth-first met chart (datastructuur met alle gevonden deelresultaten)

- Depth-first met backtracking
  - vaak inefficiënt (rekentijd)
  - vaak zuinig in geheugengebruik
- Breadth-first met chart
  - vaak efficiënt (rekentijd)
  - vaak enorm geheugengebruik

Depth-first zoeken

- Als er een **keuze** is (tussen regels, tussen shift en reduce acties);
- onderzoek je 1 mogelijkheid volledig (**depth-first**);
- en andere mogelijkheden pas als de eerste keuze faalt;
- Maakt gebruik van **back-tracking**.

Depth-first parsing

- DCG: **top-down**, depth-first, link-rechts
- Shift-reduce: **bottom-up**, depth-first, links-rechts

Nadelen van backtracking

Marie ziet de jongen *met de hond*

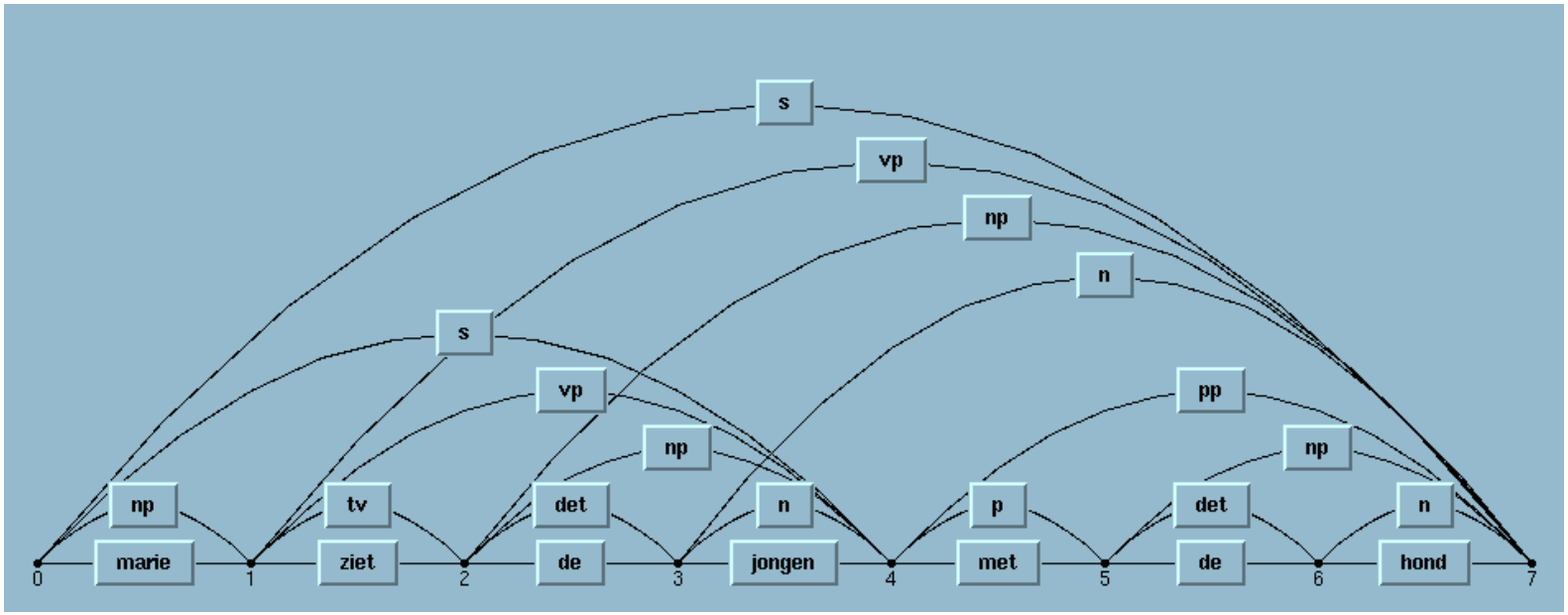
Ander nadeel: ongrammaticale input

- Als de zin niet correct is, heb je niets
- Liever: geef betekenisvolle delen terug
- Chart

Chart Parsing

- Bereken nooit iets twee keer!
- Bewaar gedeeltelijke resultaten in een tabel (chart),
  - B.v. een lijst van constituenten die zijn gevonden in de input (*well-formed substring table*).
- Is toepasbaar voor depth-first en breadth-first

Een chart


s
vp
np
n
s
vp
np
pp
np
np
tv
det
n
p
det
n
marie
ziet
de
jongen
met
de
hond
0
1
2
3
4
5
6
7

Een chart als een lijst

0 Marie 1 ziet 2 de 3 jongen 4 met 5 de 6 hond 7

$\langle 0,1,\text{np}\rangle$	$\langle 1,2,\text{v}\rangle$	$\langle 2,3,\text{det}\rangle$
$\langle 3,4,\text{n}\rangle$	$\langle 2,4,\text{np}\rangle$	$\langle 1,4,\text{vp}\rangle$
$\langle 0,4,\text{s}\rangle$	$\langle 4,5,\text{p}\rangle$	$\langle 5,6,\text{det}\rangle$
$\langle 6,7,\text{n}\rangle$	$\langle 5,7,\text{np}\rangle$	$\langle 4,7,\text{pp}\rangle$
$\langle 1,7,\text{vp}\rangle$	$\langle 3,7,\text{np}\rangle$	$\langle 0,7,\text{s}\rangle$

Chart Parsing

- **Doel**: Vind alle constituenten in de input,
- **Succes**: Er is een constituent in de chart die de hele input beslaat en van categorie S is.

