Einführung in die Computerlinguistik
CYK Parsing and Context-Free Grammars

Yihong Liu

2024-22-01

Center for Information and Language Processing

Syntactic Constituents

- Sequential models like HMMs (regular grammars, etc.) assume a linear structure
- But language clearly isn't like that

Example

[*A man*] [*saw* [*a dog*] [*in* [*the park*]]]

- Words **group** together to form syntactic constituents
  - Can be replaced, or moved around *as a **unit***
- **Grammars** allow us to formalize these intuitions
  - **Symbols** correspond to syntactic constituents

Outline

- The context-free grammar formalism
- CYK parsing

Basics of Context-free grammars

Symbols

- **Terminal**: actual word such as book
- **Non-terminal**: syntactic label such as *NP* or *NN*
- Convention to use upper and lower-case to distinguish, or else “quotes” for terminals

Productions (rules)

W $\rightarrow$ XYZ

- Exactly **one** non-terminal on left-hand side (LHS)
- An **ordered** list of symbols on right-hand side (RHS) – can be Terminals or Non-terminals

CFG symbols

Start symbol — often called S (as the “sentence” node)

Example: Lexicon

Noun →	*flights* | *flight* | *breeze* | *trip* | *morning*
Verb →	*is* | *prefer* | *like* | *need* | *want* | *fly* | *do*
Adjective →	*cheapest* | *non-stop* | *first* | *latest*
	| *other* | *direct*
Pronoun →	*me* | *I* | *you* | *it*
Proper-Noun →	*Alaska* | *Baltimore* | *Los Angeles*
	| *Chicago* | *United* | *American*
Determiner →	*the* | *a* | *an* | *this* | *these* | *that*
Preposition →	*from* | *to* | *on* | *near* | *in*
Conjunction →	*and* | *or* | *but*

Example: Grammar rules

Grammar Rules			Examples
S	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Proper\text{-}Noun$	Los Angeles
	$\mid$	$Det\ Nominal$	a + flight
$Nominal$	$\rightarrow$	$Nominal\ Noun$	morning + flight
	$\mid$	$Noun$	flights
VP	$\rightarrow$	$Verb$	do
	$\mid$	$Verb\ NP$	want + a flight
	$\mid$	$Verb\ NP\ PP$	leave + Boston + in the morning
	$\mid$	$Verb\ PP$	leaving + on Thursday
PP	$\rightarrow$	$Preposition\ NP$	from + Los Angeles

Regular expressions as CFGs

Regular expressions match simple patterns

- For example, words starting with a **capital**:
  *[A-Z][a-z]**

Can rewrite as a grammar (”regular grammar”)

- S → U   S → U LS
- U → ”A”   U → ”B” …   U → ”Z”
- LS → L   LS → L LS
- L → “a”   L → “b” …   L → “z”

The class of regular languages is a **subset** of the context-free languages, which are specified using a CFG

CFGs vs regular grammars

CFGs (and regexs) used to describe a set of strings, aka a "language"

Regular grammars

- describe a **smaller** class of languages
- can be parsed using **finite state machines** (FSA, FST)

CFGs

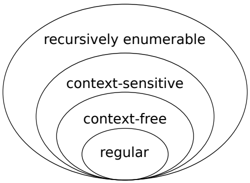
- can describe **hierarchical groupings**
- requires more **complex** automata to parse (PDA: pushdown automaton)

Context sensitive grammars are even more expressive (and **intractable**)

Chomsky hierarchy

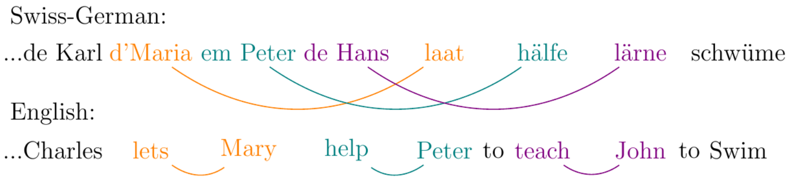
CF languages more general than RLs

- Allows representation of recursive nesting



CF adequate for most constructions in natural language

- but **not** e.g., cross-serial dependencies in Swiss-German



A simple CF grammar

Terminal symbols: *rat, the, ate, cheese*

Non-terminal symbols: S, NP, VP, DT, VBD, NN

Productions:

