Syntax — I

Anton Alekseev
Steklov Mathematical Institute in St Petersburg

ITMO University, St Petersburg, 2019
anton.m.alexeyev+itmo@gmail.com

http://bit.ly/2hPyEMH

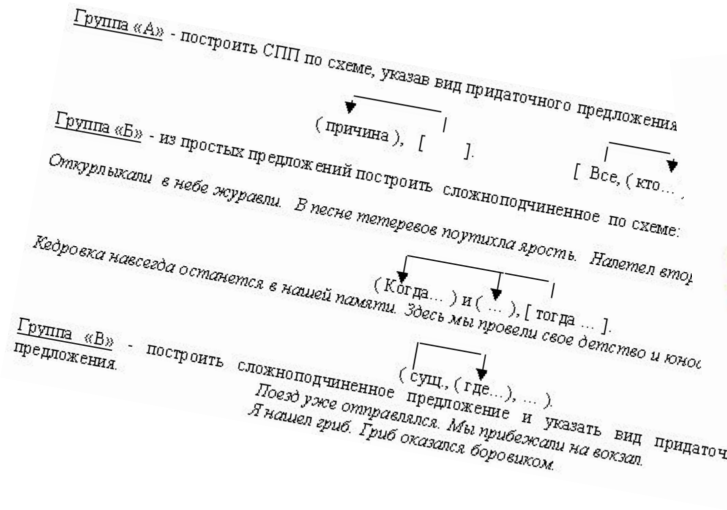
Plan

1. What is parsing and why we need it
2. Phrase structure grammar
    a. Intuition
    b. Formal grammars
    c. CKY-algorithms
    d. *Shallow parsing
    e. Probabilistic grammars
    f. Probabilistic grammars lexicalization
    g. Quality evaluation
    h. Tools and data

Parsing

machine analysis of the text structure, esp. the **sentence structure**

We’ve all done it at school, and sometimes machines can do it as well



Группа «А» - построить СПП по схеме, указав вид придаточного предложения

(причина), [ ].

[Все, (кто… ,

Группа «Б» - из простых предложений построить сложноподчиненное по схеме:

Откурлыкали в небе журавли. В песне тетеревов поутихла ярость. Налетел вто

Кедровка навсегда останется в нашей памяти. Здесь мы провели свое детство и юнос

(Когда…) и (…), [тогда …].

Группа «В» - построить сложноподчиненное предложение и указать вид придаточного предложения.

(сущ., (где…), …).

Поезд уже отправлялся. Мы прибежали на вокзал.

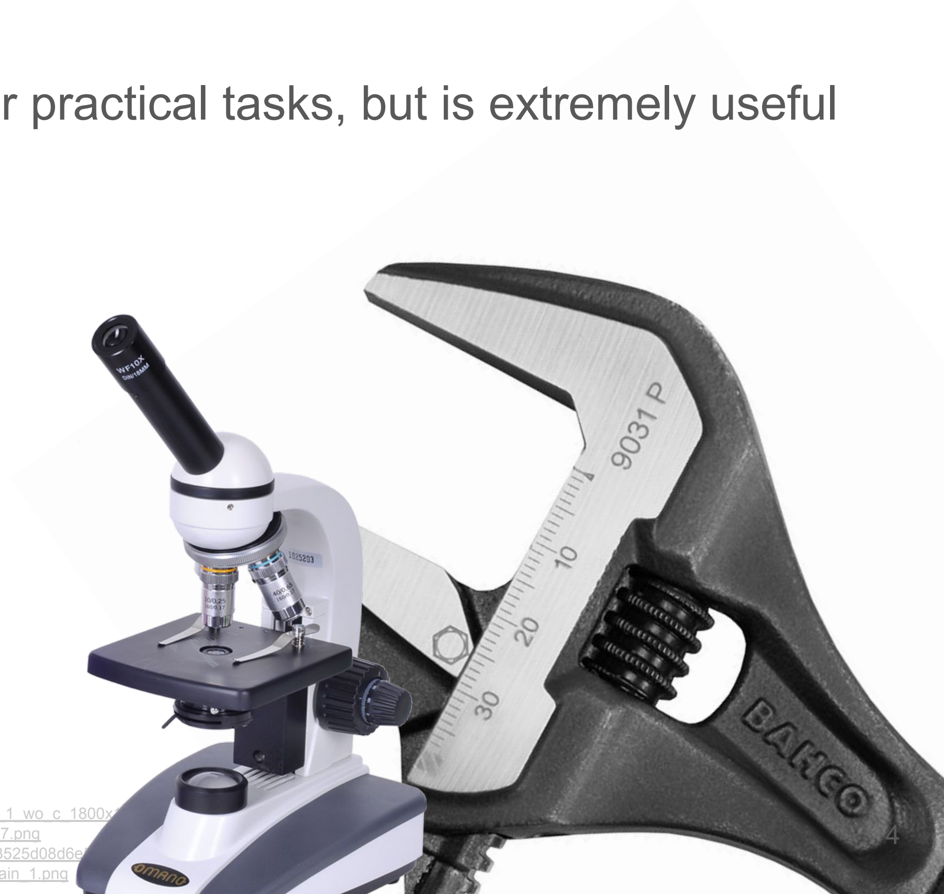
Я нашел гриб. Гриб оказался боровиком.

https://www.kakprosto.ru/kak-30136-kak-delat-sintaksicheskiy-razbor-predlozheniya
http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/516044/img1.gif

Motivation

Sentence structure itself is never a goal for practical tasks, but is extremely useful as a preprocessing step e.g. for:

- facts extraction and opinion mining,
- text summarization,
- machine translation, etc.



https://static-eu.insales.ru/images/products/1/850/61170514/9031_P_open_1_wo_c_1800x
http://papus666.narod.ru/clipart/g/gaek/gaechnkl017.png
https://www.microscope.com/media/catalog/product/cache/2/image/9df78eab33525d08d6e
mano_om136c_monocular_compound_microscope_main_1.png

Syntax

grammar subset studying sentences and ways of combining words within a sentence

Main approaches to syntax description

1) dependency grammar

   Tesnière, L. 1959. Éléments de syntaxe structurale. Paris: Klincksieck

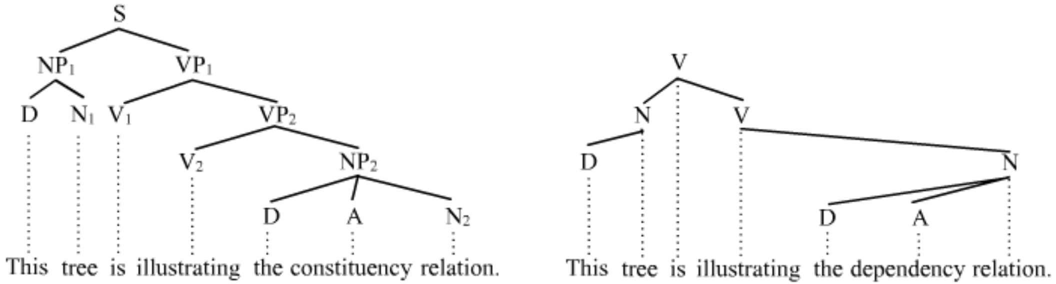
2) phrase structure grammars

   Chomsky, Noam 1957. Syntactic structures. The Hague/Paris: Mouton

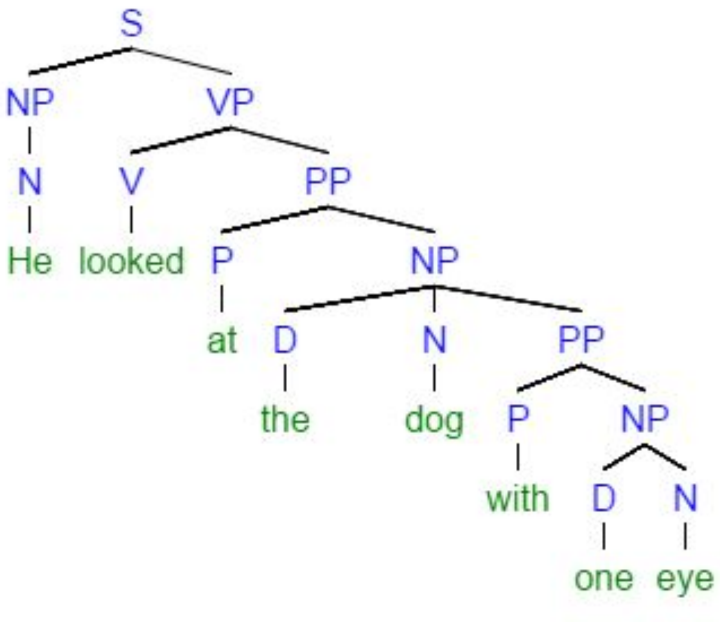
3) link grammar

   Daniel Sleator and Davy Temperley. 1991. Parsing English with a Link Grammar. Carnegie Mellon University Computer Science technical report, October 1991.

