Language Technology
http://cs.lth.se/edan20/
Chapter 12: Constituent Parsing

Pierre Nugues

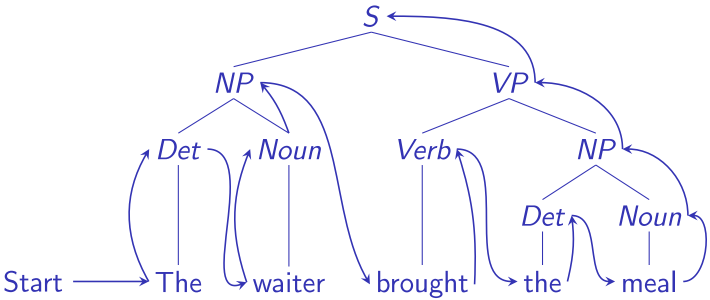
Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

September 26, 2019

Parsing

Possible parsing strategies are top-down or bottom-up
Prolog uses a top-down exploration and backtracks in case of error
Ambiguity can produce two or more possible parse trees
It is necessary to use probabilistic or symbolic techniques to rank parse trees

Bottom-up Parsing



Shift and Reduce

The shift and reduce algorithm implements bottom-up parsing.
Two input arguments: the list of words to parse and the parsing goal.
The algorithm gradually reduces words, parts of speech, and phrases until it reaches the parsing goal.
The algorithm consists of a loop of two steps:

- **Shift** a word from the phrase or sentence to parse onto a stack;
- Apply a sequence of grammar rules to **reduce** elements of the stack

until there is no more word in the list and the stack is reduced to the parsing goal.

DCG Rules

Nonterminal symbols

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

Terminal symbols

```
det --> [the].
det --> [a].
noun --> [waiter].
noun --> [meal].
noun --> [table].
noun --> [day].
verb --> [brought].
prep --> [to].
prep --> [of].
```

Shift and Reduce in Action

It.	Stack	S/R	Word list
0			[the, waiter, brought, the, meal]
1	[the]	Shift	[waiter, brought, the, meal]
2	[det]	Reduce	[waiter, brought, the, meal]
3	[det, waiter]	Shift	[brought, the, meal]
4	[det, noun]	Reduce	[brought, the, meal]
5	[np]	Reduce	[brought, the, meal]
6	[np, brought]	Shift	[the, meal]
7	[np, v]	Reduce	[the, meal]
8	[np, v, the]	Shift	[meal]
9	[np, v, det]	Reduce	[meal]
10	[np, v, det, meal]	Shift	[]
11	[np, v, det, n]	Reduce	[]
12	[np, v, np]	Reduce	[]
13	[np, vp]	Reduce	[]
14	[s]	Reduce	[]

Backtracking May be Inefficient

Example:

The meal of the day

```
np --> npx. npx --> det, noun.
np --> npx, pp.

pp --> prep, np.
```

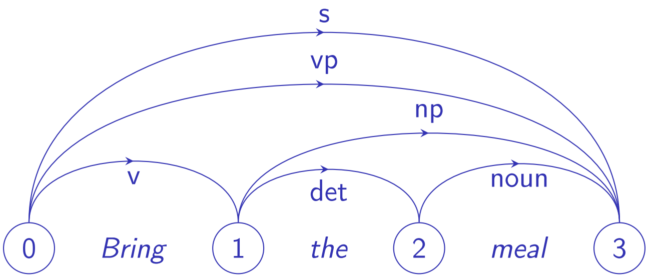
The Structure of a Chart

A chart is a data structure that avoids backtracking
It uses classical grammar rules
It is a graph (DAG) where nodes are intervals between words

