Computational Linguistics

Parsing

Aurélie Herbelot

Centre for Mind/Brain Sciences
University of Trento

2016

Table of Contents

Introduction

What is parsing?

- Parsing is the process of automatically assigning a structured interpretation to an input string.
- For natural language, this normally means obtaining one – or more likely a few thousands – syntactic or semantic representations ('parses') of a sentence.

Syntactic vs semantic parsing

- **Syntactic parsing** is concerned with the syntactic structure of sentences, i.e. how words combine into acceptable constituents:
  - *[The cat] chases the mouse.*
  - *Sylvester chases the mouse.*
  - **The chases the mouse.*
- **Semantic parsing** is concerned with the meaning of sentences, i.e. ‘who did what’ in a sentence:
  - *The cat chases the mouse*: the cat is doing the chasing and the mouse is being chased.
  - *It is raining*: nothing is doing the raining.

A syntactic parser: RASP (Briscoe et al, 2006)

```
(|T/txt-sc1/-+|
 (|S/np_vp| |We:1_PPIS2|
  (|V1/v_np| |describe:2_VV0|
   (|NP/det_n1| |a:3_AT1|
    (|N1/n1_pp1|
     (|N1/ap_n1/-| (|AP/a1| (|A1/a| |robust:4_JJ|))
      (|N1/ap_n1/-| (|AP/a1| (|A1/a| |accurate:5_JJ|))
       (|N1/n_n1| |domain-independent:6_NN1|
        (|N1/n| |approach:7_NN1|))))
     (|PP/p1|
      (|P1/p_n1| |to:8_II|
       (|N1/ap_n1/-| (|AP/a1| (|A1/a| |statistical:9_JJ|))
        (|N1/n_ppart| |parsing:10_NN1|
         (|V1/v_ap| |incorporate+ed:11_VVN|
          (|AP/a1|
           (|A1/adv_a1|
            (|AP/a1|
             (|A1/a|
              (|A/pp_adv-coord/+|
               (|PP/p1|
                (|P1/p_np| |into:12_II|
                 (|NP/det_n1| |the:13_AT|
                  (|N1/ap_n1/-| (|AP/a1| (|A1/a| |new:14_JJ|))
                   (|N1/n_pp-of| |release:15_NN1|
                    (|PP/p1|
                     (|P1/p_np| |of:16_IO|
                      (|NP/det_n1| |the:17_AT|
                       (|N1/n-name_n1| |ANLT:18_NP1|
                        (|N1/n| |toolkit:19_NN1|))))))))))
               |,:20_,|
               (|A/cj-end_a/-| |and:21_CC| |publicly:22_RR|))))
            (|A1/a_pp-as| |available:23_JJ|
             (|PP/p1|
              (|P1/p_np| |as:24_CSA|
               (|NP/det_n1| |a:25_AT1|
                (|N1/n_n1| |research:26_NN1|
                 (|N1/n| |tool:27_NN1|)))))))))))))))))
 (|End-punct3/-| |.:28_.|))
```

A semantic parser: Boxer (Bos, 2008)

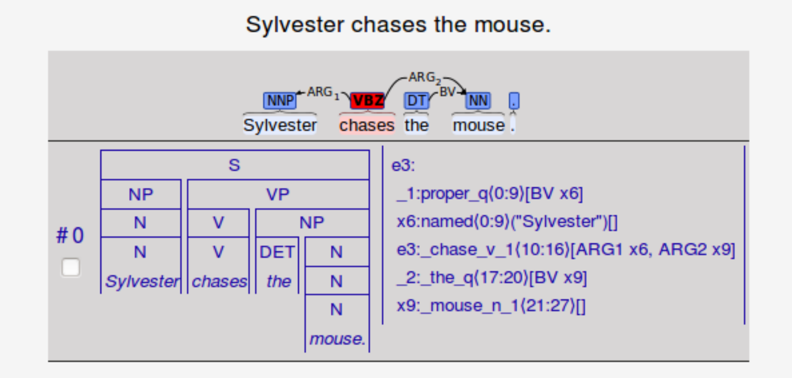
```
% bin/boxer --input working/test.ccg --box
%%% This output was generated by the following command:
%%% bin/boxer --input working/test.ccg --box

:- multifile     sem/3, id/2.
:- discontiguous sem/3, id/2.
:- dynamic       sem/3, id/2.
%%% Every man runs .
id(1,1).
sem(1,[1001:[tok:'Every',pos:'DT',lemma:every,namex:'O'],1002:[tok:ma
%%%  __________________________
%%% |                          |
%%% |..........................|
%%% |   _______    ____________ |
%%% |  |x1     |  |x2          | |
%%% |  |.......|  |............| |
%%% |  |man(x1)|>|agent(x2,x1)| |
%%% |  |_______|  |run(x2)     | |
%%% |             |____________| |
%%% |__________________________|

Attempted: 1. Completed: 1 (100.00%).
```

http://svn.ask.it.usyd.edu.au/trac/candc/wiki/BoxerSimple

Analysis as tree



What? 1000s of interpretations?

