T-(538|725)-MALV, Natural Language Processing
Context-free grammar and Prolog

Hrafn Loftsson[1] Hannes Högni Vilhjálmsson[1]

[1]School of Computer Science, Reykjavik University

October 2008

Outline

Outline

Prolog (1970)

Material to read

- `http://en.wikipedia.org/wiki/Prolog`
- Appendix A in the textbook.

Design issues

- To support Natural Language Processing.
- Declarative programming
- Built in search and unification mechanism
  - (`http://en.wikipedia.org/wiki/Declarative_programming`

Prolog (1970)

Material to read

- `http://en.wikipedia.org/wiki/Prolog`
- Appendix A in the textbook.

Design issues

- To support Natural Language Processing.
- Declarative programming
- Built in search and unification mechanism
  - (`http://en.wikipedia.org/wiki/Declarative_programming`

Outline

Definite Clause Grammar (DCG)

- A feature of Prolog.
- Enables us to write context-free grammmar (CFG) rules directly in a Prolog program.
- Prolog converts the CFG rules to Prolog clauses.
- The prolog engine (algorithm) automatically carries out the parse without the need for additional programming.
- Main advantage of Prolog with regard to CFG: Few lines of code!

Definite Clause Grammar (DCG)

An example

```
Rules                    Lexicon
-------------------------------------------------------
s --> np, vp.            det --> [the].     verb --> [brought].
np --> det, noun.        det --> [a].       prep --> [to].
np --> np, pp.           noun --> [waiter]. prep --> [of].
vp --> verb, np.         noun --> [meal].
vp --> verb, np, pp.     noun --> [table].
pp --> prep, np.         noun --> [day].
```

Usually, the operator *:-* is used as a separator between the left-hand side and the right-hand side of a Prolog clause. In the case of DCG, `-->` is used.

Definite Clause Grammar (DCG)

Prolog search

- The Prolog search mechanism checks whether a fact is true or generates all the solutions.
- In the case of DCG, Prolog checks if a given sentence can be derived from the grammar or generates all the sentences accepted by the grammar.

Test

- Start Prolog: `"c:\Program Files\pl\bin\plwin"`
- Load af file (program): ?- consult('cfg.pro').

Definite Clause Grammar (DCG)

Prolog search

- The Prolog search mechanism checks whether a fact is true or generates all the solutions.
- In the case of DCG, Prolog checks if a given sentence can be derived from the grammar or generates all the sentences accepted by the grammar.

Test

- Start Prolog: `"c:\Program Files\pl\bin\plwin"`
- Load af file (program): ?- consult('cfg.pro').

Definite Clause Grammar (DCG)

Is a given string in the language?

```
?- s([the, waiter, brought, the, meal, to , the, table], []).
Yes
?- s([the, waiter, brought, the, meal, of , the, day], []).
Yes
```

Generate all (grammatically correct) sentences

```
?- s(L, []).
L = [the, waiter, brought, the, waiter] ;
L = [the, waiter, brought, the, meal] ;
L = [the, waiter, brought, the, table] ;
...
```

Definite Clause Grammar (DCG)

Is a given string in the language?

```
?- s([the, waiter, brought, the, meal, to , the, table], []).
Yes
?- s([the, waiter, brought, the, meal, of , the, day], []).
Yes
```

Generate all (grammatically correct) sentences

```
?- s(L, []).
L = [the, waiter, brought, the, waiter] ;
L = [the, waiter, brought, the, meal] ;
L = [the, waiter, brought, the, table] ;
...
```

Translating DCGs into Prolog clauses

- The DCG rule s --> np, vp.
- . . . is translated to the clause:
- s(L) :- np(L1), vp(L2).
-
- Example: *the waiter brought the meal*
  - L matches [the, waiter, brought the meal]
  - L1 matches the noun phrase [the, waiter]
  - L2 matches the verb phrase [brought, the, meal]

Translating DCGs into Prolog clauses

To be precise this is implemented with *difference lists*

- The DCG s --> np, vp.
- . . . is translated to the clause:
- s(L1,L) :- np(L1,L2), vp(L2,L).
- The lexical DCG rule: det --> [the].
- . . . is translated into the fact:
- det([the | L], L).