4) hybrid approaches



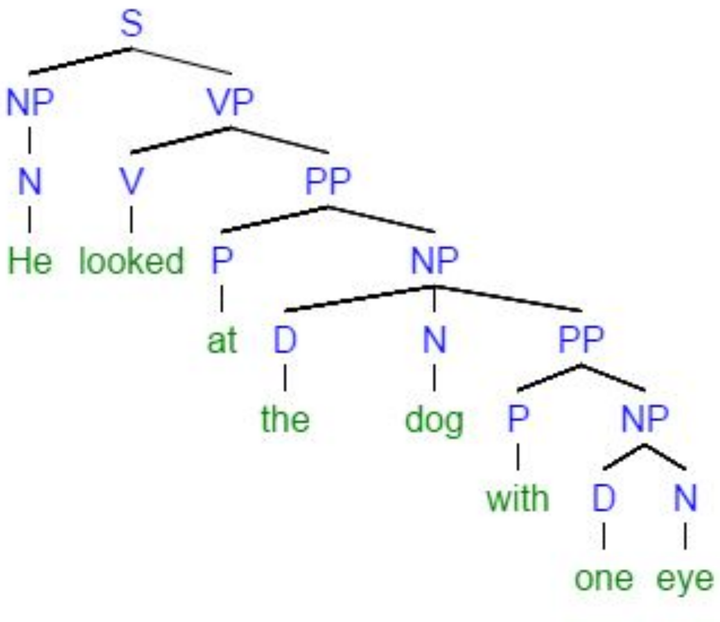
Before we define stuff thoroughly: an example



https://i.imgur.com/ShMtNEy.png

Is everything OK here?

Before we define stuff thoroughly: an example



https://i.imgur.com/ShMtNEy.png

Is everything OK here?

Yes! But if we want to say one has LOOKED with one eye, then the tree should be different

This is called **attachment ambiguity** (“dunno where to hang the subtree”)

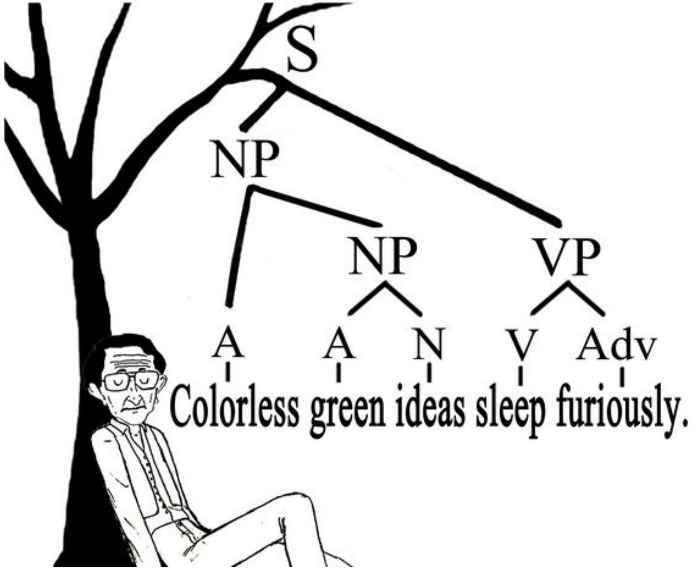
coordination ambiguity is widely spread as well:
[old [men and women]] vs [old men] and [women]

Example, discussion

- There may be several parse trees, this is OK
  (BTW, there are parsers that yield multiple parse trees
  given the text)

- sometimes there is only one ‘true’ parse tree, and this is evident for us, but not for the machine, because **we know word meanings, context and how this world works in general**

- it is also hard thanks to:
  - ellipsis (omission of the word),
  - context-dependent meanings
    (“watch TV”, “how come they got into TV”),
  - morphological ambiguity
    (“river flow”, “the river can flow”)

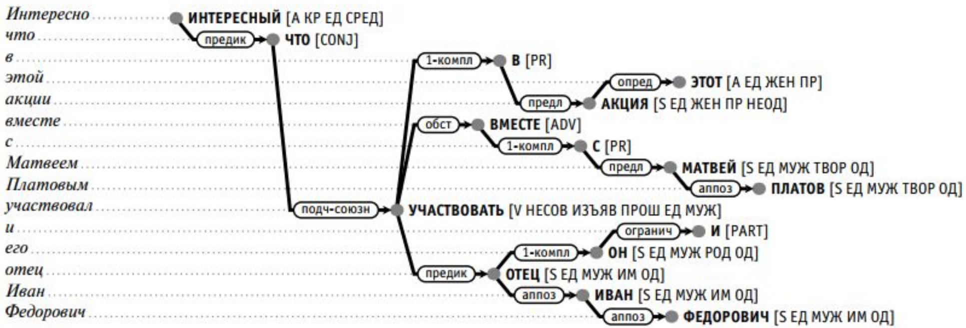


http://www.jnanam.net/slade/Trees/colourless_green_ideas.jpg?w=240

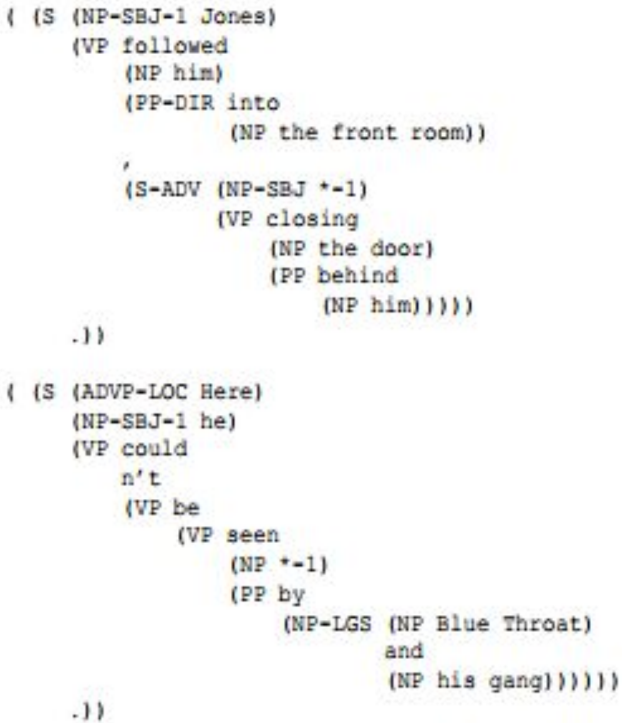
Where the rules come from

- Experts (weak, expensive)
- Annotated data (great, also expensive)

Data banks, where the sentences are parsed



http://ruscorpora.ru/syntax/2013/ataman_vikhr_11.pdf



```
( (S (NP-SBJ-1 Jones)
     (VP followed
         (NP him)
         (PP-DIR into
                 (NP the front room))
         ,
         (S-ADV (NP-SBJ *-1)
                (VP closing
                    (NP the door)
                    (PP behind
                        (NP him)))))
     .))

( (S (ADVP-LOC Here)
     (NP-SBJ-1 he)
     (VP could
         n't
         (VP be
             (VP seen
                 (NP *-1)
                 (PP by
                     (NP-LGS (NP Blue Throat)
                             and
                             (NP his gang))))))
     .))
```

