Syntactic Formalisms for Parsing Natural Languages

Aleš Horák, Miloš Jakubíček, Vojtěch Kovář
(based on slides by Juyeon Kang)

ia161@nlp.fi.muni.cz

Autumn 2013

Basic parsing methods

Main points

- Context-free grammar
- Parsing methods
  - Top-down or bottom-up
  - Directional or non-directional
- Basic parsing algorithms
  - Unger
  - CKY (or CYK)
  - Left-corner parsing
  - Earley

Ambiguity in Natural Language

- Notion of ambiguity
  - Essential ambiguity: same syntactic structure but the semantics differ
  - Spurious ambiguity: different syntactic structure but no change in semantics

  There is no unambiguous languages!
- An input may have exponentially many parses
- Should identify the “correct” parse

Ambiguity in Natural Language

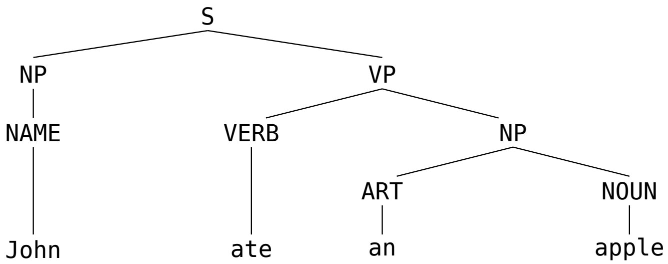
- Main idea of parsing

Parsing (syntactic structure)

Input: sequence of tokens

`John ate an apple`

Output: `parse tree`



Ambiguity in Natural Language

- Basic connection between a sentence and the grammar it derives from is the “parse tree”, which describes how the grammar was used to produce the sentences.
- For the reconstruction of this connection we need a “parsing techniques”

Ambiguity in Natural Language

- **Word categories**: Traditional parts of speech

Noun	Names of things	boy, cat, truth
Verb	Action or state	become, hit
Pronoun	Used for noun	I, you, we
Adverb	Modifies V, Adj, Adv	sadly, very
Adjective	Modifies noun	happy, clever
Conjunction	Joins things	and, but, while
Preposition	Relation of N	to, from, into
Interjection	An outcry	ouch, oh, alas, psst

Formal language

- Symbolic string set which describe infinitely unlimited language as mathematical tool for recognizing and generating languages.
- Topic of formal language: finding finitely infinite languages using rewriting system.
- Three basic components of formal language: finite symbol set, finite string set, finite formal rule set

Constituency

- Sentences have parts, some of which appear to have subparts. These groupings of words that go together we will call constituents.

(How do we know they go together?)

- I hit the man with a cleaver
  I hit [the man with a cleaver]
  I hit [the man] with a cleaver

- You could not go to her party
  You [could not] go to her party
  You could [not go] to her party

The Chomsky hierarchy

- Type 0 Languages / Grammars (LRE: Recursively enumerable grammar)
  Rewrite rules $\alpha \rightarrow \beta$
  where α and β are any string of terminals and non-terminals

- Type 1 Context-sensitive Languages / Grammars (LCS)
  Rewrite rules $\alpha X \beta \rightarrow \alpha \Upsilon \beta$
  where X is a non-terminal, and α, Υ, β are any string of terminals and non-terminals, (Υ must be non-empty but strings α and β can be empty).

- Type 2 Context-free Languages / Grammars (LCF)
  Rewrite rules $X \rightarrow \Upsilon$
  where X is a non-terminal and Υ is any string of terminals and non-terminals

- Type 3 Regular Languages / Grammars (LREG)
  Rewrite rules $X \rightarrow \alpha Y$
  where X, Y are single non-terminals, and α is a string of terminals; Y might be missing.

The Chomsky hierarchy

Type 0 > 1 > 2 > 3

according to generative power

- Superior language can generate inferior language but superior language is more inefficient and slow than inferior language.

