T-(538|725)-MALV, Natural Language Processing
Parsing techniques

Hrafn Loftsson[1] Hannes Högni Vilhjálmsson[1]

[1]School of Computer Science, Reykjavik University

October 2009

Outline

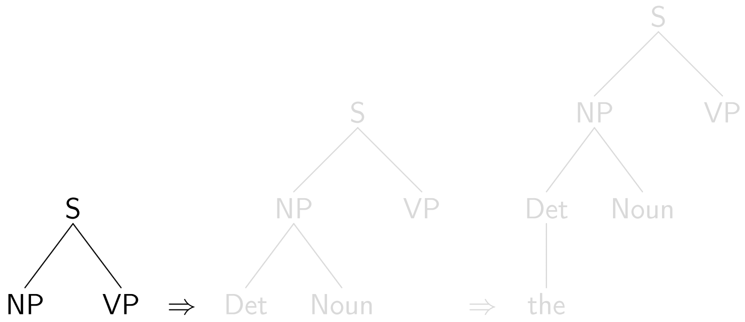
Outline

Top-down parsing (í. ofansækin þáttun)

- Constructs the parse tree starting at the root down to the leaves.
- We begin at the start symbol, S.
- All subtrees constructed having S to the left of the arrow: $S \rightarrow X$
- This is continued at the next subtree (node), X.
  - Rules found having X to the left of the arrow and all subtrees constructed for the right side.
- And so on until the leaves (words/tokens) are reached.
- Trees that don't match the input are discarded.

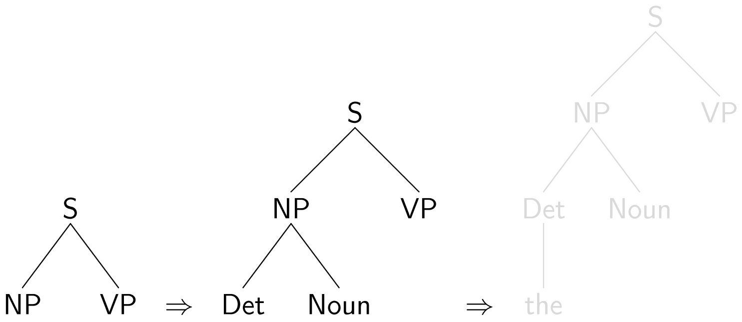
Top-down parsing: An example

The waiter brought the meal



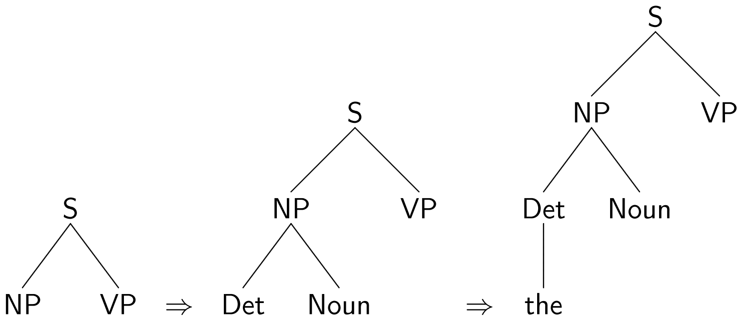
Top-down parsing: An example

The waiter brought the meal



Top-down parsing: An example

The waiter brought the meal



Top-down parsing

- For example, used in DCG (Definite Clause Grammar) in Prolog.
- Depth-first strategy:
  - Does not handle left-recursion, like:
  - np –> np, pp.
  - np –> np, conj, np.
- Backtracking:
  - Reparsing of the same constituents can happen over and over again.