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.8216&rep=rep1&type=pd

Plan

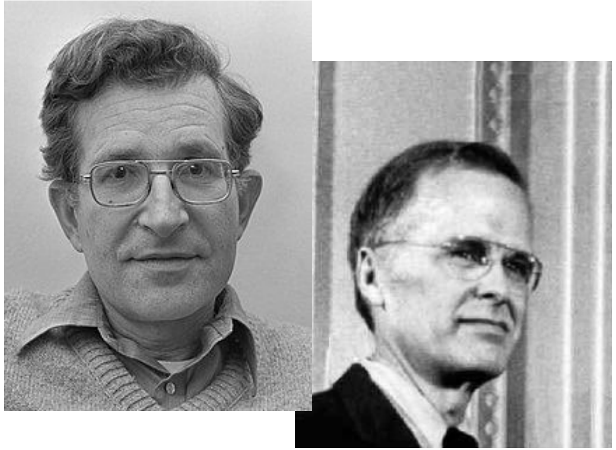
1. ~~What is parsing and why we need it~~
2. Phrase structure grammar
    a. Intuition
    b. Formal grammars
    c. CKY-algorithms
    d. *Shallow parsing
    e. Probabilistic grammars
    f. Probabilistic grammars lexicalization
    g. Quality evaluation
    h. Tools and data

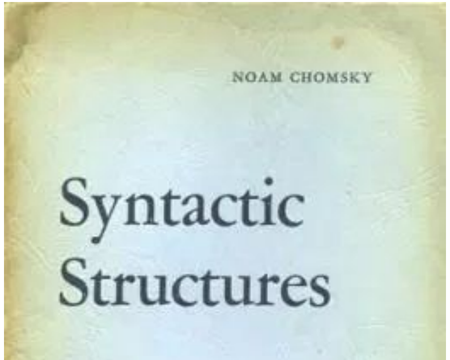
Phrase structure grammar

Key points

- some words are ‘connected more tightly with each other’ than other ones
- the words in a sentence can be grouped into phrases, that ‘behave like a single language entity’
- phrases can be nested

First formulated by Wilhelm Wundt (1900), formalized by **Noam Chomsky** (1956) and by **John Backus** (1959; BNF; independently).



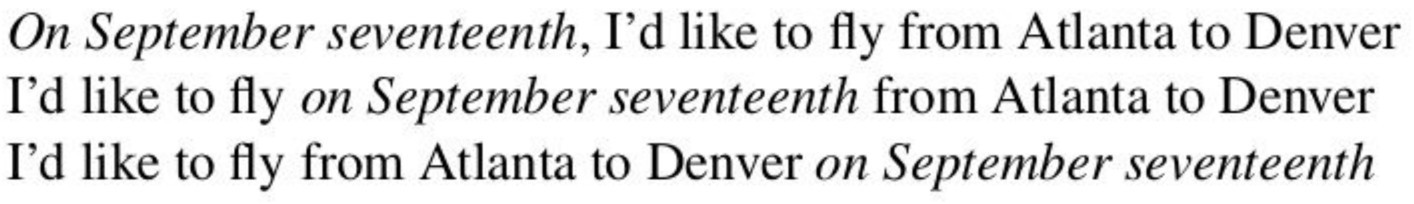


NOAM CHOMSKY

Syntactic Structures

Illustration

Phrases can be swapped

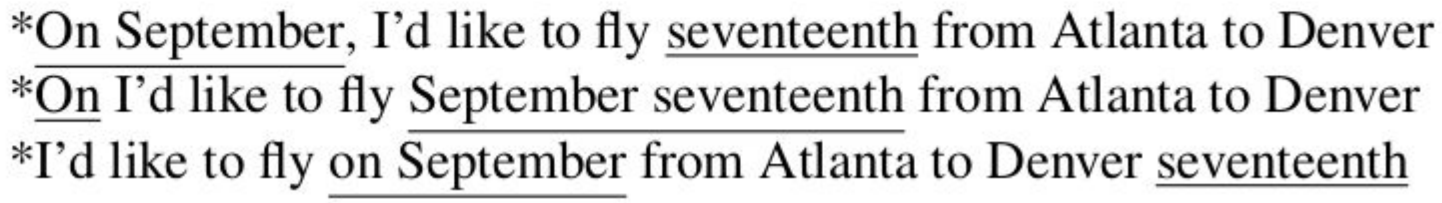


On September seventeenth, I’d like to fly from Atlanta to Denver

I’d like to fly *on September seventeenth* from Atlanta to Denver

I’d like to fly from Atlanta to Denver *on September seventeenth*

yet words forming a phrase sometimes can’t

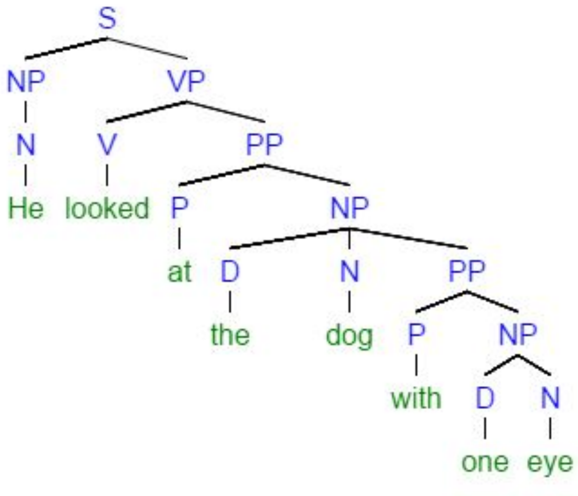


*On September, I’d like to fly seventeenth from Atlanta to Denver

*On I’d like to fly September seventeenth from Atlanta to Denver

*I’d like to fly on September from Atlanta to Denver seventeenth

Parse tree example



https://i.imgur.com/ShMtNEy.png

VP - verb phrase
(approx: a verb and dependent PoS)

NP - noun phrase
(approx: a noun is a root)

PP - prepositional phrase
AP - adjective phrase

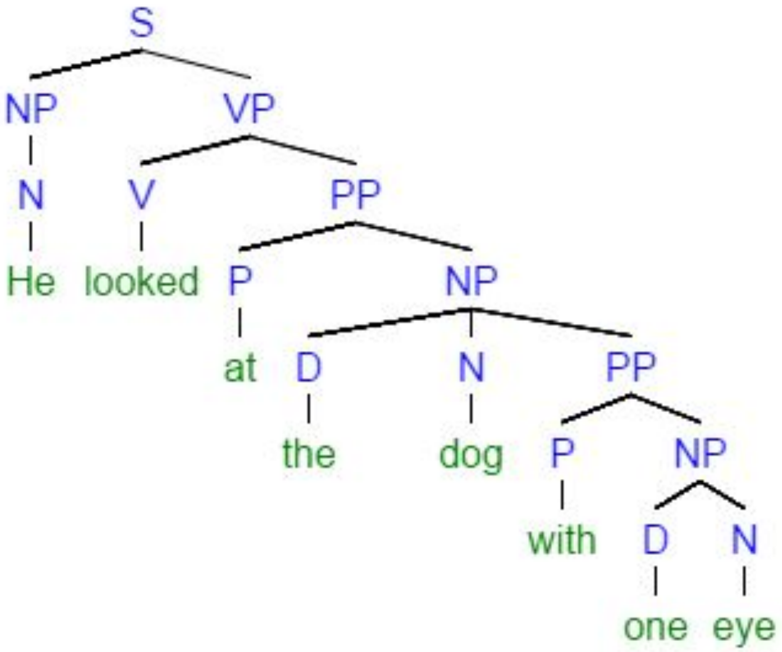
D (Det) - determinatives: articles, certain pronouns, quantifiers, numbers, Q-words, etc.
...

Formal grammar rules

Formal grammar can be treated as a set of rules, which, after a sequence of applications to the initial symbol (**S**), we use to ‘generate’ the text

Parse tree in the example could be built ONLY if the grammar contains this set of rules:

S → NP VP
NP → N
NP → D N
NP → D N PP
VP → V PP
PP → P NP
N → He | dog | eye, V → looked, D → the | one, P → at | with



Plan

1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    a. ~~Intuition~~
    b. Formal grammars
    c. CKY-algorithms
    d. *Shallow parsing
    e. Probabilistic grammars
    f. Probabilistic grammars lexicalization
    g. Quality evaluation
    h. Tools and data

Definition

Formal grammar (aka context-free grammar)
is a tuple of...