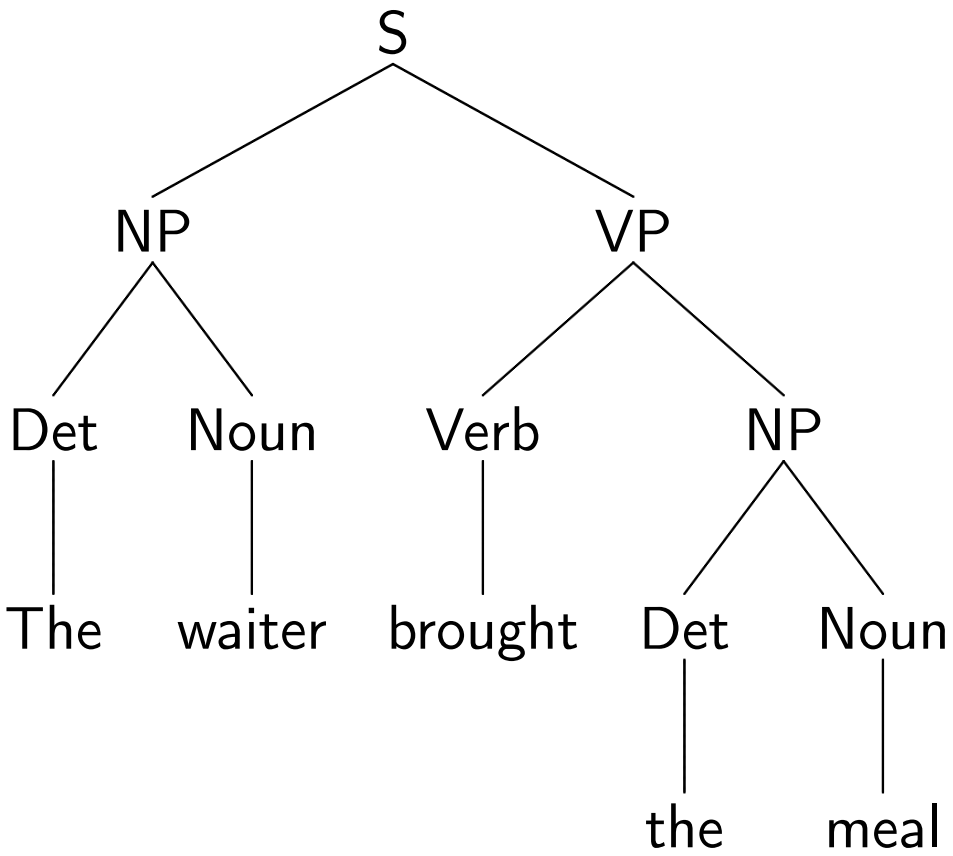
Difference lists

Constituent values correspond to the difference of each pair of arguments

- s(L1,L) :- np(L1,L2), vp(L2,L).
- *The waiter brought the meal* corresponds to the *s* symbol and unifies with L1\L, where L1\L,denotes L1 minus L.
- *The waiter* corresponds to the *np* symbol and unifies with L1\L2.
- *brought the meal* corresponds to the *vp* symbol and unifies with L2\L.
- In therms of lists
  - L1\L corresponds to [the, waiter brought, the meal]
  - L1\L2 corresponds to [the, waiter]
  - L2\L corresponds to [brought, the, meal]

Prolog peforms *depth-first search*

- ?- trace.
- ?- s([the, waiter, brought, the, meal], []).



Prolog performs *depth-first search*

- ?- s([the, waiter, brought, the, meal], []).
- s(L1, L) :- np(L1, L2), vp(L2, L).
  - L1 = [the, waiter, brought, the, meal], L=[].
- np(L1, L) :- det(L1, L2), noun(L2, L).
- det(the | L), L.
  - L = [waiter, brought, the, meal].
  - $\Rightarrow$ L1 = [the].
  - $\Rightarrow$ L2 = [waiter, brought, the, meal].
- noun([waiter, brought, the, meal],L).
- ...