0 *Bring* 1 *the* 2 *meal* 3

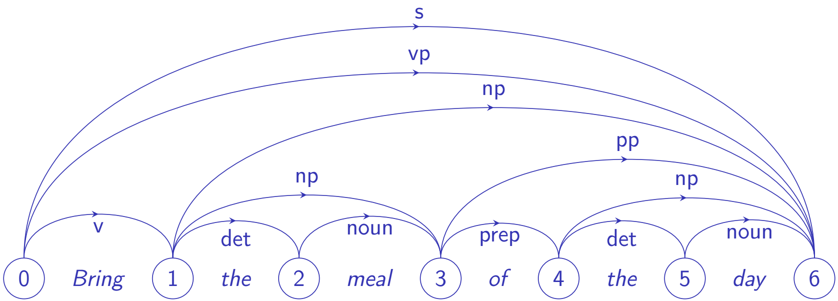
0 *The* 1 *meal* 2 *of* 3 *the* 4 *day* 5

Parsing with a Chart



Charts Contain Alternative Parses

We can view rules `vp --> v, np` and `vp --> v, np, pp` in the chart



The Active Chart

The active chart stores constituents being parsed and marks the rules accordingly.
The rule:

```
np --> det noun •
```

is a completely parsed noun phrase: a determiner and a noun.
The arc is said to be inactive
The rules below are said to be active:

`np --> det • noun`	A determiner has been found
`np --> • det noun`	We are seeking a noun phrase

The Earley Algorithm

Complexity of 0(N^3)
Three modules: the predictor, the scanner, and the completer.
They use phrase-structure rules as:

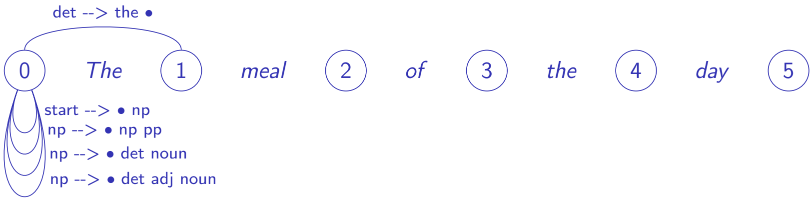
```
start --> • np
np --> det, noun.
np --> det, adj, noun.
np --> np, pp.
pp --> prep, np.
```

The Predictor

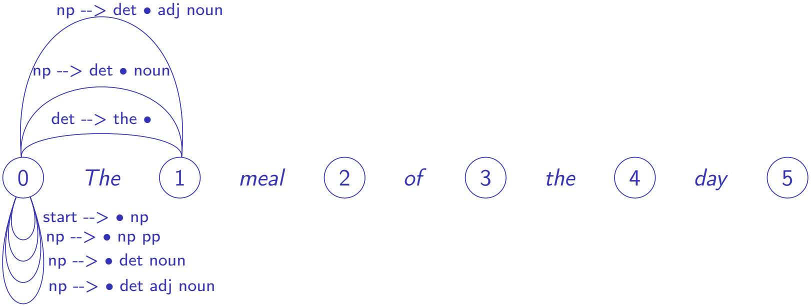
0 *The* 1 *meal* 2 *of* 3 *the* 4 *day* 5

start --> • np
np --> • np pp
np --> • det noun
np --> • det adj noun

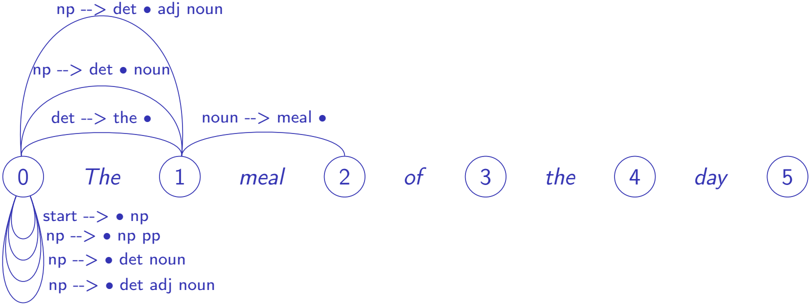
The Scanner


det --> the •
0
The
1
meal
2
of
3
the
4
day
5
start --> • np
np --> • np pp
np --> • det noun
np --> • det adj noun