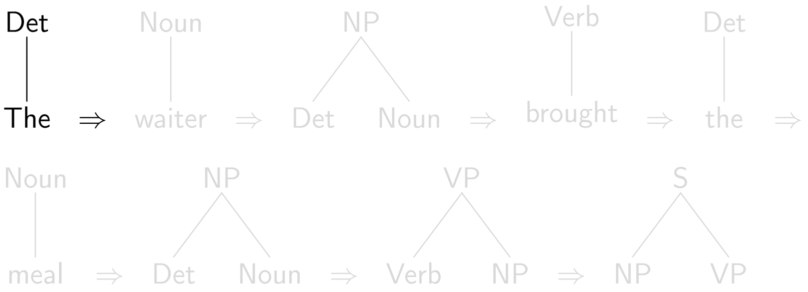
Outline

Bottom-up parsing (í. neðansækin þáttun)

- Constructs the parse tree starting at the leaves up to the root.
- Starts with the words/tokens.
- Checks if a word W appears to the right of an arrow in some rule: $X \rightarrow W$
- The right side symbol(s) is removed and the left side, X, added instead.
- And so on until the root is reached.
- Trees that do not lead to the root are discarded.

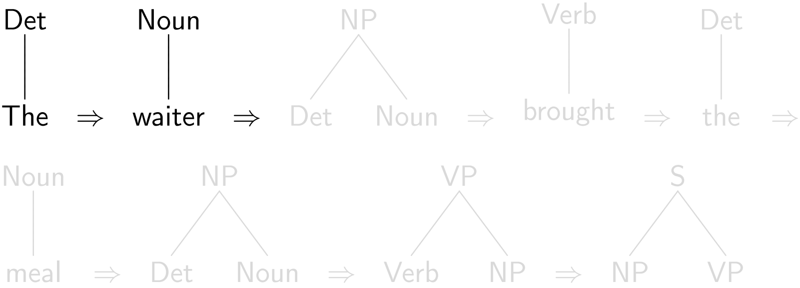
Bottom-up parsing: An example

The waiter brought the meal



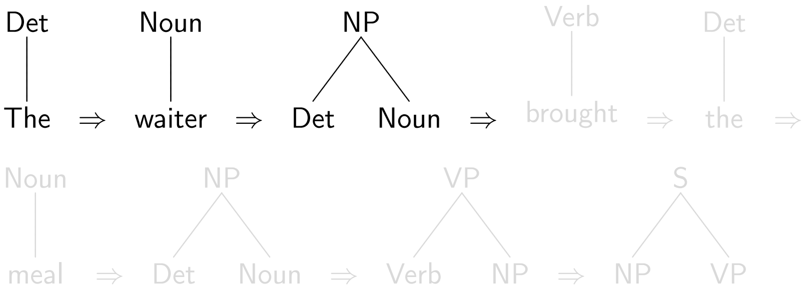
Bottom-up parsing: An example

The waiter brought the meal



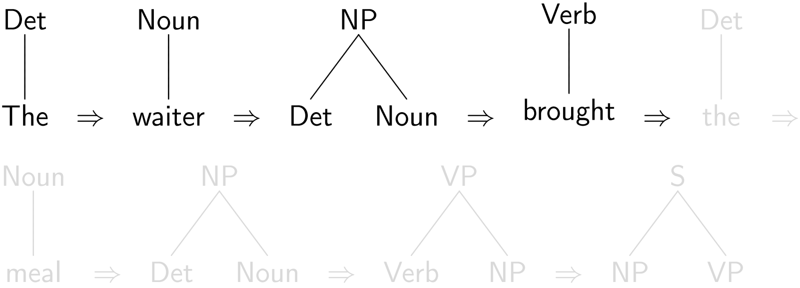
Bottom-up parsing: An example

The waiter brought the meal



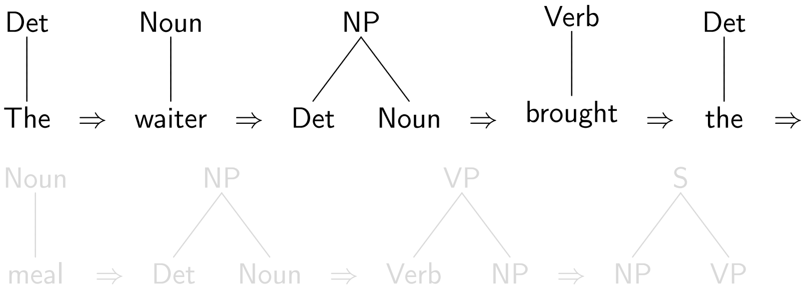
Bottom-up parsing: An example