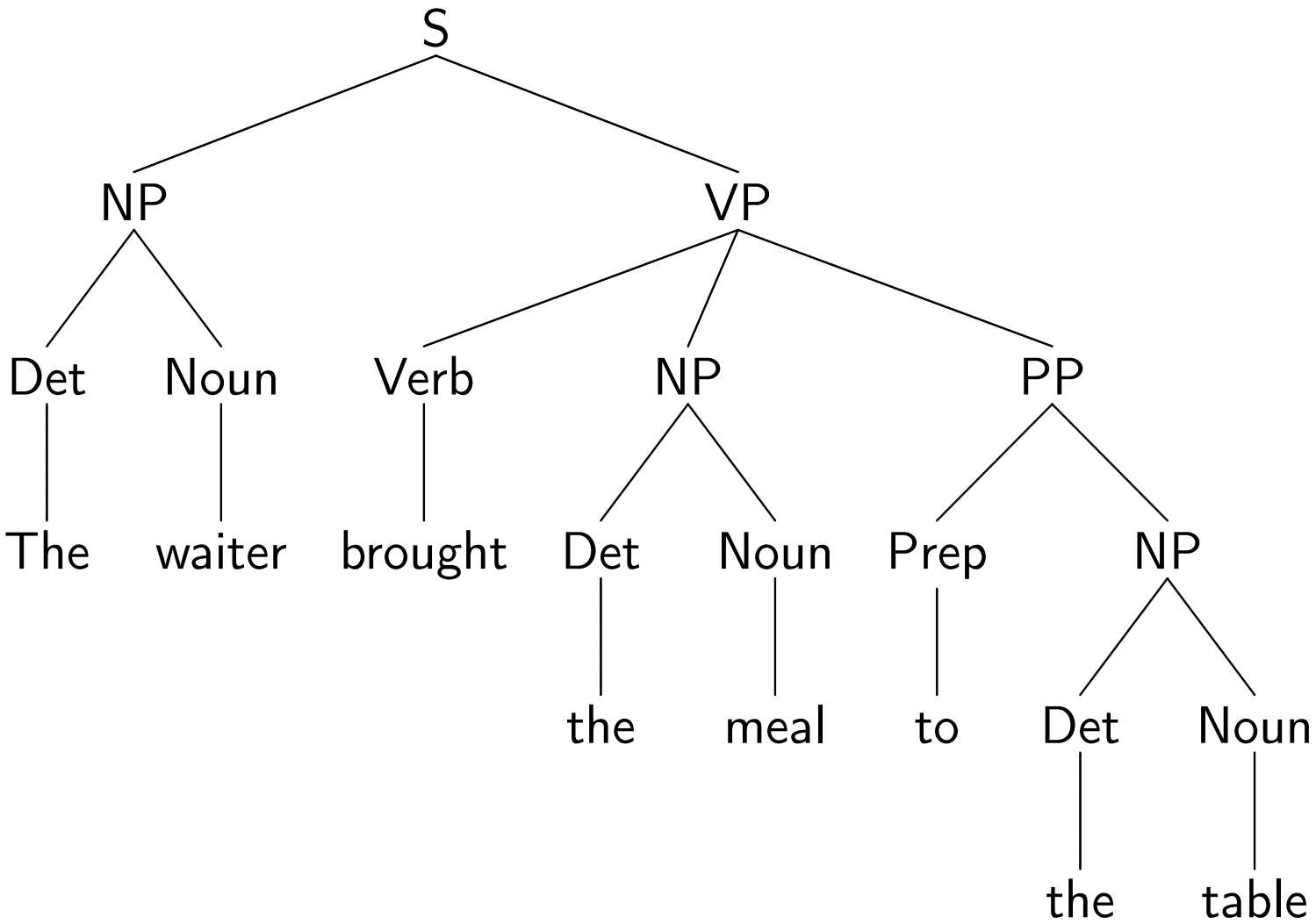
Left recursion (í. vinstri endurkvæmni)

- ?- s([the, brought, the, meal], []).
  ERROR: Out of local stack
- Due to the left-recursive rule: np --> np, pp.
- Left-recursiveness can be eliminated from a grammar (see Section 8.3.3 in the textbook).