Chart Parsing

- SCANNING
$$\frac{\text{lex}(C,\text{Wrd}),\ \text{Wrd op positie } i+1}{\langle i, i+1, C\rangle}$$
- COMPLETION
$$\frac{C \rightarrow C_1\, C_2,\ \langle i_0, i_i, C_1\rangle \quad \langle i_i, i_n, C_2\rangle}{\langle i_0, i_n, C\rangle}$$

Chart Parser

- Gebruik *chart* met alle gevonden items
- Gebruik *agenda* met alle recent toegevoegde items
- Voor elk item in de agenda bepaal je of er nieuwe items kunnen worden toegevoegd
- `item(Cat,P0,P,HisList)`

Prolog

```
parse(Sent) :-
    add_words_to_agenda(Sent,0,P,Agenda),
    process_agenda(Agenda),
    extract_tree(s,0,P,Tree).

process_agenda([]).

process_agenda([Item|Agenda]) :-
    chart_member_check(Item),!,
    process_agenda(Agenda).

process_agenda([Item|Agenda0]) :-
    chart_add(Item),
    findall(NewItem,completion(Item,NewItem),Agenda1),
    append(Agenda0, Agenda1, Agenda),
    process_agenda(Agenda).
```

Completion

Alleen regels met 1 of 2 dochters

```
completion(item(D2,P1,P),item(Cat,P0,P)) :-
   rule(Cat,[D1,D2]),
   chart_member(item(D1,P0,P1)).

completion(item(D1,P0,P1),item(Cat,P0,P)) :-
   rule(Cat,[D1,D2]),
   chart_member(item(D2,P1,P)).

completion(item(D,P0,P), item(Cat,P0,P)) :-
   rule(Cat,[D]).
```

Geschiedenis bijhouden

Deze parser is nog niet ideaal, we houden namelijk niet bij hoe een consituent geconstrueerd is. Bijvoorbeeld in

0 Marie 1 ziet 2 de 3 jongen 4 met 5 de 6 hond 7

kan `item(vp,1,7)` geconstueerd zijn met:

- `item(vp,1,4)` en `item(pp,4,7)`
- `item(v,1,2)` en `item(np,2,7)`

Geschiedenis bijhouden (2)

We gaan in een item, `item(Cat,P0,P,HisList)`, een lijstje bijhouden van hoe het geconstrueerd is. Bijvoorbeeld:

```
item(vp,1,7,[r(vp,4,pp),r(v,2,np)])
```

Prolog

```
parse(Sent) :-
    add_words_to_agenda(Sent,0,P,Agenda),
    process_agenda(Agenda),
    extract_tree(s,0,P,Tree).

process_agenda([]).

process_agenda([Item|Agenda]) :-
    chart_member_check(Item),!,
    process_agenda(Agenda).

process_agenda([Item|Agenda0]) :-
    chart_add(Item),
    findall(NewItem,completion(Item,NewItem),Agenda1),
    append(Agenda0, Agenda1, Agenda),
    process_agenda(Agenda).
```

Completion

Alleen regels met 1 of 2 dochters

```
completion(item(D2,P1,P,_),item(Cat,P0,P,[r(D1,P1,D2)])) :-
   rule(Cat,[D1,D2]),
   chart_member(item(D1,P0,P1,_)).

completion(item(D1,P0,P1,_),item(Cat,P0,P,[r(D1,P1,D2)])) :-
   rule(Cat,[D1,D2]),
   chart_member(item(D2,P1,P,_)).

completion(item(D,P0,P,_), item(Cat,P0,P,[r(D)])) :-
   rule(Cat,[D]).
```

Terugvinden van parsebomen

- Samenpakken van geschiedenis van items

```
chart_member_check(item(Cat,P0,P,His0)) :-
    chart_delete(item(Cat,P0,P,His1)),
    append(His0,His1,His),
    chart_add(item(Cat,P0,P,His)).
```

Terugvinden van parsebomen

- Terugvinden van parseboom:

```
extract_tree(Cat,P0,P,b(Cat,Ds)) :-
    chart_item(Cat,P0,P,His),
    member(Single,His), % kies eentje
    extract_trees(Single,P0,P,Ds).

extract_trees(r(D),P0,P,[Tree]) :-
    extract_tree(D,P0,P,Tree).

extract_trees(r(D1,P1,D2),P0,P,[Tree1,Tree2]):-
    extract_tree(D1,P0,P1,Tree1),
    extract_tree(D2,P1,P, Tree2).

extract_trees(lex(Word),[w(Word)]).
```

Chart

- Resultaat van parsing: chart
- Bevat (impliciet) alle parses voor de input
- Elk item kan extra informatie bevatten:
  - Welke regel voor dit item is gebruikt
  - Welke items voor elk van de dochters van de regel is gebruikt
- Compacte representatie van alle parse bomen

Partial parsing

- Vind alle NPs in de invoer,
- Geen volledige parse (grammatica) nodig.
- Gemakkelijk met een chart,
- Moeilijk (en langzaam) met een shift-reduce parser.

Partial Parsing Experiment

	Runtime (ms)	
Zinnen	Shift-Reduce	Chart
5	804	5
10	1229	12
20	41295	26