The Chomsky hierarchy

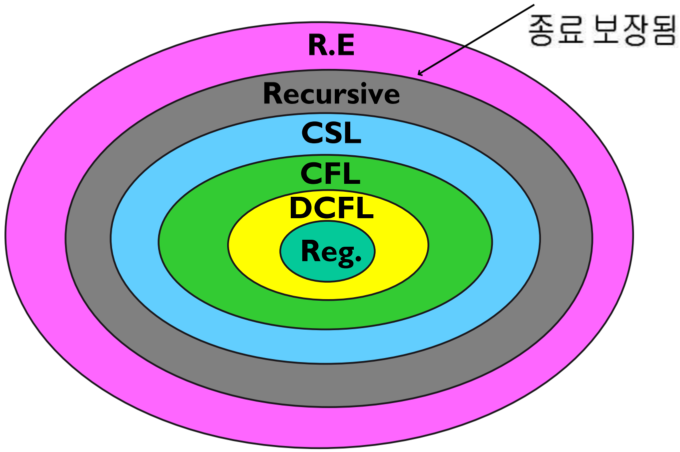


Figure : Chomsky hierarchy

Context-free grammar (Type 2)

The most common way of modeling constituency.

The idea of basing a grammar on constituent structure dates back to Wilhem Wundt (1890), but not formalized until Chomsky (1956), and, independently, by Backus (1959).

CFG = Context-Free Grammar = Phrase Structure Grammar= BNF = Backus-Naur Form

Context-free grammar (Type 2)

- CFG rewriting rule

$$X \rightarrow \Upsilon$$

where X is a non-terminal symbol and ϒ is string consisting of terminals/non-terminals.

The term “Context-free” expresses the fact that the non-terminal *v* can always be replaced by *w*, regardless of the context in which it occurs.

Context-free grammar (Type 2)

G = < T, N, S, R>

T is set of terminals (lexicon)

N is set of non-terminals (written in capital letter). S is start symbol (one of the non-terminals)

R is rules/productions of the form $X \rightarrow \Upsilon$, where X is a non-terminal and ϒ is a sequence of terminals and non-terminals (may be empty).

A grammar G generates a language L

Example1 of Context-Free Grammar

G = < T, N, S, R>

T = { that, this, a, the, man, book, flight, meal, include, read, does }

N = { S, NP, NOM, VP, DET ,N, V, AUX }

S = S

R = {

S → NP VP
S → Aux NP VP
S → VP
NP → Det NOM
NP → N
VP → V
VP → V NP
}

Det → that | this | a | the
N → book | flight | meal | man
V → book | include | read
AUX → does

Example2 of Context-Free Grammar

R1: S -> NP VP
R2: NP -> DET N
R3: NP -> NP PNP
R4: NP -> PN
R5: VP -> V
R6: VP -> V NP
R7: VP -> V PNP
R8: VP -> V NP PNP
R9: VP -> V PNP PNP
R10: PNP -> PP NP
R11: PP-> to|from|of
R12: DET -> an|a
R13: DET -> his|her
R14: DET -> the
R15: V -> eat|serve
R16: V -> give
R17: V -> speak|speaks
R18: V -> discuss
R19: PN -> John|Mark
R20: PN -> Mary|Juliette
R21: N -> daugther|mother
R22: N -> son|boy
R23: N -> salad|soup|meat
R24: N -> desert|cheese|bread
R25: ADJ -> small|kind

Simplified example of $CFG = G_D$

Example2 of Context-Free Grammar

Using the presented grammar, we make a first derivation for the sentence *"John speaks"*,

S -> G_D NP VP (by R1)
S -> G_D PN VP (by **R4**)
-> G_D John VP (by R19)
-> G_D John V (by **R5**)
-> G_D John speaks (by R17)

Example2 of Context-Free Grammar

Another derivation of *"John speaks"* from G_D using rule 5 before rule 4

S -> G_D NP VP
S -> G_D NP V
-> G_D NP speaks
-> G_D PN speaks
-> G_D John speaks

Production Rule 3

NP -> NP PNP

Because it contains the same symbol in his left and his right, we say that the production having this property is recursive.

Production Rule 3

This property of $R3$ involves that the language generated by the grammar G_D is infinite, because we can create the sentences of arbitrary length by iterative application of $R3$.

Test

NP -> G_D NP PNP -> G_D NP PNP PNP -> G_D NP PNP PNP PNP....

- The son of John speaks
- The son of the mother of John speaks
- The son of the daughter of the daughter ....of John speaks.