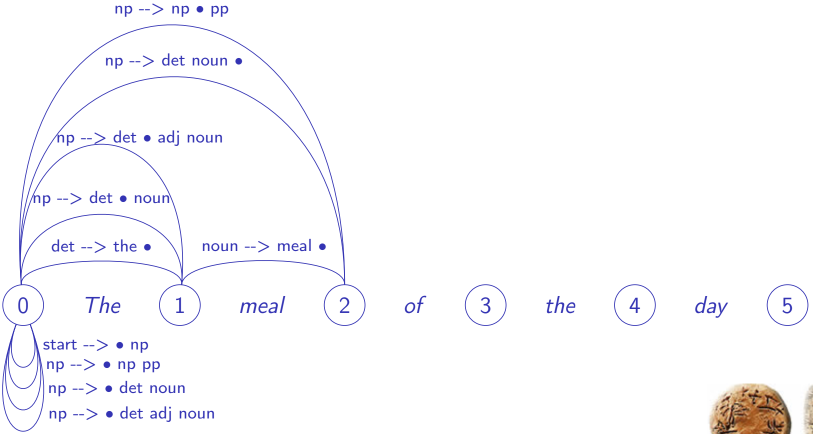
The Completer



The Next Steps (I)



The Next Steps (II)



The Prolog Database

Module	New chart entries
	Position 0
start predictor	arc(start, ['.', np], 0, 0) arc(np, [., d, n], 0, 0), arc(np, [., d, a, n], 0, 0), arc(np, [., np, pp], 0, 0)
	Position 1
scanner completer predictor	arc(d, [the, .], 0, 1) arc(np, [d, ., a, n], 0, 1), arc(np, [d, ., n], 0, 1) []
	Position 2
scanner completer completer predictor	arc(n, [meal, .], 1, 2) arc(np, [d, n, .], 0, 2) arc(np, [np, ., pp], 0, 2), arc(start, [np, .], 0, 2) arc(pp, [., prep, np], 2, 2)

The Prolog Database

	Position 3
scanner	`arc(prep, [of, .], 2, 3)`
completer	`arc(pp, [prep, ., np], 2, 3)`
predictor	`arc(np, [., d, n], 3, 3), arc(np, [., d, a, n], 3, 3), arc(np, [., np, pp], 3, 3)`
	Position 4
scanner	`arc(d, [the, .], 3, 4)`
completer	`arc(np, [d, ., a, n], 3, 4), arc(np, [d, ., n], 3, 4)`
predictor	`[]`
	Position 5
scanner	`arc(n, [day, .], 4, 5)`
completer	`arc(np, [d, n, .], 3, 5)`
completer	`arc(np, [np, ., pp], 3, 5), arc(pp, [prep, np, .], 2, 5)`
completer	`arc(np, [np, pp, .], 0, 5)`
completer	`arc(np, [np, ., pp], 0, 5), arc(start, [np, .], 0, 5)`

Probabilistic Context-Free Grammars

$$P(T,S) = \prod_{rule(i) \text{producing} T} P(rule(i)).$$

where

$$P(lhs \to rhs_i | lhs) = \frac{Count(lhs \to rhs_i)}{\sum_j Count(lhs \to rhs_j)}.$$

An Example of PCFG

Rules			*P*	Rules			*P*
s	-->	np vp	0.8	det	-->	the	1.0
s	-->	vp	0.2	noun	-->	waiter	0.4
np	-->	det noun	0.3	noun	-->	meal	0.3
np	-->	det adj noun	0.2	noun	-->	day	0.3
np	-->	pronoun	0.3	verb	-->	bring	0.4
np	-->	np pp	0.2	verb	-->	slept	0.2
vp	-->	v np	0.6	verb	-->	brought	0.4
vp	-->	v np pp	0.1	pronoun	-->	he	1.0
vp	-->	v pp	0.2	prep	-->	of	0.6
vp	-->	v	0.1	prep	-->	to	0.4
pp	-->	prep np	1.0	adj	-->	big	1.0