N — a set of nonterminal symbols (e.g., *NP, VP, N,...*)
Σ — a set of terminal symbols (all symbols but ones in **N**)
R — a set of production rules of type: **A** → **β** ,
where
A — a nonterminal symbol,
β — a string, an element of the set of all possible strings over **Σ** and **N:** $(\Sigma \cup N)*$
S — a special 'starting' symbol in **N**

Then **the language defined by this grammar** is a set of all strings over **Σ**, that can be deduced from **S** using the production rules: $L = \{w \mid w \text{ is in } \Sigma * \text{ and } S \Rightarrow w\}$

Chomsky Normal Form (CNF)

Any context-free grammar can be converted to the equivalent one (in terms of the defined language) that would contain production rules that would ‘generate’ not more than 2 ‘branches’, e.g.:

$$A \rightarrow B\ C\ D \qquad \longrightarrow \qquad \begin{array}{lcl} A & \rightarrow & B\ X \\ X & \rightarrow & C\ D \end{array}$$

How to parse any CFG

Naive approach: traverse all possible parse trees? :)

Grammars are well studied objects, those who have taken classes on compilers should be very familiar with ones

Program code is long; one needs very effective algorithms to parse it

Sentences in natural languages are usually shorter (though some extremes exist), hence the requirements to parsing speed are lower.
We will take a look at probably the simplest algorithm

Plan

1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    a. ~~Intuition~~
    b. ~~Formal grammars~~
    c. CKY-algorithms
    d. *Shallow parsing
    e. Probabilistic grammars
    f. Probabilistic grammars lexicalization
    g. Quality evaluation
    h. Tools and data

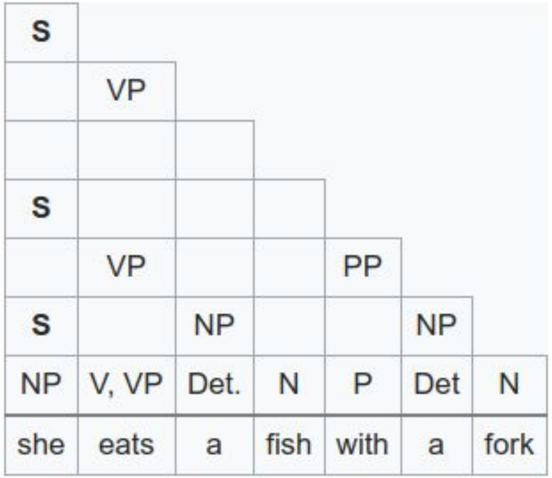
Cocke-Younger-Kasami algorithm (CYK)

From bottom to top, dynamic programming, the grammar should be in the form of CNG

The idea, first approximation: for the sentence of length N, fill the 3D array with markers **p[l, s, a]** determining whether *there is a rule that parses the substring* **a[s:s+l]**

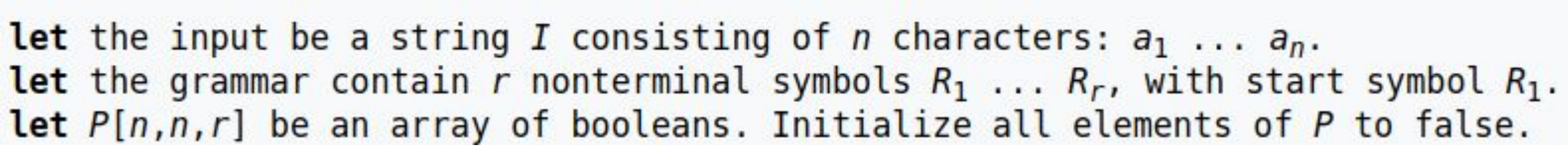
$S \rightarrow NP\ VP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ NP$
$VP \rightarrow eats$
$PP \rightarrow P\ NP$
$NP \rightarrow Det\ N$
$NP \rightarrow she$
$V \rightarrow eats$
$P \rightarrow with$
$N \rightarrow fish$
$N \rightarrow fork$
$Det \rightarrow a$

CYK table



S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

CYK-algorithm

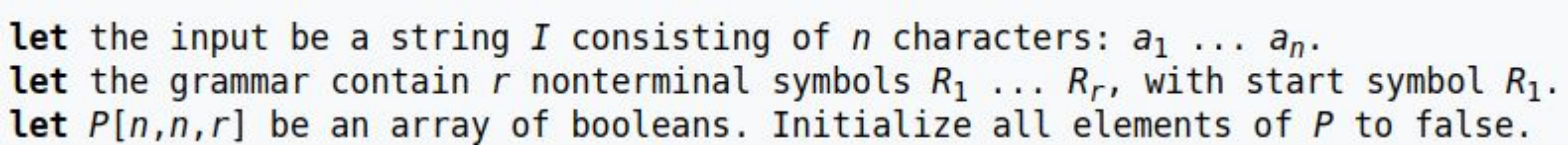


```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of booleans. Initialize all elements of $P$ to false.
```

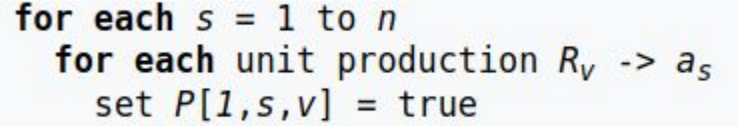
The input is a sentence of length **n**

Boolean 3D array **P** is initialized with **False**

CYK-algorithm



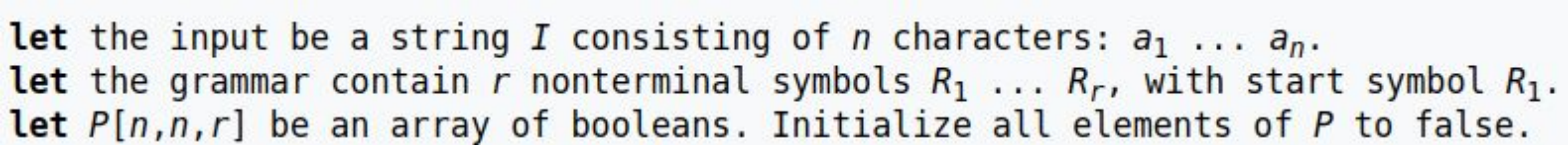
```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of booleans. Initialize all elements of $P$ to false.
```



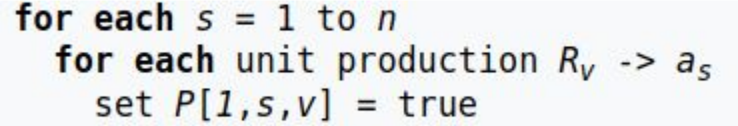
```
for each $s$ = 1 to $n$
  for each unit production $R_v$ -> $a_s$
    set $P[1,s,v]$ = true
```

First we traverse the rules of the kind **A** → **β**, where **β** is a terminal symbol, and set for those **β**s TRUE for the corresponding rules and length = 1

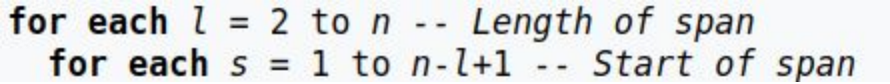
CYK-algorithm



```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of booleans. Initialize all elements of $P$ to false.
```



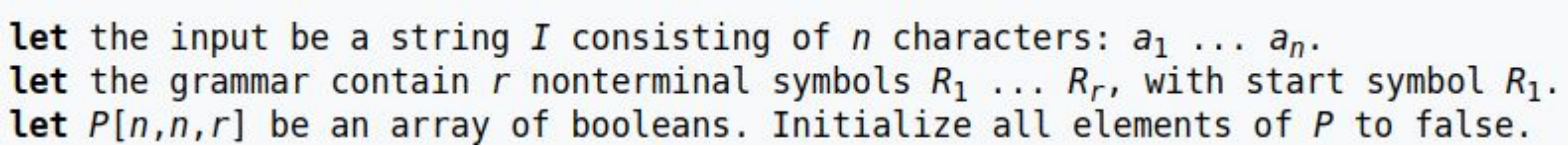
```
for each $s$ = 1 to $n$
  for each unit production $R_v$ -> $a_s$
    set $P[1,s,v]$ = true
```