Production Rule 3

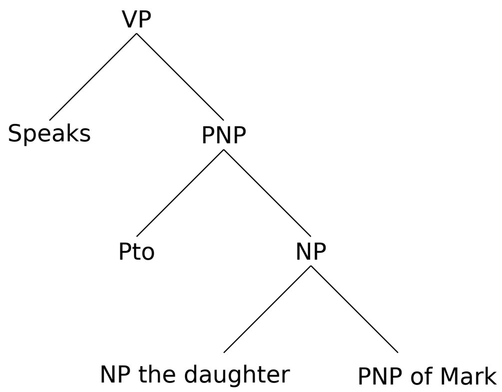
- Last remark concerning this grammar (G_D)

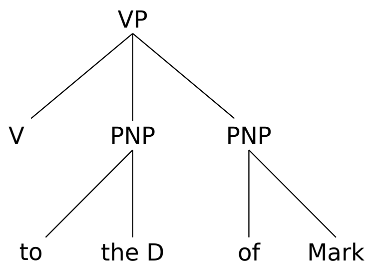
This grammar can generate sentences which are **ambiguous**.
"John speaks to the daughter of Mark"

Example

1. A conversation between **John** and **the daughter of Mark** (R7)
2. A conversation **about Mark** between **John** and **the daughter** (R9)

Production Rule 3





Commonly used non-terminal abbreviations

S	sentence
NP	noun phrase
PP	prepositional phrase
VP	verb phrase
*X*P	*X* phrase
N	noun
PREP	preposition
V	verb
DET/ART	determiner / article
ADJ	adjective
ADV	adverb
AUX	auxiliary verb
PN	proper noun

Parsing methods

- Classification of parsing methods

  Top-down parsing vs. **Bottom-up** parsing

  - Directional vs. non-directional parsing

Top-down or bottom-up

- Top-down parsing
  - The sentence from the start symbol, the production tree is reconstructed from the top downwards
  - Identify the production rules in prefix order
  - Never explores a tree that cannot result in an *S*
  - BUT Wastes time generating trees inconsistent with the input
- Bottom-up parsing
  - The sentence back to the start symbol
  - Identify the production rules in postfix order
  - Never generates trees that are not grounded in the input
  - BUT Wastes time generating trees that do not lead to an *S*

Top-down parsing

- Top-down parsing is goal-directed.
  - A top-down parser starts with a list of constituents to be built.
  - It rewrites the goals in the goal list by matching one against the LHS of the grammar rules,
  - and expanding it with the RHS,
  - ...attempting to match the sentence to be derived.
- If a goal can be rewritten in several ways, then there is a choice of which rule to apply (search problem)
- Can use depth-first or breadth-first search, and goal ordering.

Top-down parsing

Simulation of the operation of parser in top-down methods

The son speaks

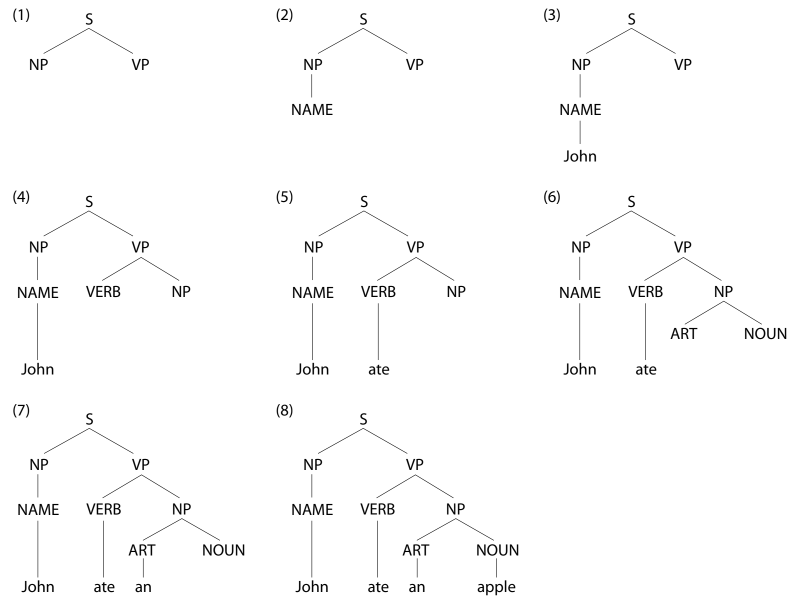
1. S
2. NP VP
3. DET N VP
4. 4. a N VP. Fail: input begin by the. We return to DET N VP
5. the N VP
6. the daughter VP. New fail α=le N VP
   ......
7. the son VP
8. the son V
9. the son speaks.
   ......