The waiter brought the meal



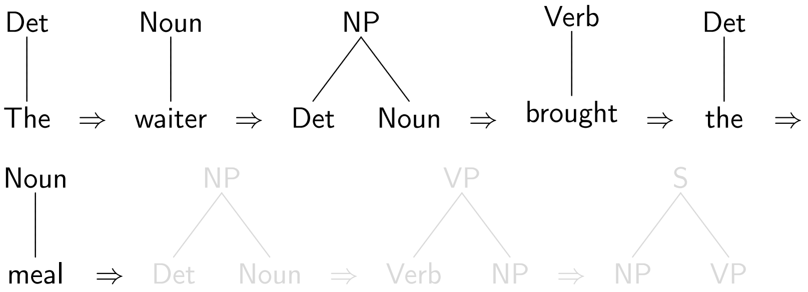
Bottom-up parsing: An example

The waiter brought the meal



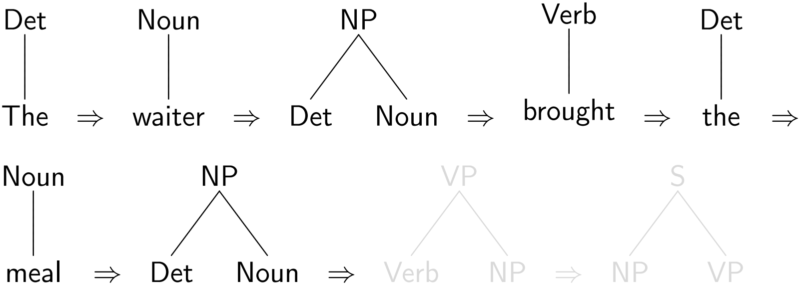
Bottom-up parsing: An example

The waiter brought the meal



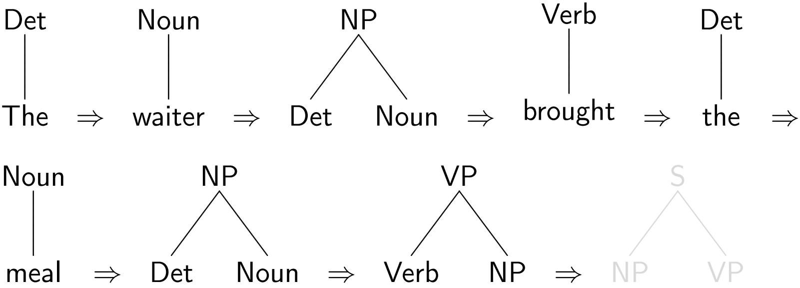
Bottom-up parsing: An example

The waiter brought the meal



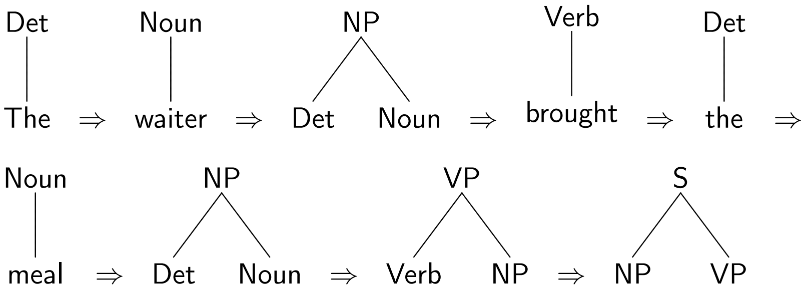
Bottom-up parsing: An example

The waiter brought the meal



Bottom-up parsing: An example

The waiter brought the meal



Shift-Reduce parsing

- A bottom-up parsing method:
  1. **Shifts** a word from the phrase or sentence to parse onto a stack.
  2. Applies a sequence of grammar rules to **reduce** elements of the stack.
- This loop repeated until there are no more words in the list and the stack is reduced to the parsing goal (the start symbol).