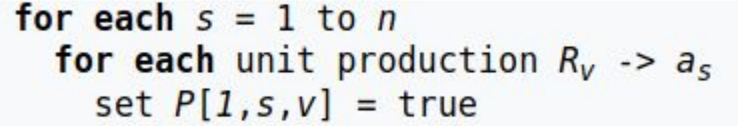
```
for each $l$ = 2 to $n$ -- Length of span
  for each $s$ = 1 to $n-l+1$ -- Start of span
```

For all substrings set by length **l** and starting index **s**

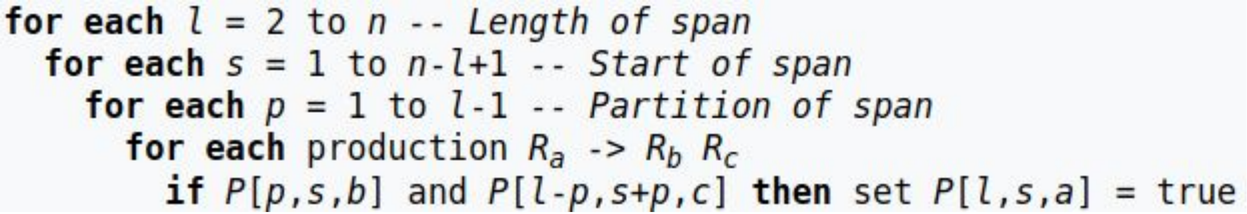
CYK-algorithm



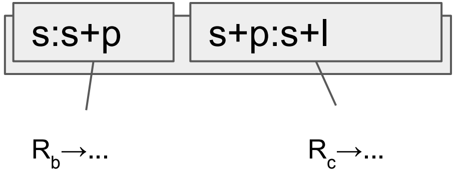
```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of booleans. Initialize all elements of $P$ to false.
```



```
for each $s$ = 1 to $n$
  for each unit production $R_v$ -> $a_s$
    set $P[1,s,v]$ = true
```



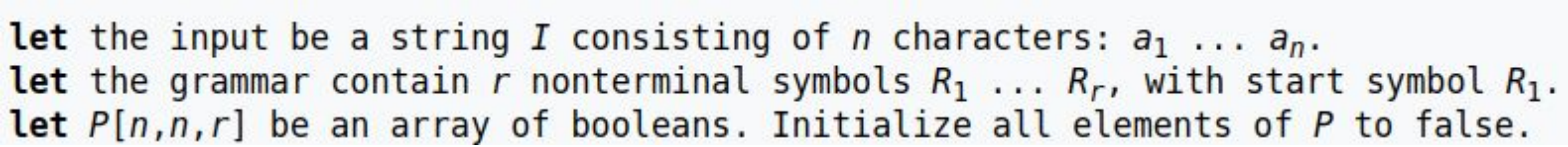
```
for each $l$ = 2 to $n$ -- Length of span
  for each $s$ = 1 to $n-l+1$ -- Start of span
    for each $p$ = 1 to $l-1$ -- Partition of span
      for each production $R_a$ -> $R_b$ $R_c$
        if $P[p,s,b]$ and $P[l-p,s+p,c]$ then set $P[l,s,a]$ = true
```



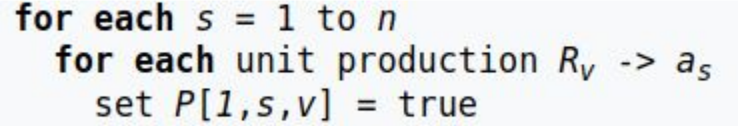
...and all possible splits into two substrings we check if there is such a way to parse each of the two substrings so that their ‘heads’ (here: R_b and R_c) are in the right side of some rule (here: $R_a \rightarrow R_b R_c$)

If yes, set the corresponding array element to **True**.

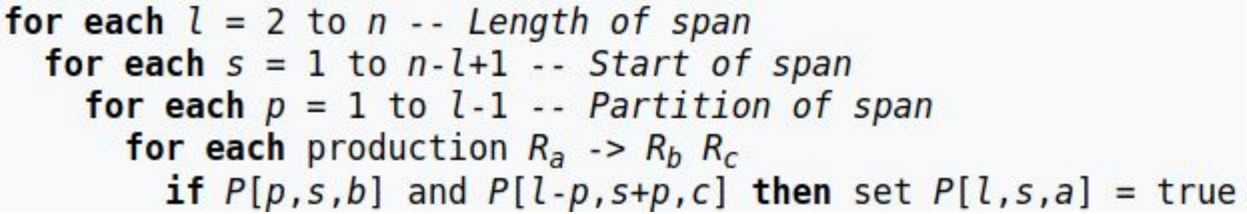
CYK-algorithm



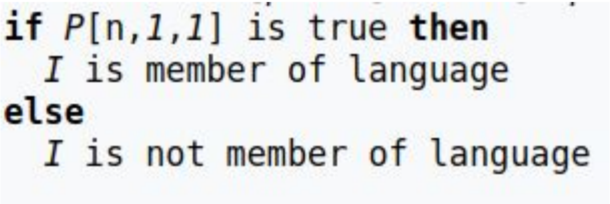
```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of booleans. Initialize all elements of $P$ to false.
```



```
for each $s$ = 1 to $n$
  for each unit production $R_v$ -> $a_s$
    set $P[1,s,v]$ = true
```



```
for each $l$ = 2 to $n$ -- Length of span
  for each $s$ = 1 to $n-l+1$ -- Start of span
    for each $p$ = 1 to $l-1$ -- Partition of span
      for each production $R_a$ -> $R_b$ $R_c$
        if $P[p,s,b]$ and $P[l-p,s+p,c]$ then set $P[l,s,a]$ = true
```



```
if $P[n,1,1]$ is true then
  $I$ is member of language
else
  $I$ is not member of language
```

CYK-algorithm: discussion

- As in Viterbi algorithms, we can store backpointers and do a backward pass to recover **all possible parse trees**
- There may exist several solutions, usually we need **just one**
- For the analysis we have to ‘denormalize’ CFG back from CNF
- Cubic complexity: **$O(n^3 |G|)$**
- Complexity for parsing with arbitrary CFGs can be reduced in terms of the ‘big O’, e.g. using the fast matrix product

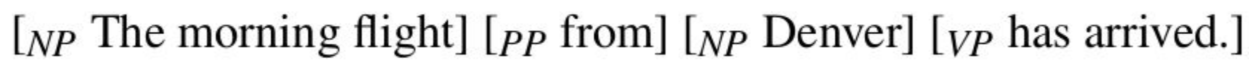
  Valiant, Leslie G. (1975). "General context-free recognition in less than cubic time". J. Comput. Syst. Sci. 10 (2): 308–314.

Plan

1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    1. ~~Intuition~~
    2. ~~Formal grammars~~
    3. ~~CKY-algorithms~~
    4. *Shallow parsing
    5. Probabilistic grammars
    6. Probabilistic grammars lexicalization
    7. Quality evaluation
    8. Tools and data

BTW: shallow parsing

Sometimes for practical tasks shallow parsing is enough, e.g. **chunking** extracting several non-intersecting phrases

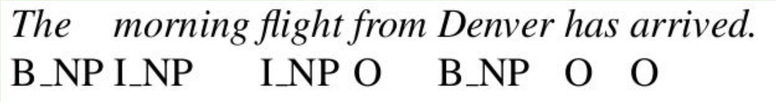


[$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived.]

E.g. we want to extract noun phrases:

[$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived.