S $\rightarrow$ NP VP

NP $\rightarrow$ DT NN

VP $\rightarrow$ VBD NP

DT $\rightarrow$ the

NN $\rightarrow$ rat

NN $\rightarrow$ cheese

VBD $\rightarrow$ ate

Generating sentences with CFGs

In each step we rewrite the **left-most non-terminal**

S $\rightarrow$ NP VP
NP $\rightarrow$ DT NN
VP $\rightarrow$ VBD NP
DT $\rightarrow$ the
NN $\rightarrow$ rat
NN $\rightarrow$ cheese
VBD $\rightarrow$ ate

Always start with S (the sentence/start symbol)

S

Apply a rule with S on LHS (S $\rightarrow$ NP VP), i.e substitute RHS

NP VP

Apply a rule with NP on LHS (NP $\rightarrow$ DT NN)

DT NN VP

Apply rule with DT on LHS (DT $\rightarrow$ the)

the NN VP

Apply rule with NN on LHS (NN $\rightarrow$ rat)

the rat VP

Generating sentences with CFGs (cont.)

S $\rightarrow$ NP VP
NP $\rightarrow$ DT NN
VP $\rightarrow$ VBD NP
DT $\rightarrow$ the
NN $\rightarrow$ rat
NN $\rightarrow$ cheese
VBD $\rightarrow$ ate

Apply rule with VP on LHS (VP $\rightarrow$ VBD NP)

the rat VBD NP

Apply rule with VBD on LHS (VBD $\rightarrow$ ate)

the rat ate NP

Apply rule with NP on LHS (NP $\rightarrow$ DT NN)

the rat ate DT NN

Apply rule with DT on LHS (DT $\rightarrow$ the)

the rat ate the NN

Apply rule with NN on LHS (NN $\rightarrow$ cheese)

the rat ate the cheese

No non-terminals left, we're done!

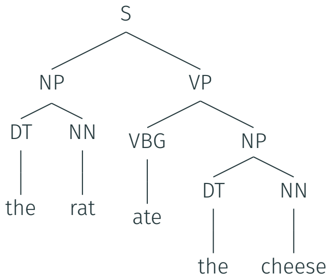
CFG trees

Generation corresponds to a **syntactic tree**

Non-terminals are internal nodes

Terminals are leaves

(S (NP (DT the)
(NN rat))
(VP (VBG ate)
(NP (DT the)
(NN cheese))))

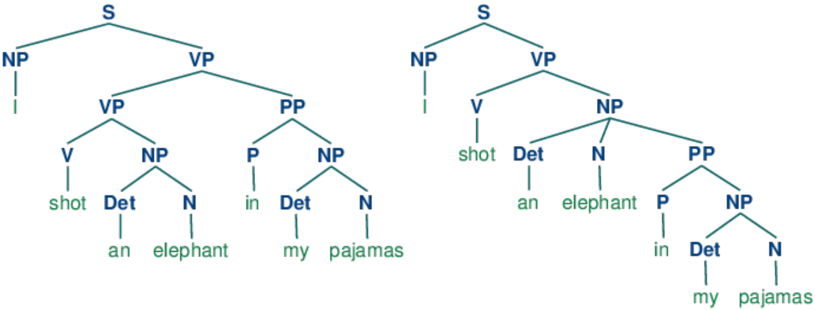


Parsing is the ***reverse*** process

Parse Ambiguity

Often more than one tree can describe a string

"While hunting in Africa, ***I shot an elephant in my pajamas****. How he got into my pajamas, I don't know."*
— Animal Crackers (1930)



Parsing CFGs

Parsing: given **string**, identify possible **structures**

Brute force search is intractable for non-trivial grammars

- Good solutions use **dynamic programming**

Two general strategies

- **Bottom-up** (Start with words, work up towards S; CYK parsing)
- **Top-down** (Start with S, work down towards words; Earley parsing (not covered))

The CYK parsing algorithm

The Cocke–Younger–Kasami algorithm

Steps:

- Convert grammar to **Chomsky Normal Form** (CNF)
- Fill in a parse table
- Use table to derive parse
- Convert result back to original grammar

Convert to Chomsky Normal Form

Change grammar so all rules of form

$A \rightarrow B\ C$ (*B and C cannot be S*) or $A \rightarrow a$

Step 1: Convert rules of form $A \rightarrow B\ c$ into pair of rules $A \rightarrow B\ X$, $X \rightarrow c$

Step 2: Convert rules $A \rightarrow B\ C\ D$ into $A \rightarrow B\ Y$, $Y \rightarrow C\ D$

- Usually necessary, but not for our toy grammar
- E.g., VP $\rightarrow$ VP NP NP for ditransitive cases, *"sold [her] [the book]"*

Step 3: Convert rules $A \rightarrow B\ S$ into $A \rightarrow B\ S'$ and add all the rules of S for S'.

X, Y, S' are new non-terminal symbols we have introduced

CNF (cont)

CNF disallows unary rules, A $\rightarrow$ B. Why?

Imagine NP $\rightarrow$ S; and S $\rightarrow$ NP ... leads to infinitely many trees with **same** yield.

If no cycles, can transform grammar, e.g.,

- if A $\rightarrow$ B and B $\rightarrow$ c and B $\rightarrow$ d then make new non-terminal Z, with rules Z $\rightarrow$ c and Z $\rightarrow$ d; all instances of A in RHS of other rules now also support Z; **or** simply add A $\rightarrow$ c and A $\rightarrow$ d
- common occurrence in formal grammars, e.g., NP $\rightarrow$ NN, VP $\rightarrow$ VB, where NN and VB are pre-terminals (POS tags), and only rewrite as strings

CYK algorithm

function CYKParse($words, grammar$)
  for $col = 1 \ldots len(words)$ **do**
    for all $\{A | A \rightarrow words[col] \in grammar\}$ **do**
      $table[col - 1, col] \leftarrow$ add A
    for $r = col - 2 \ldots 0$ **do**
      for $c = r + 1 \ldots col - 1$ **do**
        for all $\{A | A \rightarrow BC \in grammar$ **and** $B \in table[r, c]$ **and** $C \in table[c, col]\}$ **do**
          $table[r, col] \leftarrow$ add A

words – indexed [1, 2, ..., length(words)]

CYK by example

$S \rightarrow NP\ VP$
$NP \rightarrow DT\ NN$
$VP \rightarrow VBD\ NP$
$DT \rightarrow$ the
$NN \rightarrow$ rat
$NN \rightarrow$ cheese
$VBD \rightarrow$ ate

the	rat	ate	the	cheese
DT 0,1	NP 0,2	0,3	0,4	S 0,5
	NN 1,2	1,3	1,4	1,5
		VBD 2,3	2,4	VP 2,5
			DT 3,4	NP 3,5
				NN 4,5

CYK: Retrieving The parses

S in the top-right corner of parse table indicates **success**

To get parse(s), follow **pointers** back for each match

Convert back from CNF by transforming new non-terminals back to their original values

- E.g., if VP $\rightarrow$ VP NP NP was changed to VP $\rightarrow$ VP NP_NP; NP_NP $\rightarrow$ NP NP
- If we have the latter two productions in tree, transform tree back to top production, i.e., VP $\rightarrow$ VP NP NP

Parse table with backpointers

S → NP VP

NP → DT NN

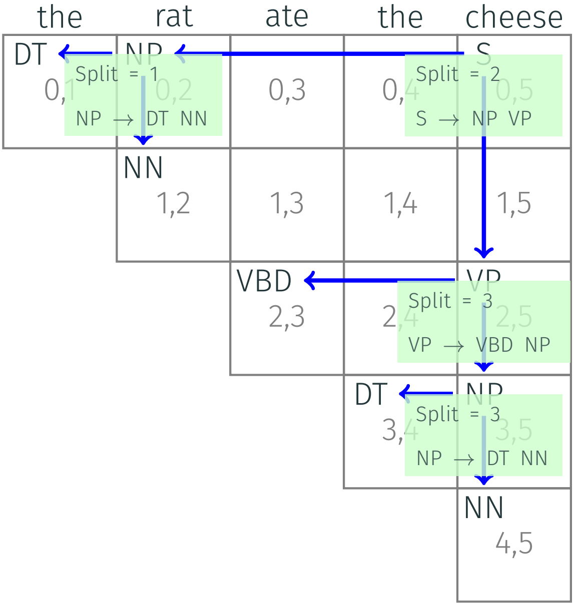
VP → VBD NP

DT → the

NN → rat

NN → cheese

VBD → ate



CYK Properties

CYK returns an efficient representation of **the set of parse trees** for a sentence