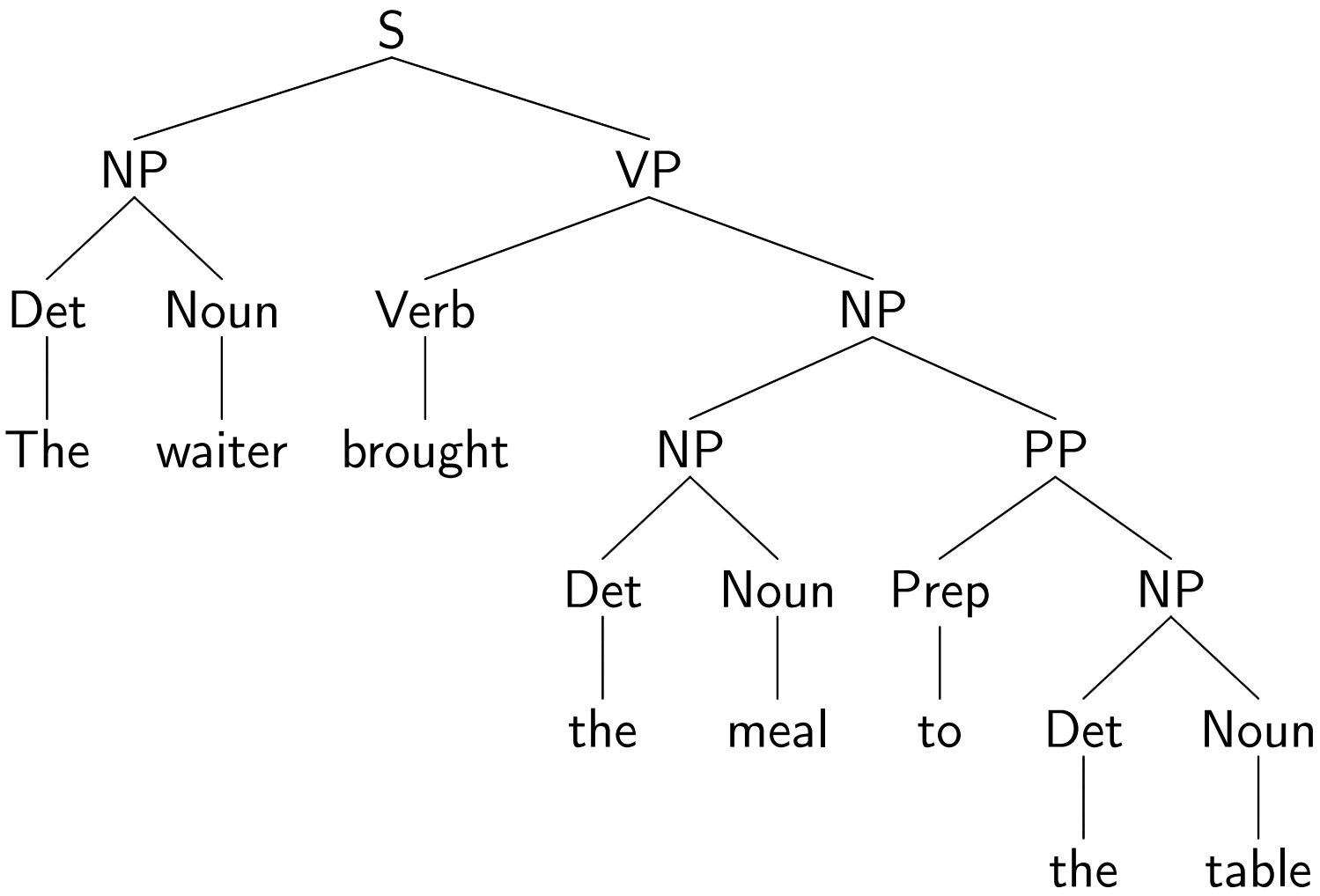
Parsing ambiguity (í. margræðni í þáttun)

- The rules . . .
- vp --> verb, np and np --> np, pp
- vp --> verb, np, pp
- . . . lead to ambiguity, because more than one parse tree is possible for a given string.
- For example the string: *The waiter brought the meal to the table*
- Tree 1a and 2a in the following slides are correct.