This can even be treated as a sequence learning task



The	*morning*	*flight*	*from*	*Denver*	*has*	*arrived.*
B_NP	I_NP	I_NP	O	B_NP	O	O

To evaluate all this, we compute precision, recall and f-measure for the selected chunks, taking exact borders matches + tags matches

Plan

1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    a. ~~Intuition~~
    b. ~~Formal grammars~~
    c. ~~CKY-algorithms~~
    d. ~~*Shallow parsing~~
    e. Probabilistic grammars
    f. Probabilistic grammars lexicalization
    g. Quality evaluation
    h. Tools and data

Probabilistic Context-Free Grammars (PGFG)

...are a way to choose the best possible parse; we'll assign probability to each production rule

N — a set of nonterminal symbols (e.g., *NP, VP, N,...*)
Σ — a set of terminal symbols (all symbols but ones in **N**)
R — a set of production rules of type: **A** → **β** **[p]**,
where
- **A** — a nonterminal symbol,
- **β** — a string, an element of the set of all possible strings over **Σ** and **N**: $(\Sigma \cup N)^*$
- **p(β|A)** — a number between 0 and 1, such that $\sum_{\beta} p(\beta|A) = 1$

S — a special 'starting' symbol in **N**

PCFG is consistent if the sum of probabilities of all possible sentences in the language in concern =1

PCFG in action

The probability of the parse tree **T** for the sentence **S** is a product of all production rules that ‘were applied during the generation of the sentence’

$$P(T, S) = \prod_{i=1}^{n} P(RHS_i | LHS_i)$$

also we know that

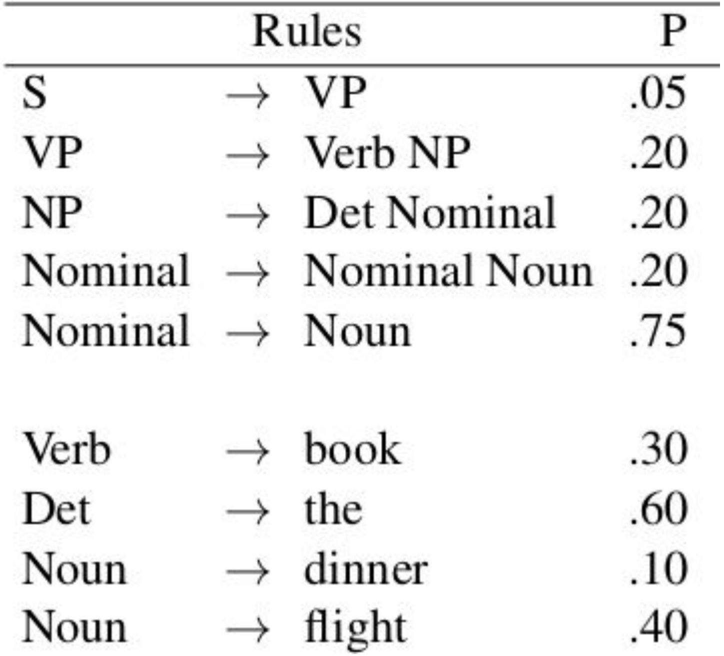
$$P(T, S) = P(T)P(S|T)$$

and **P(S|T) = 1**, because the parse contains all sentence’s words **S**, so

$$P(T, S) = P(T)P(S|T) = P(T)$$

PCFG in action

This is what it looks like



Rules			P
S	→	VP	.05
VP	→	Verb NP	.20
NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20
Nominal	→	Noun	.75
Verb	→	book	.30
Det	→	the	.60
Noun	→	dinner	.10
Noun	→	flight	.40

Having multiplied conditional probabilities we get probabilities of parse trees P(T) and choose the most probable one

But what we need is

$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\operatorname{argmax}} P(T|S)$$

However, if we use Bayes rule and the fact that P(S,T) = P(T) it is easy to show this is correct

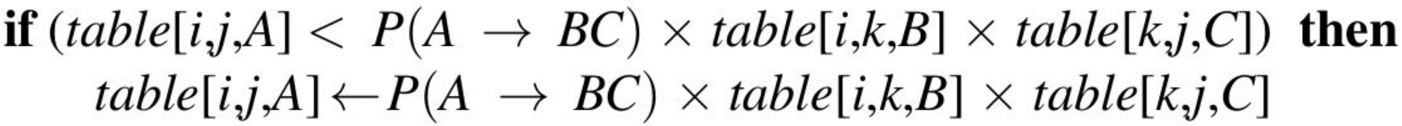
All trees T generating sentence S

Training: CYK again

Exactly the same algorithm, but

- boolean-valued array P => probabilities

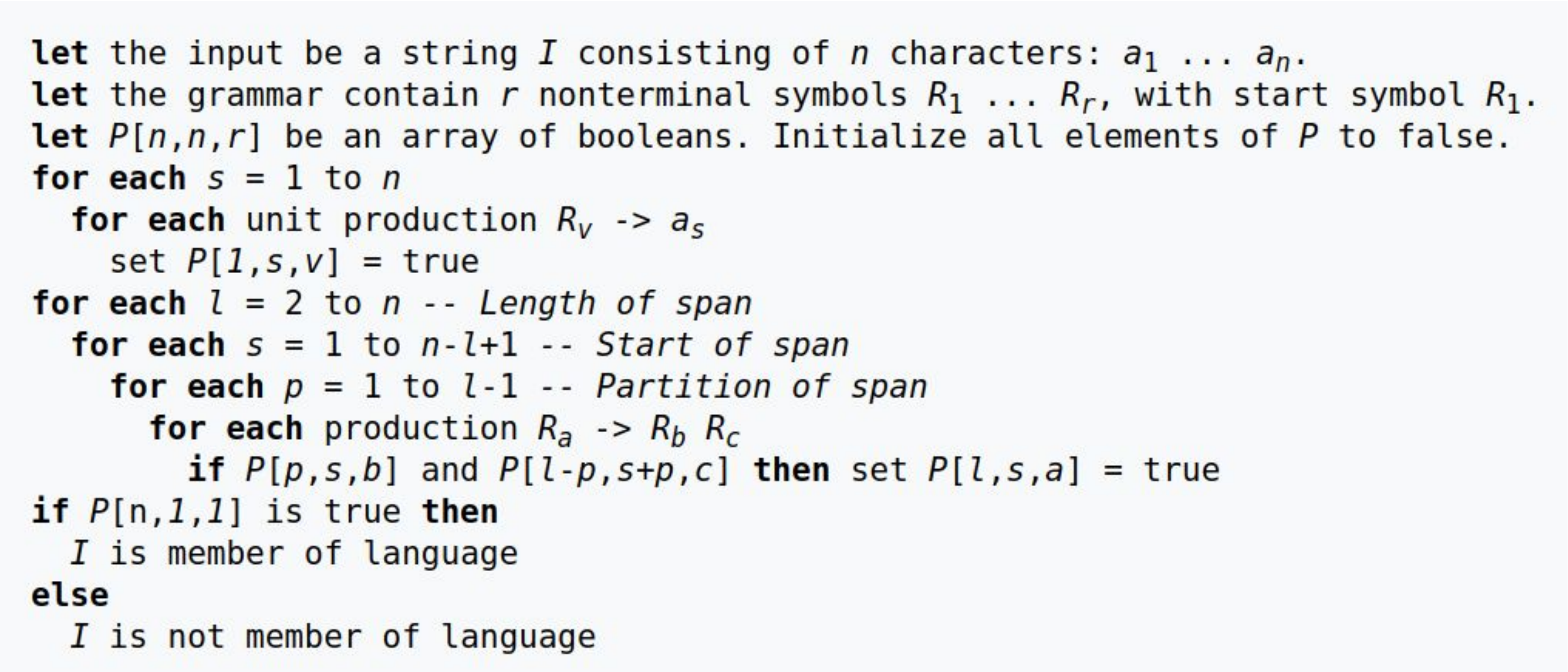
- setting **True** for any good split =>
  updating the probability if it’s greater than the current value



```
if ($table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$) then
    $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
```