Shift-Reduce parsing: An example

It.	Stack	S/R	Word list
0		shift	the waiter brought the meal
1	the	reduce	waiter brought the meal
2	det	shift	waiter brought the meal
3	det waiter	reduce	brought the meal
4	det noun	reduce	brought the meal
5	np	shift	brought the meal
6	np brought	reduce	the meal
7	np verb	shift	the meal
8	np verb the	reduce	meal
9	np verb det	shift	meal
10	np verb det meal	reduce	
11	np verb det noun	reduce	
12	np verb np	reduce	
13	np vp	reduce	
14	s		

Top-down vs. bottom-up parsing

Top-down

- Does not spend time on parse trees which cannot end in S.
- Does not use the words to guide the parsing $\Rightarrow$ it leads to the expansion of trees that have no chance to yield any solution.
- Can handle null constituents (e.g. det –> [].)

Top-down vs. bottom-up parsing

Bottom-up

- Uses the words to guide the parsing.
- Does not spend time on trees which do not match the input words.
- Does not use S to guide the parsing $\Rightarrow$ it constructs subtrees even if they have no chance to result in a sentence (a full tree).
- Can handle left-recursion.

Outline

Reparsing of constituents

- Backtracking, a characteristic of top-down parsers, often leads to reparsing of constituents:
  - np –> npx.
  - np –> npx, pp.
  - npx –> det, noun.
  - pp –> prep, np.
- The reparsing of *npx* happens for a string like "the meal of the day"
- (Can be solved by the so-called *left factoring* (í. vinstri þáttun) but then the grammar needs to be modified; see the course *Compilers*)

Chart parsing (í. töfluþáttun)

What is it?

- A technique to avoid a parser repeating the same analysis.
- A **chart** is a data structure in which the parser stores all the possible partial results at a given position in the sentence.
- When a subsequent word is processed, the parser fetches partial parse structures obtained so far in the chart instead of reparsing them.
- At the end, the chart contains all possible parse trees and subtrees (see Fig. 11.7).

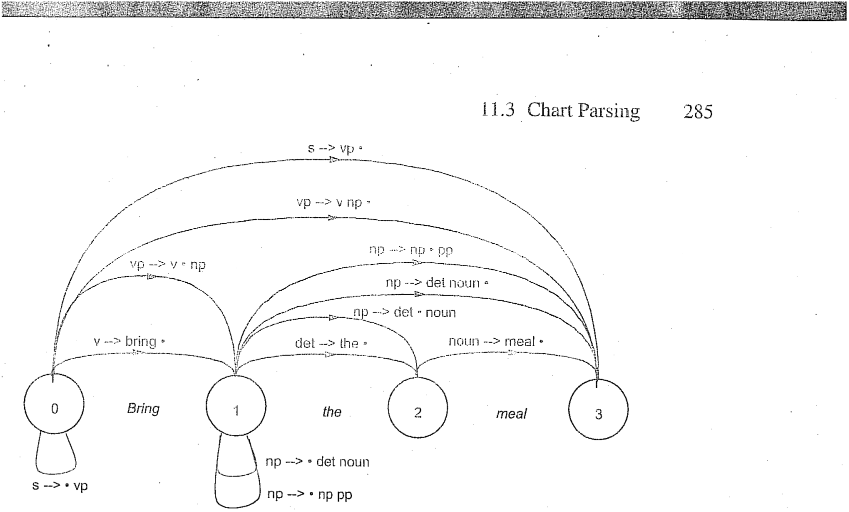
Chart parsing

```
s --> vp.          s --> np.            vp --> v, np, pp.
vp --> v, np.      np --> det, noun.    np --> det, adj, noun.
np --> np, pp.     pp --> prep, np.
```

A "dotted" rule

- Represents what has been parsed so far.
- np –> det noun • (inactive arc)
- np –> det • noun (active arc)
- np –> • det noun (active arc)

Dotted rules: An example



Dotted rules: An example

Rules	Arcs	Constituents
s –> • vp	[0,0]	• Bring the meal
vp –> v • np	[0,1]	Bring • the meal
np –> • det noun	[1,1]	• the meal
np –> • np pp	[1,1]	• the meal
np –> det • noun	[1,2]	the • meal
np –> det noun •	[1,3]	the meal •
np –> np • pp	[1,3]	the meal •
vp –> v np •	[0,3]	Bring the meal •
s –> vp •	[0,3]	Bring the meal •

The table does not show all possible rules.