- Language is highly ambiguous, i.e. a sentence can generally be interpreted in several ways.
- Some examples of ambiguity that humans are aware of:
  - *Kim saw the woman in the park with a telescope.*
  - *Every student read a book.*
  - *Smoke!*
- Some examples of ambiguity that humans (normally) can't detect:
  - *We bake our cakes with love.*
  - *All students have picked a topic for their thesis.*

Lexical/structural ambiguity

- As far as parsing is concerned, we are interested in two types of ambiguity:
  - **Lexical Ambiguity:** a single word can have more than one syntactic category; for example, *smoke* can be a noun or a verb, *her* can be a pronoun or a possessive determiner.
  - **Structural Ambiguity:** there are a few valid tree forms for a single sequence of words:
    Kim saw ((the woman in the park) (with a telescope)).
    Kim saw ((the woman)(in the park with a telescope)).

Global/local ambiguity

- An important distinction must also be made between:
  - **Global (or total) ambiguity:** in which an entire sentence has several grammatically allowable analyses.
  - **Local (or partial) Ambiguity:** in which portions of a sentence, viewed in isolation, may present several possible options, even though the sentence taken as a whole has only one analysis that fits all its parts.

Global ambiguity

- Global ambiguity can be resolved only by resorting to information outside the sentence (the context, etc.) and so cannot be solved without access to discourse and/or world knowledge.
- A good parser should, however, ensure that all possible readings can be found, so that some further disambiguating process could make use of them.
- For instance:
  We bake our cakes with love.

An example from the ERG

```
We bake our cakes with love.
SENT: We bake our cakes with love.
[ LTOP: h0
INDEX: e2 [ e SF: prop TENSE: pres MOOD: indicative PROG: - PERF: - ]
RELS: < [ pron_rel<0:2> LBL: h4 ARG0: x3 [ x PERS: 1 NUM: pl PRONTYPE: std_pron ] ]
 [ pronoun_q_rel<0:2> LBL: h5 ARG0: x3 RSTR: h6 BODY: h7 ]
 [ "_bake_v_cause_rel"<3:7> LBL: h1 ARG0: e2 ARG1: x3 ARG2: x8 [ x PERS: 3 NUM: pl ] ]
 [ def_explicit_q_rel<8:11> LBL: h9 ARG0: x8 RSTR: h10 BODY: h11 ]
 [ poss_rel<8:11> LBL: h12 ARG0: e13 [ e SF: prop TENSE: untensed MOOD: indicative PROG: - PERF: - ] ARG1: x8 ARG2: x14 [ x PERS: 1 NUM: pl PRO
NTYPE: std_pron ] ]
 [ pronoun_q_rel<8:11> LBL: h15 ARG0: x14 RSTR: h16 BODY: h17 ]
 [ pron_rel<8:11> LBL: h18 ARG0: x14 ]
 [ "_cake_n_1_rel"<12:17> LBL: h12 ARG0: x8 ]
 [ _with_p_rel<18:22> LBL: h12 ARG0: e19 [ e SF: prop TENSE: untensed MOOD: indicative ] ARG1: x8 ARG2: x20 [ x PERS: 3 NUM: sg ] ]
 [ udef_q_rel<23:28> LBL: h21 ARG0: x20 RSTR: h22 BODY: h23 ]
 [ "_love_n_of-for_rel"<23:28> LBL: h24 ARG0: x20 ARG1: i25 ] >
HCONS: < h0 qeq h1 h6 qeq h4 h10 qeq h12 h16 qeq h18 h22 qeq h24 > ]
[ LTOP: h0
INDEX: e2 [ e SF: prop TENSE: pres MOOD: indicative PROG: - PERF: - ]
RELS: < [ pron_rel<0:2> LBL: h4 ARG0: x3 [ x PERS: 1 NUM: pl PRONTYPE: std_pron ] ]
 [ pronoun_q_rel<0:2> LBL: h5 ARG0: x3 RSTR: h6 BODY: h7 ]
 [ "_bake_v_cause_rel"<3:7> LBL: h1 ARG0: e2 ARG1: x3 ARG2: x8 [ x PERS: 3 NUM: pl ] ]
 [ def_explicit_q_rel<8:11> LBL: h9 ARG0: x8 RSTR: h10 BODY: h11 ]
 [ poss_rel<8:11> LBL: h12 ARG0: e13 [ e SF: prop TENSE: untensed MOOD: indicative PROG: - PERF: - ] ARG1: x8 ARG2: x14 [ x PERS: 1 NUM: pl PRO
NTYPE: std_pron ] ]
 [ pronoun_q_rel<8:11> LBL: h15 ARG0: x14 RSTR: h16 BODY: h17 ]
 [ pron_rel<8:11> LBL: h18 ARG0: x14 ]
 [ "_cake_n_1_rel"<12:17> LBL: h12 ARG0: x8 ]
 [ _with_p_rel<18:22> LBL: h1 ARG0: e19 [ e SF: prop TENSE: untensed MOOD: indicative ] ARG1: e2 ARG2: x20 [ x PERS: 3 NUM: sg ] ]
 [ udef_q_rel<23:28> LBL: h21 ARG0: x20 RSTR: h22 BODY: h23 ]
 [ "_love_n_of-for_rel"<23:28> LBL: h24 ARG0: x20 ARG1: i25 ] >
HCONS: < h0 qeq h1 h6 qeq h4 h10 qeq h12 h16 qeq h18 h22 qeq h24 > ]
NOTE: 2 readings, added 1587 / 408 edges to chart (141 fully instantiated, 144 actives used, 115 passives used) RAM: 3172k
```