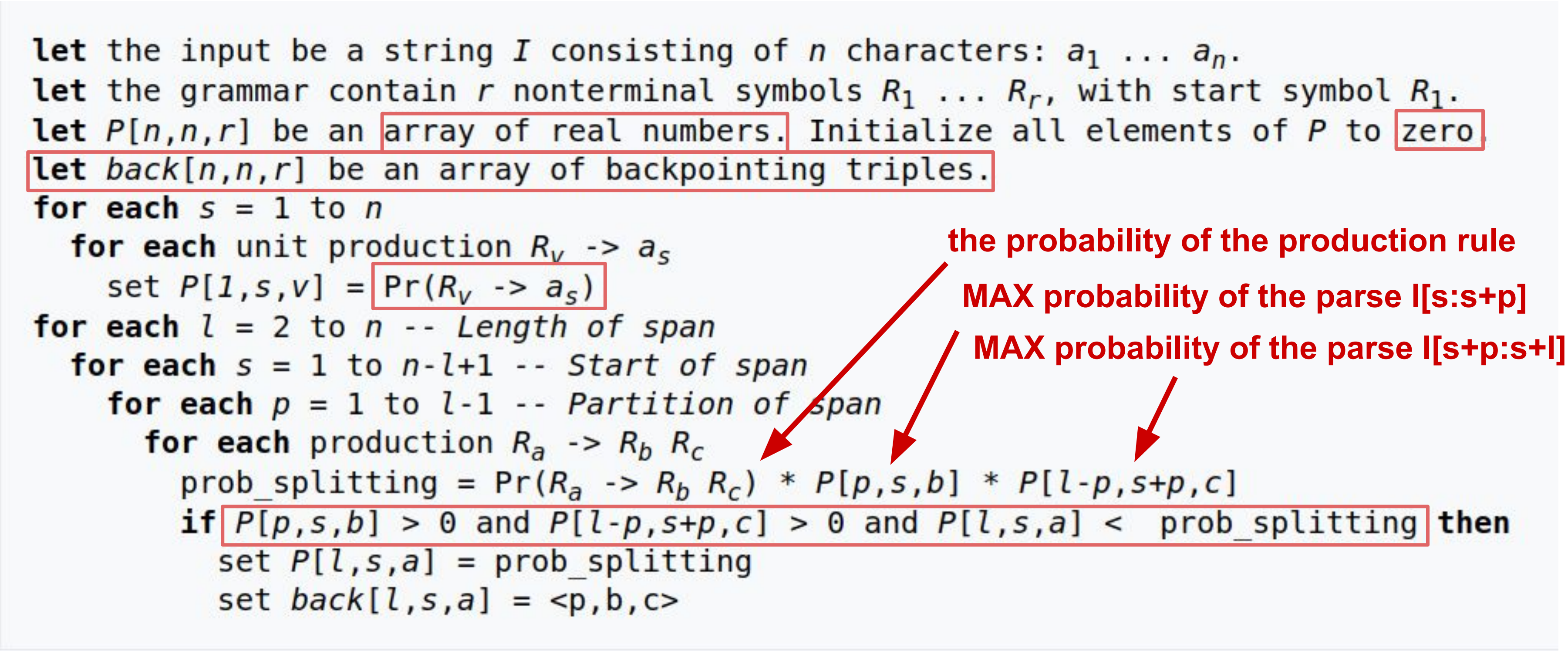
Ney, H. (1991). Dynamic programming parsing for context- free grammars in continuous speech recognition. IEEE Transactions on Signal Processing, 39(2), 336–340.

CYK: before



```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of booleans. Initialize all elements of $P$ to false.
for each $s$ = 1 to $n$
  for each unit production $R_v$ -> $a_s$
    set $P[1,s,v]$ = true
for each $l$ = 2 to $n$ -- Length of span
  for each $s$ = 1 to $n$-$l$+1 -- Start of span
    for each $p$ = 1 to $l$-1 -- Partition of span
      for each production $R_a$ -> $R_b$ $R_c$
        if $P[p,s,b]$ and $P[l-p,s+p,c]$ then set $P[l,s,a]$ = true
if $P[n,1,1]$ is true then
  $I$ is member of language
else
  $I$ is not member of language
```

CYK: after



```
let the input be a string $I$ consisting of $n$ characters: $a_1$ ... $a_n$.
let the grammar contain $r$ nonterminal symbols $R_1$ ... $R_r$, with start symbol $R_1$.
let $P[n,n,r]$ be an array of real numbers. Initialize all elements of $P$ to zero.
let $back[n,n,r]$ be an array of backpointing triples.
for each $s$ = 1 to $n$
  for each unit production $R_v$ -> $a_s$
    set $P[1,s,v]$ = Pr($R_v$ -> $a_s$)
for each $l$ = 2 to $n$ -- Length of span
  for each $s$ = 1 to $n$-$l$+1 -- Start of span
    for each $p$ = 1 to $l$-1 -- Partition of span
      for each production $R_a$ -> $R_b$ $R_c$
        prob_splitting = Pr($R_a$ -> $R_b$ $R_c$) * $P[p,s,b]$ * $P[l-p,s+p,c]$
        if $P[p,s,b]$ > 0 and $P[l-p,s+p,c]$ > 0 and $P[l,s,a]$ <  prob_splitting then
          set $P[l,s,a]$ = prob_splitting
          set $back[l,s,a]$ = <p,b,c>
```

the probability of the production rule

MAX probability of the parse I[s:s+p]

MAX probability of the parse I[s+p:s+l]

Where do we get probabilities from?

Where do we get probabilities from?

Obtain a treebank and compute production rules applications frequencies

$$P(\alpha \to \beta | \alpha) = \frac{\text{Count}(\alpha \to \beta)}{\sum_{\gamma} \text{Count}(\alpha \to \gamma)} = \frac{\text{Count}(\alpha \to \beta)}{\text{Count}(\alpha)}$$

Also the approaches incrementally updating rules probabilities exist

PCFG: discussion

\+ Usable way to train grammar parsers using a corpus
\+ Works better than previous approaches
\+ The effective parsing algorithms exist

- Independence assumptions are too strong
  (still many errors)
- Weak expressiveness:
  - it is useful to take ‘connections types’ into account: subject/object
  - many rules and regularities are connected with certain words

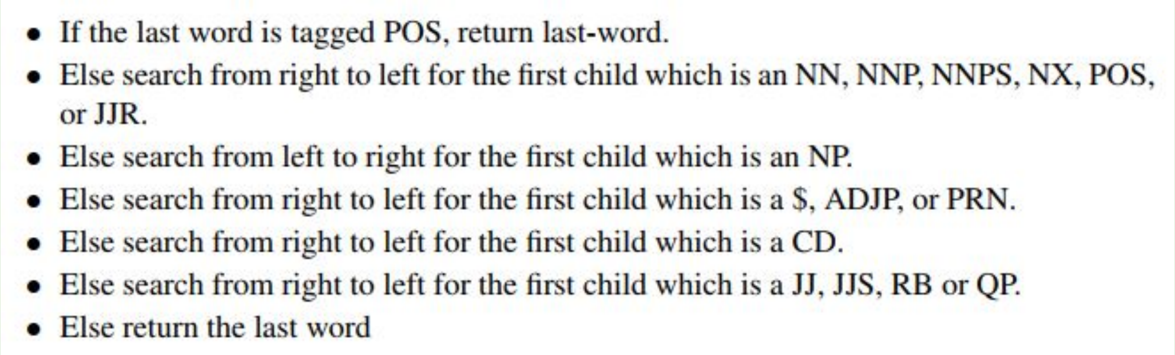
Plan

1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    a. ~~Intuition~~
    b. ~~Formal grammars~~
    c. ~~CKY-algorithms~~
    d. ~~*Shallow parsing~~
    e. ~~Probabilistic grammars~~
    f. Probabilistic grammars lexicalization
    g. Quality evaluation
    h. Tools and data

Head of a phrase

It is important to be able to determine which element (non-terminal symbol) of the phrase is the main one

It is a discussed and a non-trivial task, despite it may not seem to be so.
In English it is well-solved by the **rules** of the sort:



- If the last word is tagged POS, return last-word.
- Else search from right to left for the first child which is an NN, NNP, NNPS, NX, POS, or JJR.
- Else search from left to right for the first child which is an NP.
- Else search from right to left for the first child which is a $, ADJP, or PRN.
- Else search from right to left for the first child which is a CD.
- Else search from right to left for the first child which is a JJ, JJS, RB or QP.
- Else return the last word

Michael Collins’ dissertation
Collins, M. (1999). Head-Driven Statistical Models for Natural Language Parsing. Ph.D. thesis, University of Pennsylvania, Philadelphia. стр 238

Why did we talk about that? Lexicalization!

Phrase head elements can serve as a helpful context!