Top-down parsing

Top-down parsing example

S	$\rightarrow$	NP VP
	$\rightarrow$	NAME VP
	$\rightarrow$	"John" VP
	$\rightarrow$	"John" VERV NP
	$\rightarrow$	"John" "ate" NP
	$\rightarrow$	"John" "ate" DET NOUN
	$\rightarrow$	"John" "ate" "an" NOUN
	$\rightarrow$	"John" "ate" "an" "apple"

Top-down parsing



Top-down parsing

Algorithm of top-down left-right (LR) parsing

α is a primal current word, u input to be recognized.

tdlrp = main function
tdlrp (α, u)

begin
  if $(\alpha = u)$ then return (true) fi
    A $= u_1 \ldots\ldots u_k A \Upsilon$
  while $(\exists A -> \beta)$ do
    $(\beta = u_{k+1} \ldots\ldots\ldots u_{k+1}{}^{\delta})$ with $\delta = \epsilon$ ou $\delta = A\ldots$
  if $(tdlrp(u_1 \ldots\ldots u_{k+1}{}^{\delta}\Upsilon) = true)$ then return(true) fi
  od
    return (false)
end

Top-down parsing

Problems in top-down parsing

- Left recursive rules... e.g. NP → NP PP... lead to infinite recursion
- Will do badly if there are many different rules for the same LHS. Consider if there are 600 rules for S, 599 of which start with NP, but one of which starts with a V, and the sentence starts with a V.
- Top-down parsers do well if there is useful grammar-driven control: search is directed by the grammar.
- Top-down is hopeless for rewriting parts of speech (preterminals) with words (terminals).

Bottom-up parsing

- Bottom-up parsing is data-directed.
  - The initial goal list of a bottom-up parser is the string to be parsed.
  - If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule.
  - Parsing is finished when the goal list contains just the start symbol.
- If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)
- Can use depth-first or breadth-first search, and goal ordering.

Bottom-up parsing

Let's suppose that we have a sentence *"the son eats his soup"* in the grammar G_D.

Question

How we can do to verify that the word belong to the language generated by the grammar G_D and if the answer is positive to assign a tree?

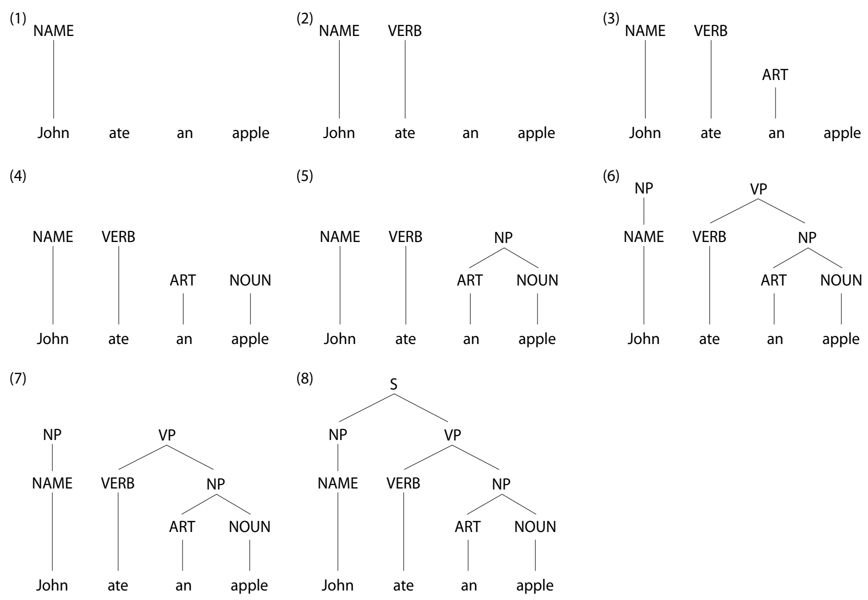
→ The first idea can be given in the following algorithms:

Bottom-up parsing

Bottom-up parsing example

"John"	"ate"	"an"	"apple"
→ NAME	"ate"	"an"	"apple"
→ NAME	VERV	"an"	"apple"
→ NAME	VERV	DET	"apple"
→ NAME	VERV	DET	NOUN
→ NP	VERV	DET	NOUN
→ NP	VERV	NP	
→ NP	VP		
→ S			

Bottom-up parsing



Bottom-up parsing

Problems with bottom-up parsing

- Unable to deal with empty categories: termination problem, unless rewriting empties as constituents is somehow restricted (but then it's generally incomplete)
- Inefficient when there is great lexical ambiguity (grammar-driven control might help here). Conversely, it is data-directed: it attempts to parse the words that are there.
- Both Top-down (LL) and Bottom-up (LR) parsers can (and frequently do) do work exponential in the sentence length on NLP problems.