Parse Trees of *Bring the meal of the day*

Parse trees

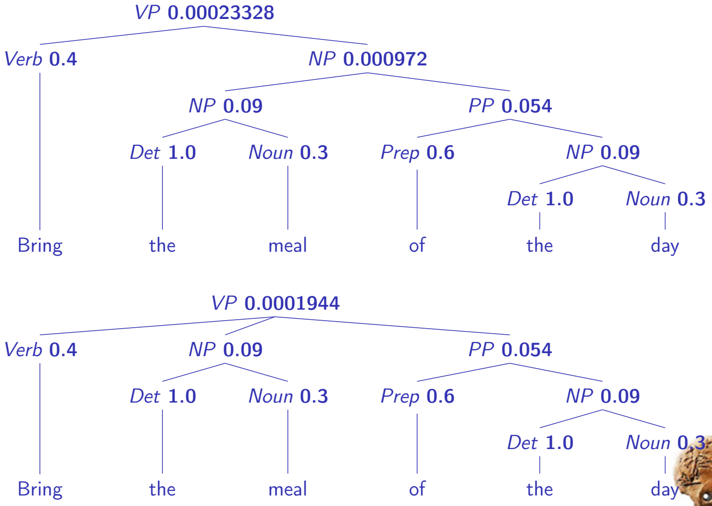
```
T1:  vp(verb(bring),
         np(np(det(the), noun(meal)),
            pp(prep(of), np(det(the), noun(day)))))
T2:  vp(verb(bring),
         np(np(det(the), noun(meal))),
         pp(prep(of), np(det(the), noun(day))))
```

Computing the Probabilities

$$P(T_1, \text{Bring the meal of the day}) =$$
$$P(vp \rightarrow v, np) \times P(v \rightarrow Bring) \times P(np \rightarrow np, pp) \times$$
$$P(np \rightarrow det, noun) \times P(det \rightarrow the) \times P(noun \rightarrow meal) \times$$
$$P(pp \rightarrow prep, np) \times P(prep \rightarrow of) \times P(np \rightarrow det, noun) \times$$
$$P(det \rightarrow the) \times P(noun \rightarrow day) =$$
$$0.6 \times 0.4 \times 0.2 \times 0.3 \times 1.0 \times 0.3 \times 1.0 \times 0.6 \times 0.3 \times 1.0 \times 0.3 = 0.00023328,$$

$$P(T_2, \text{Bring the meal of the day}) =$$
$$P(vp \rightarrow v, np, pp) \times P(v \rightarrow Bring) \times P(np \rightarrow det, noun) \times$$
$$P(det \rightarrow the) \times P(noun \rightarrow meal) \times P(pp \rightarrow prep, np) \times P(prep \rightarrow of) \times$$
$$P(np \rightarrow det, noun) \times P(det \rightarrow the) \times P(noun \rightarrow day) =$$
$$0.1 \times 0.4 \times 0.3 \times 1.0 \times 0.3 \times 1.0 \times 0.6 \times 0.3 \times 1.0 \times 0.3 = 0.0001944.$$

Computing the Probabilities



PCF Grammars Ignore Lexical Preferences

$$\frac{P(T1|\text{Bring the meal of the day})}{P(T2|\text{Bring the meal of the day})} = \frac{P(T1|\text{Bring the meal to the table})}{P(T2|\text{Bring the meal to the table})},$$

$$= \frac{P(vp \rightarrow v, np) \times P(np \rightarrow np, pp)}{P(vp \rightarrow v, np, pp)}$$

PCF grammars do not take into account the lexicon and the attachment preferences of *of* and *to*.

Parser Evaluation

Constituent parsing

$$\text{Recall} = \frac{\text{Number of correct constituents generated by the parser}}{\text{Number of constituents in the manually bracketed corpus}}.$$

$$\text{Precision} = \frac{\text{Number of correct constituents generated by the parser}}{\text{Total number of constituents generated by the parser}}.$$

Bracketing	**Crossing brackets**
(((bring) (the meal)) (of the day))	( ) ( )
((bring) ((the meal) (of the day)))	( ) ( )