Chart parsing

The Earley algorithm

- An efficient context-free parsing algorithm (1970)
  - http://portal.acm.org/citation.cfm?id=362035
- A top-down method.
- Can handle left-recursion and empty constituents.
- Uses three operations:
  - Predictor
  - Scanner
  - Completer

The Earley algorithm

Predictor

- Selects all possible further parses by selecting all the rules that can process active arcs.
- For a rule: lhs –> $c_1 c_2 \ldots \bullet c \ldots c_n$
- The predictor introduces new parsing goals: c –> $\bullet x_1 x_2 \ldots x_k$

Scanner

- Accepts a new word from the input.
- The PoS to the right of a dot are matched against the word.
- Puts the rule: pos –> word • into the chart.

The Earley algorithm

Completer

- Uses the new constituents generated by the Scanner to advance the dot af active arcs expecting them, and possibly complete the corresponding constituents.
- It first searches for rules having the dot and the end of a rule: c –> $x_1 x_2 \ldots x_n \bullet$
- Then it searches a rule like: lhs –> $c_1 c_2 \ldots \bullet\, c \ldots c_n$ and
- moves the dot over: lhs –> $c_1 c_2 \ldots c \bullet \ldots c_n$, and
- inserts the new arc into the chart.

The Earley algorithm: An example

Chart#	Rules	Arcs	Module	Constituents
0	s –> • np	[0,0]	Start state	• the meal of the day
0	np –> • det noun	[0,0]	Predictor	• the meal of the day
0	np –> • det adj noun	[0,0]	Predictor	• the meal of the day
0	np –> • np pp	[0,0]	Predictor	• the meal of the day
1	det –> the •	[0,1]	Scanner	• meal of the day
1	np –> det • noun	[0,1]	Completer	• meal of the day
1	np –> det • adj noun	[0,1]	Completer	• meal of the day
2	noun –> meal •	[1,2]	Scanner	• of the day
2	np –> det noun •	[0,2]	Completer	• of the day
2	np –> np • pp	[0,2]	Completer	• of the day
2	s –> np •	[0,2]	Completer	• of the day
2	pp –> • prep np	[2,2]	Predictor	• of the day

The Earley algorithm: An example (cont.)

Chart#	Rules	Arcs	Module	Constituents
3	prep –> of •	[2,3]	Scanner	• the day
3	pp –> prep • np	[2,3]	Completer	• the day
3	np –> • det noun	[3,3]	Predictor	• the day
3	np –> • det adj noun	[3,3]	Predictor	• the day
3	np –> • np pp	[3,3]	Predictor	• the day
4	det –> the •	[3,4]	Scanner	• day
4	np –> det • noun	[3,4]	Completer	• day
4	np –> det • adj noun	[3,4]	Completer	• day
5	noun –> day •	[4,5]	Scanner	
5	np –> det noun •	[3,5]	Completer	
5	pp –> prep np •	[2,5]	Completer	
5	np –> np pp •	[0,5]	Completer	
5	s –> np •	[0,5]	Completer	

Outline

Probabilistic parsing (í. tölfræðileg þáttun)

- Sometimes, we may want to find the most likely parse tree ...
- ... instead of generating all possible trees.
- This is possible if a treebank exists for the language.
  - A treebank is a syntactically annotated corpus.

PCFG

- PCFG – **P**robabilistic **C**ontext **F**ree **G**rammar (Collins 1996; Charniak 1997)
- A CFG where each rule is augmented with its probability $P(lhs \rightarrow rhs|lhs)$
- The probabilities are derived from a treebank.

Probabilistic parsing

- The probabilities are estimated with maximum likelihoods:

$$P(lhs \rightarrow rhs_i | lhs) = \frac{Count(lhs \rightarrow rhs_i)}{\Sigma_j Count(lhs \rightarrow rhs_j)}$$

- The probability for a sentence S to have the parse tree T is defined as the product of the probabilities attached to rules used to produce the tree:

$$P(T, S) = \prod_{rule(i) Producing T} P(rule(i))$$

- See the calculation of probabilities for two parse trees on page 296.