An example from the ERG

```
[ LTOP: h0
INDEX: e2 [ e SF: prop TENSE: pres MOOD: indicati
RELS: < [ pron_rel<0:2> LBL: h4 ARG0: x3 [ x PERS
 [ pronoun_q_rel<0:2> LBL: h5 ARG0: x3 RSTR: h6 BO
 [ "_bake_v_cause_rel"<3:7> LBL: h1 ARG0: e2 ARG1
 [ def_explicit_q_rel<8:11> LBL: h9 ARG0: x8 RSTR
 [ poss_rel<8:11> LBL: h12 ARG0: e13 [ e SF: prop
NTYPE: std_pron ] ]
 [ pronoun_q_rel<8:11> LBL: h15 ARG0: x14 RSTR: h1
 [ pron_rel<8:11> LBL: h18 ARG0: x14 ]
 [ "_cake_n_1_rel"<12:17> LBL: h12 ARG0: x8 ]
 [ _with_p_rel<18:22> LBL: h1 ARG0: e19 [ e SF: pr
 [ udef_q_rel<23:28> LBL: h21 ARG0: x20 RSTR: h22
 [ "_love_n_of-for_rel"<23:28> LBL: h24 ARG0: x20
HCONS: < h0 qeq h1 h6 qeq h4 h10 qeq h12 h16 qeq h
NOTE: 2 readings, added 1587 / 408 edges to chart
```

But 1000s?

- Real sentences are often long and complex.

  The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at and repair.
  Douglas Adams

The ERG output

```
NOTE: hit RAM limit while unpacking
NOTE: 3521 readings, added 88892 / 79464 edges to chart
```

The ideal parser

- The ideal parser is:
  - correct: it only returns valid analyses of a sentence, given the grammar provided;
  - complete: it returns all possible analyses for a sentence;
  - efficient: it is fast.

From rules to sentences

The rules of parsing

- Parsing needs rules (i.e. a grammar):
  - S $\longrightarrow$ NP VP
  - NP $\longrightarrow$ Det N
  - ...
- Given a sentence and a grammar, a parser returns all possible analyses of the sentence that use the rules in the grammar.

Context-free grammars (CFG)

- A context-free grammar has the form $G = (N, \Sigma, R, S)$ where:
  - N is a set of non-terminal symbols
  - Σ is a set of terminal symbols
  - R is a set of rules of the form $X \rightarrow Y_1 Y_2 ... Y_n$ where
    - $n \leq 0$
    - $X \in N$
    - $Y_i \in (N \cup \Sigma)$
  - $S \in N$ is a root symbol.

An example CFG

- $N = \{N, V, Det, NP, VP, S\}$
- $\Sigma = \{the, cow, eats\}$
- $R = \{$
  Det $\rightarrow$ the
  N $\rightarrow$ cow
  V $\rightarrow$ eats
  NP $\rightarrow$ Det N
  VP $\rightarrow$ V
  S $\rightarrow$ NP VP
  $\}$

Shift-reduce bottom-up parsing

- bottom-up: start with the words in the sentence and build up a tree that terminates with the symbol *S*.
- shift-reduce: one possible (simple) algorithm for parsing. The algorithm processes the sentence one word at a time, left to right (leftmost derivation) or right to left (rightmost derivation). Two operations are possible:
  - shift: push a word on top of the stack;
  - reduce: replace a set of symbols at the top of the stack with the result of a production rule.

Shift-reduce: an example

- **Queue:** *The cow eats.*
- **Stack:**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Shift-reduce: an example

- **Queue:** *cow eats.*
- **Stack:** *The* **shift**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Shift-reduce: an example

- **Queue:** *cow eats.*
- **Stack:** *Det*

reduce

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Shift-reduce: an example

- **Queue:** *eats.*
- **Stack:** *Det cow*

shift

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Shift-reduce: an example

- **Queue:** *eats.*
- **Stack:** *Det N*

reduce

Grammar:

Det → the
N → cow
V → eats
NP → Det N
VP → V
S → NP VP

Shift-reduce: an example

- **Queue:** *eats.*
- **Stack:** *NP*

reduce

Grammar:

Det → the
N → cow
V → eats
NP → Det N
VP → V
S → NP VP

Shift-reduce: an example

- **Queue:**
- **Stack:** *NP eats*

shift

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Shift-reduce: an example

- **Queue:**
- **Stack:** *NP V*

reduce

Grammar:

Det → the
N → cow
V → eats
NP → Det N
VP → V
S → NP VP

Shift-reduce: an example

- **Queue:**
- **Stack:** *NP VP*

reduce

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Shift-reduce: an example

- **Queue:**
- **Stack:** *S* **reduce**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

When to reduce?