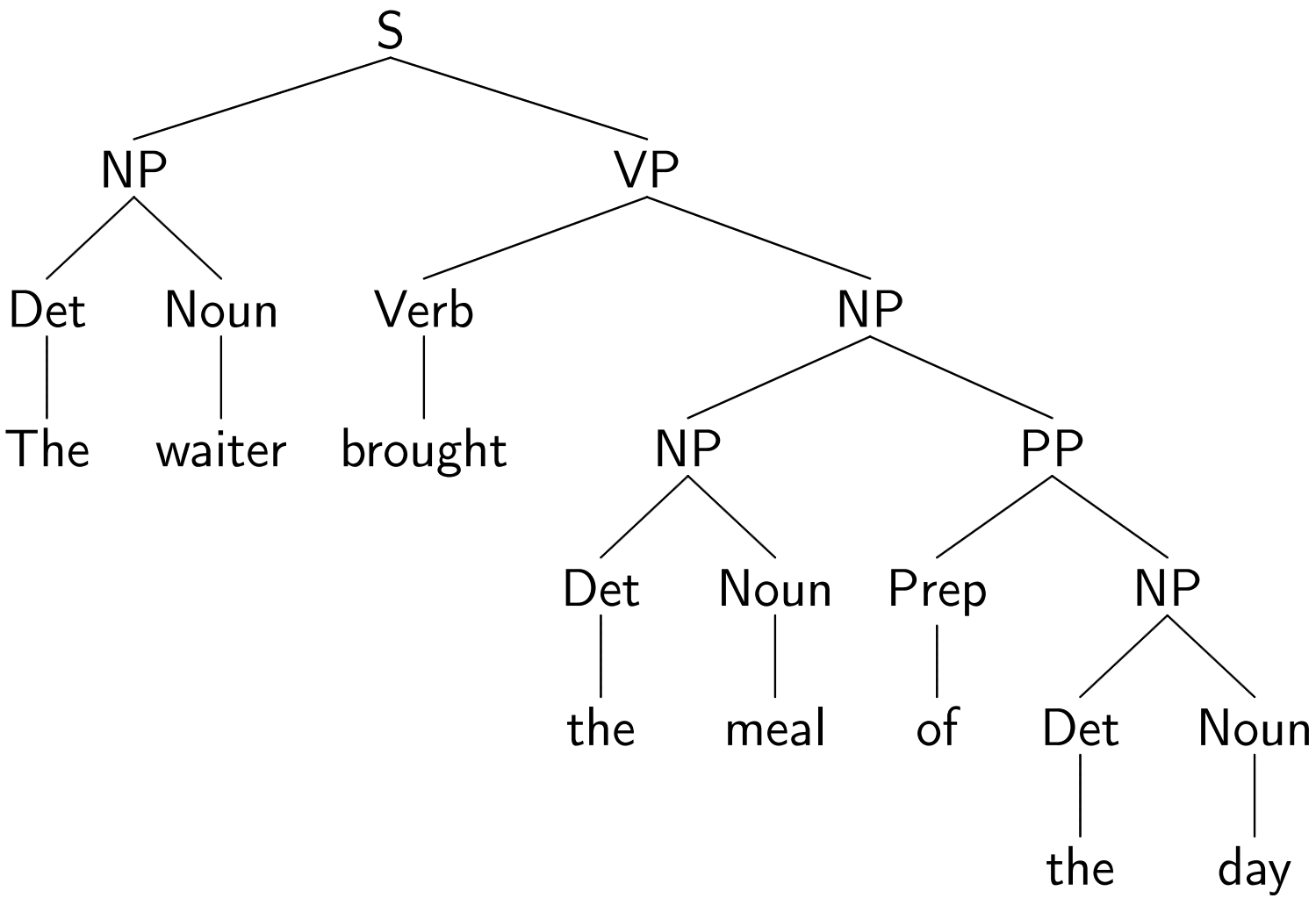
Parsing ambiguity – Tree 1a



Parsing ambiguity – Tree 1b



Parsing ambiguity – Tree 2a



Parsing ambiguity – Tree 2b