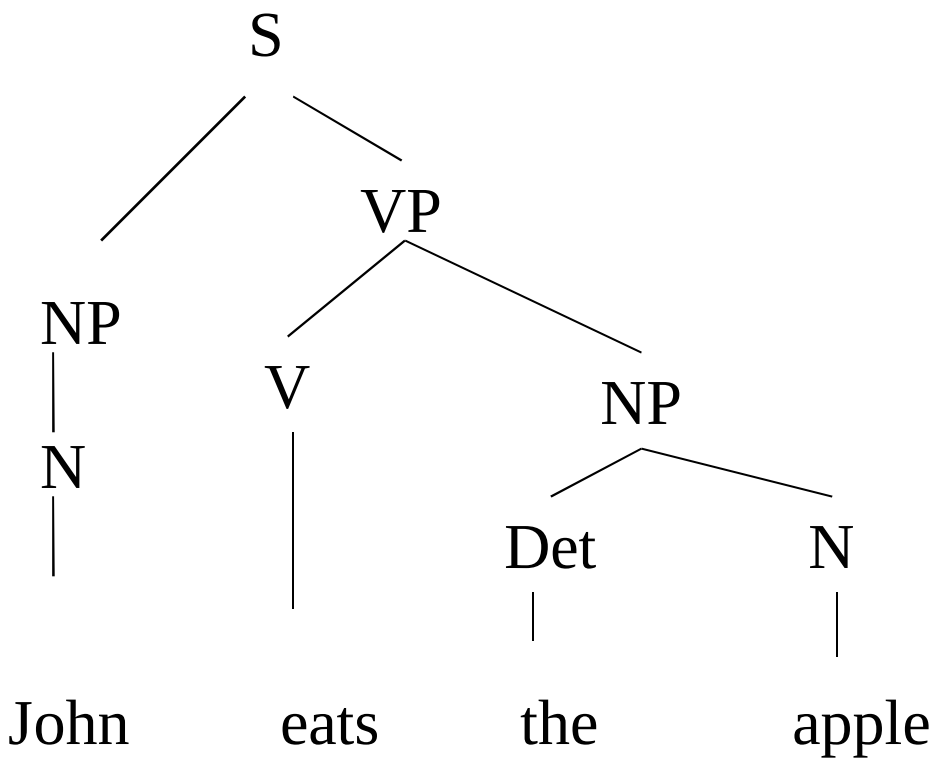
Grammar development

- Grammar rules
- context free/context sensitive
- Prolog grammar : parser
- Grammar formalism in Prolog:
  Definite Clause Grammars (DCGs)

Grammar as a production system

John eats the apple

Syntax tree:



Grammar as a production system

Rules:

S $\rightarrow$ NP VP

NP $\rightarrow$ Det N

NP $\rightarrow$ N

VP $\rightarrow$ V NP

VP $\rightarrow$ V

Grammar as a production system

Lexicon:

N → mary	V → loves	Det → the
N → john	V → eats	
N → man	V → sings	
N → apple		

Grammar as a production system

Rules:

S $\rightarrow$ NP VP

NP $\rightarrow$ Det N

NP $\rightarrow$ N

VP $\rightarrow$ V NP

VP $\rightarrow$ V

N $\rightarrow$ apple, john,..

V $\rightarrow$ loves, eats,..

Det $\rightarrow$ the

context free grammar!

Context free grammar

- Set of *terminal symbols (T)*
- Set of *non-terminal symbols (NT)*
- A distinguished NT symbol: *start symbol (S)*
- Set of *(production-)rules*

  of the form:

$$NT \rightarrow \alpha, \text{ with } \alpha \in (NT \cup T)^*$$

$$NT \rightarrow T$$

Context sensitive grammar

- Set of *terminal symbols (T)*
- Set of *non-terminal symbols (NT)*
- A distinguished NT symbol: *start symbol (S)*
- Set of *(production) rules*
  of the form:

$$\alpha A \gamma \rightarrow \alpha \beta \gamma$$

A from NT,
α, γ, β from $(NT \cup T)^*$

Parsing

- ?- s([john,eats,the,apple]).

  yes
- ?- s ([john, the,apple,eats]).

  no
- ?- s ([the, eats,john,apple]).

  no
- ?- s ([the,apple,eats,john]).

  ??

Derivations

Rules:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ Det N
3. NP $\rightarrow$ N
4. VP $\rightarrow$ V NP
5. VP $\rightarrow$ V

Derivations

(1) (3) (4) (2)

S $\Rightarrow$ NP VP

$\Rightarrow$ N VP

$\Rightarrow$ N V NP

$\Rightarrow$ N V Det N

(terminal) lexicon rules:

John eats the apple

Derivations: direction ?