- **Queue:** *The cow eats grass on the field.*
- **Stack:**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *cow eats grass on the field.*
- **Stack:** *The* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *cow eats grass on the field.*
- **Stack:** *Det* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *eats grass on the field.*
- **Stack:** *Det cow* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *eats grass on the field.*
- **Stack:** *Det N* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *eats grass on the field.*
- **Stack:** *NP* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *grass on the field.*
- **Stack:** *NP eats* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *grass on the field.*
- **Stack:** *NP V* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *on the field.*
- **Stack:** *NP V grass* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *on the field.*
- **Stack:** *NP V N* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *on the field.*
- **Stack:** *NP V NP* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *on the field.*
- **Stack:** *NP VP*

!!reduce!!

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *the field.*
- **Stack:** *NP VP on* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *the field.*
- **Stack:** *NP VP P* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *field.*
- **Stack:** *NP VP P the* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *field.*
- **Stack:** *NP VP P Det*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP VP P Det field*

shift

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP VP P Det N*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP VP P NP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP VP PP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP VP* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *S* **reduce**
  Resulting parse:
  ((the(cow))((eats(grass))((on(the(field)))))

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *on the field.*
- **Stack:** *NP V NP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *on the field.*
- **Stack:** *NP V NP on* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *the field.*
- **Stack:** *NP V NP P*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *field.*
- **Stack:** *NP V NP P the* **shift**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:** *field.*
- **Stack:** *NP V NP P Det* **reduce**

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP V NP P Det field*

shift

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP V NP P Det N*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP V NP P NP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP V NP PP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP V NP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *NP VP*

reduce

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

When to reduce?

- **Queue:**
- **Stack:** *S* **reduce**
  Resulting parse:
  ((the(cow))((eats((grass)(on(the(field)))))))

Grammar:

Det –> the
N –> cow
N –> grass
N –> field
n –> NP
V –> eats
P –> on
NP –> Det N
NP –> NP PP
PP –> P NP
N –> N PP
VP –> V NP
VP –> VP PP
S –> NP VP

Reducing in ambiguous sentences

- When we reduce affects which reading we select in structurally ambiguous sentences:
  - She saw the woman with the telescope.
  - (She (saw (the woman) with (the telescope)))
    reduce to *np* after *woman*
  - (She (saw (the (woman with (the telescope)))))
    reduce to *np* after *telescope*

Recursive descent top-down parsing

- top-down: start with the symbol *S* and build up a tree that branches out into the words in the sentence.
- In other words, we first assume that the sentence is well-formed and try to prove it using the rules at our disposal.
- We use the rules left-to-right (as opposed to right-to-left in bottom-up parsing):

$np \rightarrow np\ pp$

top-down: let's expand *np* into *np pp* and see whether we get to the sentence...

$np \rightarrow np\ pp$

bottom-up: we have an *np* and a *pp*, let's reduce them to an *np* and see whether we get an *s*...

Recursive descent: an example

- **Analysis:** *s*
- **Try: s $\rightarrow$ np vp**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Recursive descent: an example

- **Analysis:** *s(np vp)*
- **Try: np $\rightarrow$ det n**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Recursive descent: an example

- **Analysis:** *s(np(det n) vp)*
- **Try: det $\rightarrow$ the**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Recursive descent: an example

- **Analysis:** *s(np(det(the) n) vp)*
- **Try: n $\rightarrow$ cow**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Recursive descent: an example

- **Analysis:** *s(np(det(the) n(cow)) vp)*
- **Try: vp $\rightarrow$ eats**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Recursive descent: an example

- **Analysis:** *s(np(det(the) n(cow)) vp(eats))*
- **STOP**

Grammar:

Det $\rightarrow$ the
N $\rightarrow$ cow
V $\rightarrow$ eats
NP $\rightarrow$ Det N
VP $\rightarrow$ V
S $\rightarrow$ NP VP

Dealing with alternatives

- We can *backtrack*: at each step, the algorithm takes notes of the different alternatives available and tries them in turn. Upon failure, it picks the next alternative. Depth-first algorithm.
- We can try and analyse all alternatives in *parallel*. At each step, all possible alternatives are written. Upon failure, disregard the alternative. Breadth-first algorithm.

Bottom-up vs top-down

- Bottom-up:
  - Always generate trees that are consistent with the words in the sentence.
  - Trees that will never lead to the root symbol are explored.
- Top-down:
  - Wastes time exploring alternatives that are inconsistent with the input.
  - Never wastes time exploring a tree that cannot result in a root symbol.