Doesn't tell us which parse tree is the right one

For that, we need to augment CKY with **scores** for each possible constituent (e.g., with **neural** span-based parsers.)

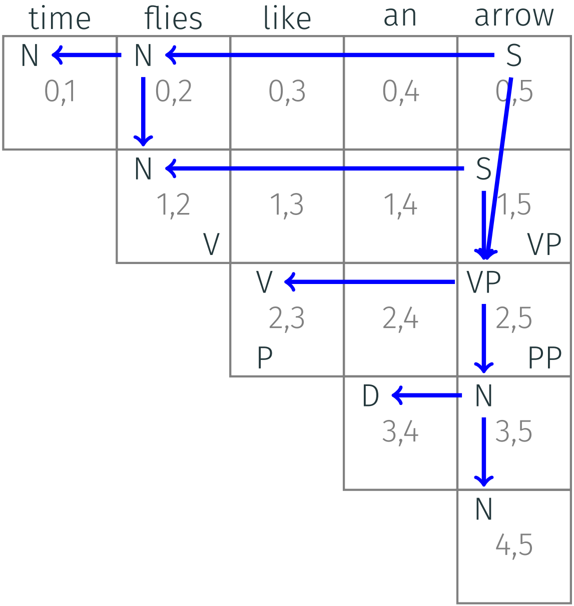
CYK: Ambiguity

$S \to N\ VP$
$VP \to V\ N$
$VP \to V\ PP$
$N \to D\ N$
$N \to N\ N$
$PP \to P\ N$
$N \to$ time
$N \to$ flies
$N \to$ arrow
$V \to$ flies
$V \to$ like
$P \to$ like
$D \to$ an

time	flies	like	an	arrow
N 0,1	N 0,2	0,3	0,4	S 0,5
	N 1,2 V	1,3	1,4	S 1,5 VP
		V 2,3 P	2,4	VP 2,5 PP
			D 3,4	N 3,5
				N 4,5

CYK: Ambiguity

S $\rightarrow$ N VP
VP $\rightarrow$ V N
VP $\rightarrow$ V PP
N $\rightarrow$ D N
N $\rightarrow$ N N
PP $\rightarrow$ P N
N $\rightarrow$ time
N $\rightarrow$ flies
N $\rightarrow$ arrow
V $\rightarrow$ flies
V $\rightarrow$ like
P $\rightarrow$ like
D $\rightarrow$ an