Left-corner parsing

Left-corner parsing

- Bottom-up with top-down filtering:
  - combine top-down processing with bottom-up processing in order to avoid going wrong in the ways that we are prone to go wrong with pure top-down and pure bottom-up techniques

Left-corner parsing

Going wrong with top-down parsing

```
S -> NP VP
NP -> DET N
NP -> PN
VP -> IV
DET -> the
N -> robber
PN -> Vincent
IV -> died
```

Vincent died.

Left-corner parsing

Going wrong with bottom-up parsing

```
S -> NP VP
NP -> DET N
VP -> IV
VP -> TV NP
TV -> plant
IV -> died
DET-> the
N -> plant
```

The plant died.

1. DET plant died
2. DET TV IV Fail
3. DET N IV OK
4. NP VP OK
5. S

Left-corner parsing

Combining Top-down and Bottom-up Information

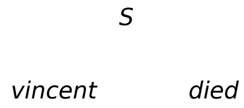
S -> NP VP
NP -> DET N
NP -> PN
VP -> IV
DET -> the
N -> robber
PN -> Vincent
IV -> died

Vincent died.

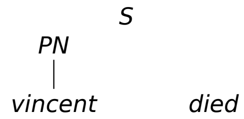
Left-corner parsing

Now, let's look at how a left-corner recognizer would proceed to recognize Vincent died.

1. Input: Vincent died. Recognize an S. (Top-down prediction.)

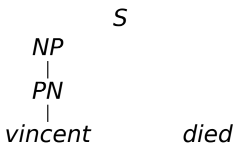


2. The category of the first word of the input is PN. (Bottom-up step using a lexical rule.)

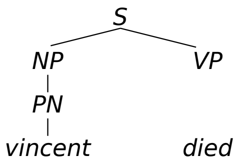


Left-corner parsing

3. Select a rule that has at its left corner : NP-> PN. (Bottom-up step using a phrase structure rule.)

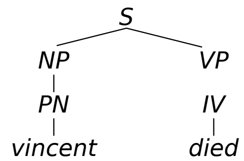


4. Select a rule that has at its left corner: S->NP VP. (Bottom-up step.)
5. Match! The left hand side of the rule matches with S, the category we are trying to recognize.

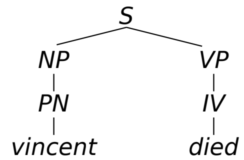


Left-corner parsing

6. Input: died. Recognize a VP. (Top-down prediction.)
7. The category of the first word of the input is IV. (Bottom-up step.)



8. Select a rule that has at its left corner: VP->IV. (Bottom-up step.)
9. Match! The left hand side of the rule matches with VP, the category we are trying to recognize.



Left-corner parsing

- What is a left-corner of a rule:
  - the first symbol on the right hand side. For example, *NP* is the left corner of the rule *S* → *NP VP*, and *IV* is the left corner of the rule *VP* → *IV*. Similarly, we can say that *Vincent* is the left corner of the lexical rule *PN* → *Vincent*.

Left-corner parsing

- What is a left-corner of a rule:
  - “Predictive” parser : it uses grammatical knowledge to predict what should come next, given what it has found already.
  - 4 operations creating new items from old: “Shift”, “Predict”, “Match” and “Reduce”

Left-corner parsing

- Definition (Corner relation)
  The relation $\angle$ between non-terminals A and B such that $B \angle A$ if and only if there is a rule $A \to B\alpha$, where α denotes some sequence of grammar symbols

- Definition (Left corner relation)
  The transitive and reflexive closure of $\angle$ is denoted by $\angle^*$, which is called left-corner relation

Left-corner parsing

Left-corner table

Non Terminal	Left-corners
S	S NP time an VorN files
NP	NP time an VorN files
VP	VP VorN files VorP like
PP	PP VorP like
VorN	VorN files
VorP	VorP like

Grammar

S → NP VP
S → S PP
NP → time
NP → an arrow
NP → VorN
VP → VorN
VP → VorP NP
PP → VorP NP
VorN → files
VorP → like

How to deal with ambiguity?

- Backtracking
  - Try all variants subsequently.
- Determinism
  - Just choose one variant and keep it (i.,e. greedy).
- Parallelism
  - Try all variants in parallel.
- Underspecification
  - Do not desambiguate, keep ambiguity.

Summary

- One view on parsing: parsing as a phrase-structure formal grammar recognition task
- Parsing approaches: top-down, bottom-up, left-corner