Better parsing

- A *complete* parse of a sentence will output all possible correct alternatives (those that result in the S symbol and an empty queue).
- But outputting every option can be time-consuming and inefficient...
- Plus, we would like to know which parse is more likely.
- The answer: 1) keep track of all options in one chart; 2) use probabilities.

The CKY algorithm

Why CKY?

- The Cocke-Kasami-Younger (CKY) algorithm: A fast parser designed to overcome the inefficiencies of naive parsing.
- The algorithm relies on:
  - Storing intermediate solutions and only pursuing the ‘promising’ ones (those that will contribute to a full parse).
  - The CKY works with a particular grammar: the Chomsky Normal Form (CNF).

CNF grammars

- A context-free grammar is said to be in Chomsky normal form if all of its production rules are of the following form:
  - $A \rightarrow BC$, or
  - $A \rightarrow a$, or
  - $S \rightarrow \epsilon$
- where A, B and C are non-terminal symbols, a is a terminal symbol (a constant), S is the start symbol, and ϵ is the empty string.
- Note: production rules are restricted to produce 2 non-terminals or 1 terminal. Empty productions are not allowed.

The Well-Formed Substring Table (WFST)

- A well-formed substring table is a data structure that contains partial constituency structures.

	1	2	3
1	DT	JJ	NN
2		NN	
3	NP		
	the	angry	dragon

NP –> DT NN
NN –> JJ NN
DT –> the
JJ –> angry
NN –> dragon

The CKY algorithm

1. Given an input sentence with n tokens, create a matrix M with dimensionality $n * n$. Each cell in the matrix corresponds to a sentence span starting at a particular position. E.g. $M[2][2]$ corresponds to a span of 2 starting at word 2 in the sentence.
2. Fill in row 1 of the matrix bottom-up, using rules going to terminals.
3. Fill in each subsequent row top-down, using the already filled rows as constraints.

Example

$$
\begin{array}{lll}
S & \rightarrow & NP\ VP \\
VP & \rightarrow & VP\ PP \\
VP & \rightarrow & V\ NP \\
VP & \rightarrow & eats \\
PP & \rightarrow & P\ NP \\
NP & \rightarrow & Det\ N \\
NP & \rightarrow & she \\
V & \rightarrow & eats \\
P & \rightarrow & with \\
NP & \rightarrow & cake \\
N & \rightarrow & fork \\
Det & \rightarrow & a
\end{array}
$$

She eats cake with a fork.

Example, step 1

S	$\rightarrow$	*NP VP*
VP	$\rightarrow$	*VP PP*
VP	$\rightarrow$	*V NP*
VP	$\rightarrow$	*eats*
PP	$\rightarrow$	*P NP*
NP	$\rightarrow$	*Det N*
NP	$\rightarrow$	*she*
V	$\rightarrow$	*eats*
P	$\rightarrow$	*with*
NP	$\rightarrow$	*cake*
N	$\rightarrow$	*fork*
Det	$\rightarrow$	*a*

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

$_0$she$_1$eats$_2$cake$_3$with$_4$a$_5$fork$_6$

n=6. (For clarity, matrix indices start at 1.)

Example, step 2

$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
VP & \rightarrow & VP\ PP \\
VP & \rightarrow & V\ NP \\
VP & \rightarrow & eats \\
PP & \rightarrow & P\ NP \\
NP & \rightarrow & Det\ N \\
NP & \rightarrow & she \\
V & \rightarrow & eats \\
P & \rightarrow & with \\
NP & \rightarrow & cake \\
N & \rightarrow & fork \\
Det & \rightarrow & a
\end{array}
$$

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2						
3						
4						
5						
6						
	she	eats	cake	with	a	fork

For i so that $1 \leq i \leq 6$,
$M[1][i] = \{A|A \rightarrow a_i \text{ in } P\}$.

Example, step 2

S	$\rightarrow$	*NP VP*
VP	$\rightarrow$	*VP PP*
VP	$\rightarrow$	*V NP*
VP	$\rightarrow$	*eats*
PP	$\rightarrow$	*P NP*
NP	$\rightarrow$	*Det N*
NP	$\rightarrow$	*she*
V	$\rightarrow$	*eats*
P	$\rightarrow$	*with*
NP	$\rightarrow$	*cake*
N	$\rightarrow$	*fork*
Det	$\rightarrow$	*a*

	1	2	3	4	5	6
1 ($l = 1$)	NP	V,VP	NP	P	Det	N
2 ($l = 2$)						
3 ($l = 3$)						
4 ($l = 4$)						
5 ($l = 5$)						
6 ($l = 6$)						
	she	eats	cake	with	a	fork

Note: each row corresponds to a particular span length.