CYK: Ambiguity

$S \to N\ VP$

$VP \to V\ N$

$VP \to V\ PP$

$N \to D\ N$

$N \to N\ N$

$PP \to P\ N$

$N \to$ time

$N \to$ flies

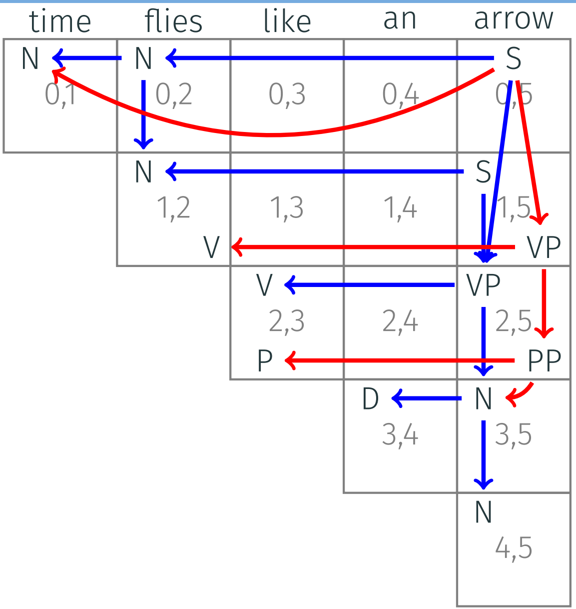
$N \to$ arrow

$V \to$ flies

$V \to$ like

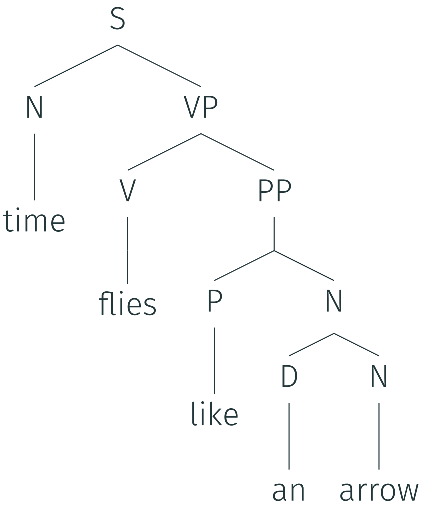
$P \to$ like

$D \to$ an

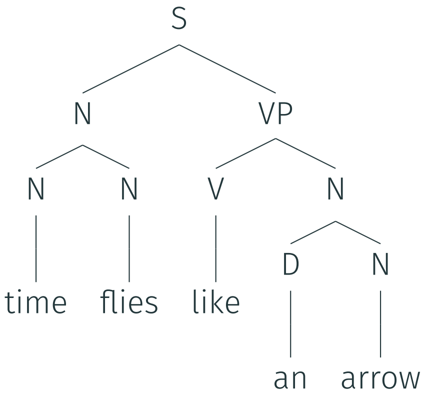


Time flies: Two parses

Red



Blue



From Toy Grammars to Real Grammars

Toy grammars with handful of productions good for **demonstration** or extremely **limited** domains

For real texts, we need **real** grammars

Many thousands of production rules

Key Constituents in Penn Treebank

Sentence (S)

Noun phrase (NP)

Verb phrase (VP)

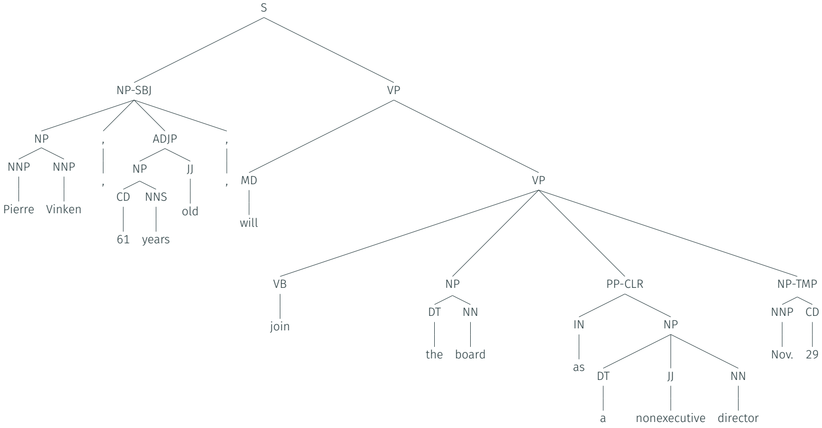
Prepositional phrase (PP)

Adjective phrase (AdjP)

Adverbial phrase (AdvP)

Subordinate clause (SBAR)

Example PTB/0001



A final word

Context-free grammars can represent **linguistic structure**

There are relatively fast **dynamic programming** algorithms to retrieve this structure

But what about **ambiguity**?

Extreme ambiguity will slow down parsing

If multiple possible parses, which is best?

Besonders klausurrelevant

- Formal definition CFG:
  terminals, non-terminals, start symbol, productions
- CYK
- Parse trees

Credits

Adapted from slides by Ivan Habernal, Paderborn University, *https://www.trusthlt.org/*

Based on slides by Trevor Cohn, University of Melbourne, *https://trevorcohn.github.io/comp90042/*

Images:

https://en.wikipedia.org/wiki/Chomsky_hierarchy

https://en.wikipedia.org/wiki/Cross-serial_dependencies

https://web.stanford.edu/~jurafsky/slp3/