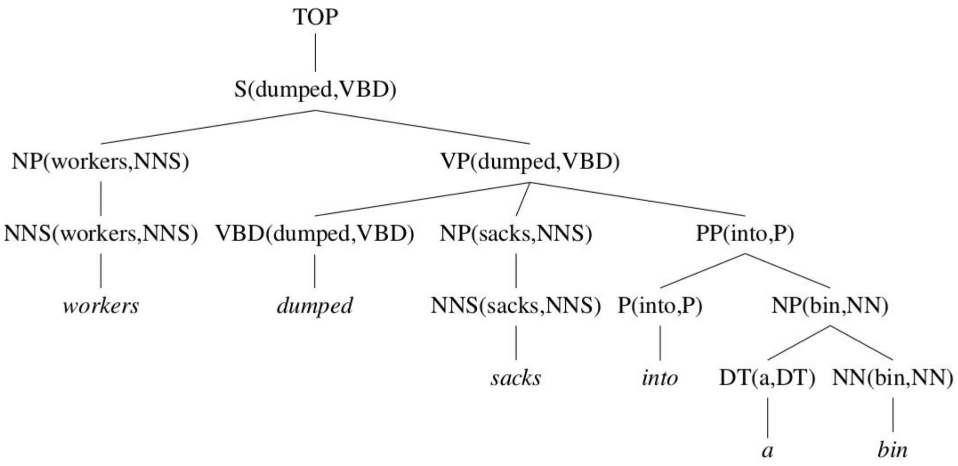
We can set it by appending carefully chosen terminal symbols (and head non-terminal symbols) to non-terminal ones

Probabilistic Lexicalized CFGs
Charniak's and Collins' parsers

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In AAAI-97, pp. 598–603. AAAI Press.

Collins, M. (1999). Head-Driven Statistical Models for Natural Language Parsing. Ph.D. thesis, University of Pennsylvania, Philadelphia.

For more details — Martin, Jurafsky, Collins books and materials



Plan

1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    a. ~~Intuition~~
    b. ~~Formal grammars~~
    c. ~~CKY-algorithms~~
    d. ~~*Shallow parsing~~
    e. ~~Probabilistic grammars~~
    f. ~~Probabilistic grammars lexicalization~~
    g. Quality evaluation
    h. Tools and data

Quality evaluation

Suppose we have the gold standard for parse trees, let us look at the **exact match of phrases borders and labels**:

$$\textbf{labeled recall:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of correct constituents in reference parse of } s}$$

$$\textbf{labeled precision:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in hypothesis parse of } s}$$

...that is, as usual, fraction of correctly predicted items among true ones / all predicted respectively

cross-brackets: cases of the kind
((A B) C) in a “gold standard”
(A (B C)) in prediction

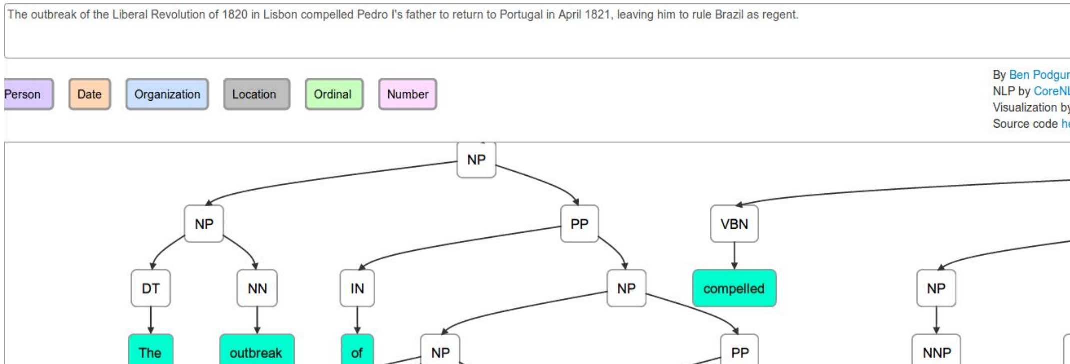
If labels themselves are not important, one can use other evaluation methods

Plan

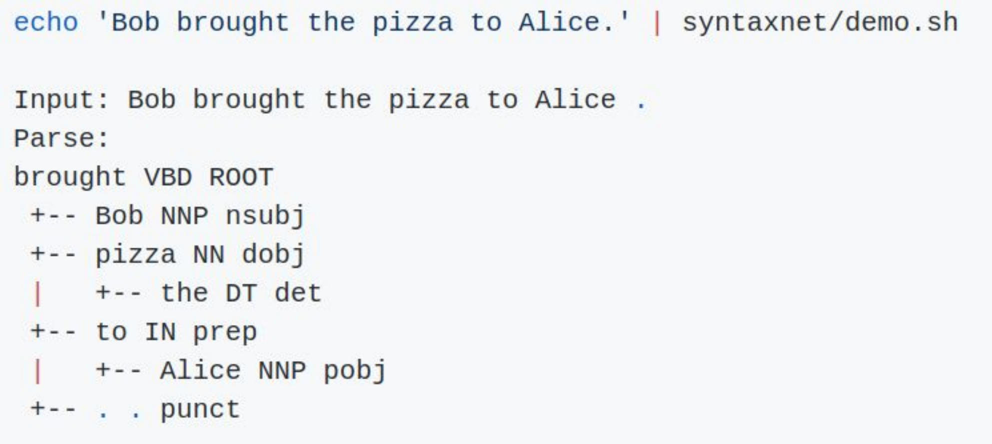
1. ~~What is parsing and why we need it~~
2. ~~Phrase structure grammar~~
    a. ~~Intuition~~
    b. ~~Formal grammars~~
    c. ~~CKY-algorithms~~
    d. ~~*Shallow parsing~~
    e. ~~Probabilistic grammars~~
    f. ~~Probabilistic grammars lexicalization~~
    g. ~~Quality evaluation~~
    h. Tools and data

Instruments

- nltk
- pattern
- spaCy, CoreNLP, CleanNLP (are said to be fast)
- MATE
- SyntaxNet



The outbreak of the Liberal Revolution of 1820 in Lisbon compelled Pedro I's father to return to Portugal in April 1821, leaving him to rule Brazil as regent.



```
echo 'Bob brought the pizza to Alice.' | syntaxnet/demo.sh

Input: Bob brought the pizza to Alice .
Parse:
brought VBD ROOT
 +-- Bob NNP nsubj
 +-- pizza NN dobj
 |   +-- the DT det
 +-- to IN prep
 |   +-- Alice NNP pobj
 +-- . . punct
```

Data

Syntax formalism

https://en.wikipedia.org/wiki/Treebank#Syntactic_treebanks

Dutch	LASSY Small and Large	Dependency	License fee
English	Penn Treebank	Phrase structure	Linguistic Data Consortium
English	CCGbank	Combinatory categorial grammar	Linguistic Data Consortium
English	Prague English Dependency Treebank	Dependency	Linguistic Data Consortium
English	Universal Dependencies	Dependency	Open source (Creative Commons license or GNU general public license)
English	BLLIP WSJ corpus	Phrase structure	Linguistic Data Consortium
English	British Component of the International Corpus of English (ICE-GB)	Phrase structure	License fee
English	Diachronic Corpus of Present-Day Spoken English (DCPSE)	Phrase structure	License fee
English	Lancaster Parsed Corpus	Phrase structure	?
English	Susanne Corpus	Phrase structure	Freely available for research
English	Christine Corpus	Phrase structure	Freely available for research
English	Lucy Corpus	Phrase structure	Freely available for research
English	Tübingen Treebank of English / Spontaneous Speech (TüBa-E/S)	HPSG	Freely available for research
English	LinGO Redwoods	HPSG	?
English	Multi-Treebank	Phrase structure	Available online for comparison purposes

Used and recommended materials

1. Martin-Jurafsky, Chapters 11-13
2. Introduction to Automata Theory, Languages, and Computation
   John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
3. Jarkko Kari, Automata and formal languages, notes
4. [Russian] Прикладная и компьютерная лингвистика
   (под ред. И.С. Николаева, О.В. Митрениной, Т.М. Ландо)
5. [Russian] Введение в общий синтаксис. Я.Г. Тестелец.

Syntax — I

Anton Alekseev
Steklov Mathematical Institute in St Petersburg

NRU ITMO, St Petersburg, 2019
anton.m.alexeyev+itmo@gmail.com

Thanks for comments and advice go to
Denis Kiryanov and Mikhail Slabodkin

http://bit.ly/2hPyEMH