Example, step 3

$S \rightarrow NP\ VP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ NP$
$VP \rightarrow eats$
$PP \rightarrow P\ NP$
$NP \rightarrow Det\ N$
$NP \rightarrow she$
$V \rightarrow eats$
$P \rightarrow with$
$NP \rightarrow cake$
$N \rightarrow fork$
$Det \rightarrow a$

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3						
4						
5						
6						
	she	eats	cake	with	a	fork

Consider spans of length 2:

Each span S can be partitioned into spans S_1 of length 1 and S_2 of length $2 - 1 = 1$

For i so that $1 \leq i \leq 6$,

$M[2][i]\{A|A \rightarrow BC$ in P,

with B in $t_{1,i}$ and C in $t_{1,i}\}$

Example, step 3

S	$\rightarrow$	$NP\ VP$
VP	$\rightarrow$	$VP\ PP$
VP	$\rightarrow$	$V\ NP$
VP	$\rightarrow$	$eats$
PP	$\rightarrow$	$P\ NP$
NP	$\rightarrow$	$Det\ N$
NP	$\rightarrow$	she
V	$\rightarrow$	$eats$
P	$\rightarrow$	$with$
NP	$\rightarrow$	$cake$
N	$\rightarrow$	$fork$
Det	$\rightarrow$	a

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3						
4						
5						
6						
	she	eats	cake	with	a	fork

Consider spans of length 2:

Each span S can be partitioned into spans S_1 of length 1 and S_2 of length $2 - 1 = 1$

For i so that $1 \leq i \leq 6$,

$M[2][i]\{A|A \rightarrow BC$ in P,

with B in $t_{1,i}$ and C in $t_{1,i}\}$

Example, step 3

S	$\rightarrow$	$NP\ VP$
VP	$\rightarrow$	$VP\ PP$
VP	$\rightarrow$	$V\ NP$
VP	$\rightarrow$	$eats$
PP	$\rightarrow$	$P\ NP$
NP	$\rightarrow$	$Det\ N$
NP	$\rightarrow$	she
V	$\rightarrow$	$eats$
P	$\rightarrow$	$with$
NP	$\rightarrow$	$cake$
N	$\rightarrow$	$fork$
Det	$\rightarrow$	a

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3						
4						
5						
6						
	she	eats	cake	with	a	fork

Consider spans of length 2:

Each span S can be partitioned into spans S_1 of length 1 and S_2 of length $2 - 1 = 1$

For i so that $1 \leq i \leq 6$,

$M[2][i]\{A|A \rightarrow BC$ in P,

with B in $t_{1,i}$ and C in $t_{1,i}\}$

Example, step 3

S	$\rightarrow$	$NP\ VP$
VP	$\rightarrow$	$VP\ PP$
VP	$\rightarrow$	$V\ NP$
VP	$\rightarrow$	$eats$
PP	$\rightarrow$	$P\ NP$
NP	$\rightarrow$	$Det\ N$
NP	$\rightarrow$	she
V	$\rightarrow$	$eats$
P	$\rightarrow$	$with$
NP	$\rightarrow$	$cake$
N	$\rightarrow$	$fork$
Det	$\rightarrow$	a

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4						
5						
6						
	she	eats	cake	with	a	fork

Consider spans of length 3:

Possible partitions: 1,2; 2,1.

For i so that $1 \leq i \leq 6$,

$M[3][i]\{A|A \rightarrow BC$ in P,

with B in $t_{k,i}$ and C in $t_{3-k,i+k}$

for $1 \leq k < 3\}$

Example, step 3

S	$\rightarrow$	$NP\ VP$
VP	$\rightarrow$	$VP\ PP$
VP	$\rightarrow$	$V\ NP$
VP	$\rightarrow$	$eats$
PP	$\rightarrow$	$P\ NP$
NP	$\rightarrow$	$Det\ N$
NP	$\rightarrow$	she
V	$\rightarrow$	$eats$
P	$\rightarrow$	$with$
NP	$\rightarrow$	$cake$
N	$\rightarrow$	$fork$
Det	$\rightarrow$	a

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4						
5						
6						
	she	eats	cake	with	a	fork

Consider spans of length 3:

Possible partitions: 1,2; 2,1.

For i so that $1 \leq i \leq 6$,

$M[3][i]\{A|A \rightarrow BC$ in P,

with B in $t_{k,i}$ and C in $t_{3-k,i+k}$

for $1 \leq k < 3\}$

Example, step 3

S	$\rightarrow$	*NP VP*
VP	$\rightarrow$	*VP PP*
VP	$\rightarrow$	*V NP*
VP	$\rightarrow$	*eats*
PP	$\rightarrow$	*P NP*
NP	$\rightarrow$	*Det N*
NP	$\rightarrow$	*she*
V	$\rightarrow$	*eats*
P	$\rightarrow$	*with*
NP	$\rightarrow$	*cake*
N	$\rightarrow$	*fork*
Det	$\rightarrow$	*a*

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4						
5						
6						
	she	eats	cake	with	a	fork

Consider spans of length 4:

Possible partitions: 1,3; 3,1; 2,2.