(1) (3) (4) (2)

S $\Rightarrow$ NP VP

$\Rightarrow$ N VP

$\Rightarrow$ N V NP

$\Rightarrow$ N V Det N

(terminal) Lexicon rules:

John eats the apple

Generation or Analysis?

Generation

- s(S) :- np(NP), vp(VP), append(NP,VP,S).
- np(NP) :- det(Det), n(N), append(Det,N,NP).
- np(NP) :- n(N), NP=N.
- vp(VP) :- v(V), np(NP), append(V,NP,VP).
- vp(VP) :- v(V), VP=V.

Lexicon

- n([mary]).
- n([apple]).
- n([john]).
- n([man]).
- v([loves]).
- v([eats]).
- v([sings]).
- det([the]).

Generation

- ?- s(S)
- X = [the,mary,loves,the,mary] ;
- X = [the,mary,loves,the,apple] ;
- ...
- X = [john,eats,the,apple] ;
- ...

Parsing?

?- s([john,eats,the,apple]).

- s(S) :- np(NP), vp(VP), append(NP,VP,S).
- np(NP) :- det(Det), n(N), append(Det,N,NP).
- np(NP) :- n(N), NP=N.
- vp(VP) :- v(V), np(NP), append(V,NP,VP).
- vp(VP) :- v(V), VP=V.

Parsing

?- s([john,eats,the,apple]).

- s(S) :- append(NP,VP,S), np(NP), vp(VP).
- np(NP) :- append(Det,N,NP), det(Det), n(N).
- np(NP) :- NP=N, n(N).
- vp(VP) :- append(V,NP,VP), v(V), np(NP).
- vp(VP) :- VP=V, v(V).

Is parsing efficient ?

?- s([john,eats,the,apple]).

- s(S) :- append(NP,VP,S), np(NP), vp(VP).
- np(NP) :- append(Det,N,NP), det(Det), n(N).
- np(NP) :- NP=N, n(N).
- vp(VP) :- append(V,NP,VP), v(V), np(NP).
- vp(VP) :- VP=V, v(V).

trace !!

Is parsing efficient ?

No!

Use two lists instead:

- Input
- Part of the input that remains after cutting off the part recognized by the rule:

Difference lists !

```
s([john,eats,the,apple],[]) :-
        np([john,eats,the,apple],[eats,the,apple]),
        vp([eats,the,apple,[]).
```

More efficient parsing

Schematically: ..

- s(NPVP,[]) :- np(NPVP,VP), vp(VP, []).
- np(NPVP,VP) :- det(NPVP,NVP), n(NVP, VP).
- np(NVP,VP) :- n(NVP, VP).
- vp(VP,[]) :- v(VP,NP), np(NP, []).
- vp(VP, []) :- v(VP, []).

More efficient parsing

Schematically, ..more generally!

- s(SR,R) :- np(SR,VPR), vp(VPR,R).
- np(DetNPR,R) :- det(DetNPR,NR), n(NR,R).
- np(NR,R) :- n(NR,R).
- vp(VNPR,R) :- v(VNPR,NPR), np(NPR,R).
- vp(VR) :- v(VR,R).

More efficient parsing

Lexicon

- n([mary|R],R).
- n([apple|R],R).
- n([john|R],R).
- ...
- v([loves|R],R).
- v([eats|R],R).
- ...
- det([the|R],R).
- ...

More efficient parsing **and** Generation

- ?- s([john,eats,the,apple],[]).
  yes
- ?- s(X,[]).
  X = [the,mary,loves,the,mary] ;
  ...
- ?- np([the,apple],[]).
  yes
- ...

A friendlier type of grammar rule writing:

- Definite clause grammars
- A formalism implemented in prolog

A DCG notation for our grammar:

Rules: omit lists (they are added in compilation to Prolog)

s **-->** np, vp.

np **-->** det, n.

np --> n.

vp --> v, np.

vp --> v.

A DCG notation for our grammar :

Lexicon: name word only (rest is added automatically)

```
n --> [mary].
n --> [apple].
...
v --> [loves].
v --> [eats].
...
det --> [the].
..
```

A DCG notation for our grammar :

DCG rule operator is recognized: therefore simply *consult* DCG

listing shows that internally difference lists are used.

Grammars/DCGs

Question: Are derivations unique?

- In our grammar?
- In an extended grammar, with:
- PP $\rightarrow$ P NP
- NP $\rightarrow$ NP PP
- VP $\rightarrow$ VP PP
- N $\rightarrow$ {worm, knife}
- P $\rightarrow$ {with, in}

  cf: *John eats the apple with the worm with the knife*.

Grammars/DCGs

How can the derivations/derivation trees be made visible ?

What about constraints like Subj-Verb agreement?