For i so that $1 \leq i \leq 6$,

$M[4][i]\{A|A \rightarrow BC$ in P,

with B in $t_{k,i}$ and C in $t_{4-k,i+k}$

for $1 \leq k < 4\}$

Example, step 3

S	$\rightarrow$	$NP\ VP$
VP	$\rightarrow$	$VP\ PP$
VP	$\rightarrow$	$V\ NP$
VP	$\rightarrow$	$eats$
PP	$\rightarrow$	$P\ NP$
NP	$\rightarrow$	$Det\ N$
NP	$\rightarrow$	she
V	$\rightarrow$	$eats$
P	$\rightarrow$	$with$
NP	$\rightarrow$	$cake$
N	$\rightarrow$	$fork$
Det	$\rightarrow$	a

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4						
5		**VP**				
6						
	she	eats	cake	with	a	fork

Consider spans of length 5:

Possible partitions: 1,4; 4,1; 3,2; 2,3.

For i so that $1 \leq i \leq 6$,

$M[5][i]\{A|A \rightarrow BC$ in P,

with B in $t_{k,i}$ and C in $t_{5-k,i+k}$

for $1 \leq k < 5\}$

Example, step 3

$S \rightarrow NP\ VP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ NP$
$VP \rightarrow eats$
$PP \rightarrow P\ NP$
$NP \rightarrow Det\ N$
$NP \rightarrow she$
$V \rightarrow eats$
$P \rightarrow with$
$NP \rightarrow cake$
$N \rightarrow fork$
$Det \rightarrow a$

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4						
5		**VP**				
6	**S**					
	she	eats	cake	with	a	fork

Consider spans of length 6:

Possible partitions: 1,5; 5,1; 2,4; 4,2; 3,3.

For i so that $1 \leq i \leq 6$,

$M[6][i]\{A|A \rightarrow BC$ in P,

with B in $t_{k,i}$ and C in $t_{6-k,i+k}$

for $1 \leq k < 6\}$

Example, step 3

S	→	*NP VP*
VP	→	*VP PP*
VP	→	*V NP*
VP	→	*eats*
PP	→	*P NP*
NP	→	*Det N*
NP	→	*she*
V	→	*eats*
P	→	*with*
NP	→	*cake*
N	→	*fork*
Det	→	*a*

	1	2	3	4	5	6
1	NP	V,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4						
5		**VP**				
6	**S**					
	she	eats	cake	with	a	fork

Note: the sentence has a parse if a root symbol (**S**) is found in row *n*.

The algorithm

(20) **Algorithme de Cocke-Younger-Kasami :**

1. Poser $\mathrm{t}_{i,1} = \{A \mid A \longrightarrow a_i$ est dans P$\}$ pour chaque i, $1 \leq i \leq n$.
2. Poser $\mathrm{t}_{i,j} = \{A \mid$ pour k, $1 \leq$ k $<$ j, A $\longrightarrow BC$ dans P,
   B est dans $t_{i,k}$, et C est dans $t_{i+k,j-k}\}$.
3. Répéter (2) jusqu'à ce que la table soit pleine.

Eric Wehrli. 2005. *L'analyse syntaxique des langues naturelles : problèmes et méthodes*

CKY: a mixed algorithm

- CKY uses a mixture of the bottom-up and top-down algorithms.
- For each row in the matrix, the cells are filled in respecting the constraints of lower-numbered rows (or the terminals, in the case of row 1).
- For each cell, we propose a rule from the grammar (top-down) and validate it against what we know already.
- All alternatives in one chart.

Alternatives in the CKY chart

$S \rightarrow NP\ VP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ NP$
$VP \rightarrow eats$
$PP \rightarrow P\ NP$
$NP \rightarrow Det\ N$
$NP \rightarrow she$
$V \rightarrow eats$
$P \rightarrow with$
$NP \rightarrow cake$
$N \rightarrow fork$
$Det \rightarrow a$
$NP \rightarrow NP\ PP$

	1	2	3	4	5	6
1	NP	,VP	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4			**NP**			
5		**VP**				
6	**S**					
	she	eats	cake	with	a	fork

(she(eats (cake))(with(a (fork))))

Alternatives in the CKY chart

$S \rightarrow NP\ VP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ NP$
$VP \rightarrow eats$
$PP \rightarrow P\ NP$
$NP \rightarrow Det\ N$
$NP \rightarrow she$
$V \rightarrow eats$
$P \rightarrow with$
$NP \rightarrow cake$
$N \rightarrow fork$
$Det \rightarrow a$
$NP \rightarrow NP\ PP$

	1	2	3	4	5	6
1	NP	**V,VP**	NP	P	Det	N
2	**S**	**VP**			**NP**	
3	**S**			**PP**		
4			**NP**			
5		**VP**				
6	**S**					
	she	eats	cake	with	a	fork

(she (eats)(cake(with(a(fork)))))

Recognition vs parsing

- **Recognition:** is this a sentence or not?
  Fill in the chart and check we have an **S** in row *n*.
- **Parsing:** what is the analysis of the sentence we found?
  Keep track of the links used in the algorithm, and return them at the end.

Probabilistic Context-Free Grammars (PCFGs)

Where do grammars come from?

- Manually written: time-consuming, potentially low-coverage, but high precision.
- Learnt from a *treebank*: some text was annotated by humans and the parser is trained on this annotation.
- Automatically induced: unsupervised parsing from raw (or POS-tagged) text. Cheap, and potentially close to what humans do. But so far, precision has remained low.

Treebanking

- The main treebank used by the parsing community is the Penn Treebank (PTB), created in the early 90s.
- Around 1M words of news text, annotated with phrase-structure trees.
- The PTB took 3 years to create, with a few annotators. See Clark (2010) for more historical details.

Problems with the PTB

- Focus on news text. Specific syntax, vocabulary, speakers, etc.
- Relatively small test set (2400 sentences), which has been used over and over again. Possibility of *overfitting* the models.

Putting probabilities on rules

- With the availability of a treebank, we can put probabilities on grammar rules:
  - s –> np vp 0.9
  - s–> vp 0.1
  - ...
- Such probabilities will help us decide which parses are more likely – or in case we want to prioritise efficiency – which alternatives to discard at each decision point.

Probability of a tree

- Given the probabilities assigned to rules $r_{1...n}$, a probability can be assigned to a particular tree:

$$P(T) = \prod_{n \in T} P(r_n) \tag{1}$$

- The best parse is then given by:

$$T(S) = \operatorname*{argmax}_{T \in \tau(S)} P(T) \tag{2}$$

Example

S	$\rightarrow$	*NP VP*	1.0
VP	$\rightarrow$	*VP PP*	0.7
VP	$\rightarrow$	*V NP*	0.5
VP	$\rightarrow$	*eats*	0.1
PP	$\rightarrow$	*P NP*	0.8
NP	$\rightarrow$	*Det N*	0.7
NP	$\rightarrow$	*NP PP*	0.2
NP	$\rightarrow$	*she*	0.1
NP	$\rightarrow$	*cake*	0.1
V	$\rightarrow$	*eats*	0.1
P	$\rightarrow$	*with*	0.2
N	$\rightarrow$	*fork*	0.1
Det	$\rightarrow$	*a*	0.2

	1	2	3	4	5	6
1	**NP**	**V**,VP	**NP**	**P**	**Det**	**N**
2	S	**VP**			**NP**	
3	S			**PP**		
4						
5		**VP**				
6	**S**					
	she	eats	cake	with	a	fork

(she(eats (cake))(with(a (fork))))

$P(T) = 0.1 * 0.1 * 0.1 * 0.2 * 0.2 * 0.1 * 0.5 * 0.7 * 0.8 * 0.7 * 1.0 = 7.84.10^-7$

Example

S	$\rightarrow$	$NP\ VP$	1.0
VP	$\rightarrow$	$VP\ PP$	0.7
VP	$\rightarrow$	$V\ NP$	0.5
VP	$\rightarrow$	$eats$	0.1
PP	$\rightarrow$	$P\ NP$	0.8
NP	$\rightarrow$	$Det\ N$	0.7
NP	$\rightarrow$	$NP\ PP$	0.2
NP	$\rightarrow$	she	0.1
NP	$\rightarrow$	$cake$	0.1
V	$\rightarrow$	$eats$	0.1
P	$\rightarrow$	$with$	0.2
N	$\rightarrow$	$fork$	0.1
Det	$\rightarrow$	a	0.2

	1	2	3	4	5	6
1	**NP**	**V**,VP	**NP**	**P**	**Det**	**N**
2	S				**NP**	
3	S			**PP**		
4			**NP**			
5		**VP**				
6	**S**					
	she	eats	cake	with	a	fork

(she (eats)(cake(with(a(fork)))))

$P(T) = 0.1 * 0.1 * 0.1 * 0.2 * 0.2 * 0.1 * 0.7 * 0.8 * 0.2 * 0.5 * 1.0 =$

$2.24.10^-7$

Parser evaluation

Evaluation measures

- The parser is evaluated against a ‘gold standard’, i.e. a manually (or mostly manually) annotated treebank.

- C = number of correct constituents in the system’s parse.
- N = number of constituents in the system’s parse.
- N_G = number of constituents in the correct, gold standard parse.

- Precision: $P = C/N$
- Recall: $P = C/N_G$

Precision and recall

- Precision and recall often work against each other: precise systems don't have large coverage; large-coverage systems let in more errors.
- To calculate a score taking both precision and recall into account, use F-score:

$$(1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{3}$$

- For $\beta = 1$, the F-score gives equal weight to precision and recall. For $\beta > 1$, more weight is given to precision. For $\beta < 1$, more weight is given to recall.

How well do they work?

- On the PTB, parsers achieve scores well into the 90% precision.
- But danger that systems have overfitted on the PTB.
- Things are much harder on:
  - spoken language;
  - tweets;
  - generally: other domains, styles.

...

That's it!