Constituency Parsing

Data structures and algorithms
for Computational Linguistics III

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2018–2019

Context free grammars
recap

- Context free grammars are sufficient for expressing most phenomena in natural language syntax
- Most of the parsing theory (and quite some of the practice) is build on parsing CF languages
- The context-free rules have the form

$$A \rightarrow \alpha$$

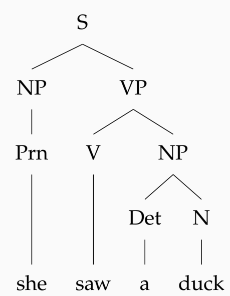
where A is a single non-terminal symbol and α is a (possibly empty) sequence of terminal or non-terminal symbols

An example context-free grammar

S → NP VP
S → Aux NP VP
NP → Det N
NP → Prn
NP → NP PP
VP → V NP
VP → V
VP → VP PP
PP → Prp NP
N → duck
N → park
V → duck
V → ducks
V → saw
Prn → she | her
Prp → in | with
Det → a | the

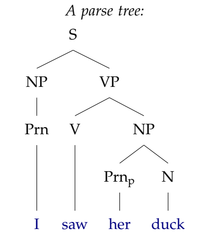
Derivation of sentence 'she saw a duck'

S ⇒ NP VP
NP ⇒ Prn
Prn ⇒ she
VP ⇒ V NP
V ⇒ saw
NP ⇒ Det N
Det ⇒ a
N ⇒ duck



Representations of a context-free parse tree

A parse tree:



A history of derivations:

- S ⇒NP VP
- NP ⇒Prn
- Prn ⇒I
- VP ⇒V NP
- V ⇒saw
- NP ⇒Prn_p N
- Prn_p ⇒her
- N ⇒duck

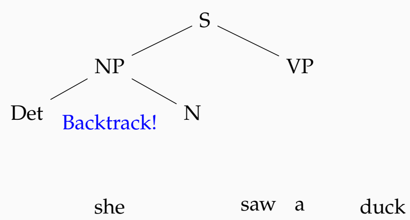
A sequence with (labeled) brackets

$$\left[_S \left[_{NP} [_{Prn}\ I]\right] \left[_{VP} [_V\ saw]\ \left[_{NP} [_{Prn_p}\ her]\ [_N\ duck]\right]\right]\right]$$

Parsing as search

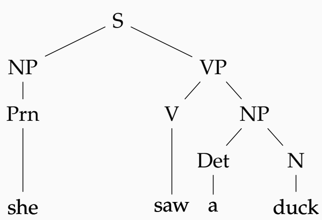
- Parsing can be seen as search constrained by the grammar and the input
- Top down: start from S, find the derivations that lead to the sentence
- Bottom up: start from the sentence, find series of derivations (in reverse) that leads to S
- Search can be depth first or breadth first for both cases

Parsing as search: top down



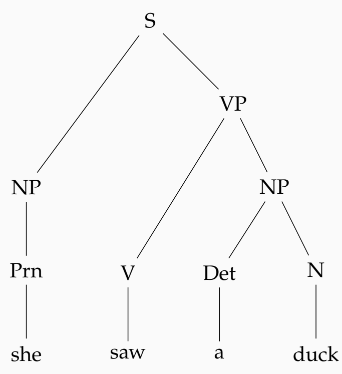
S → NP VP
S → Aux NP VP
NP → Det N
NP → Prn
NP → NP PP
VP → V NP
VP → V
VP → VP PP
PP → Prp NP
N → duck
N → park
V → duck
V → ducks
V → saw
Prn → she | her
Prp → in | with
Det → a | the

Parsing as search: top down



S → NP VP
S → Aux NP VP
NP → Det N
NP → Prn
NP → NP PP
VP → V NP
VP → V
VP → VP PP
PP → Prp NP
N → duck
N → park
V → duck
V → ducks
V → saw
Prn → she | her
Prp → in | with
Det → a | the

Parsing as search: bottom up



S → NP VP
S → Aux NP VP
NP → Det N
NP → Prn
NP → NP PP
VP → V NP
VP → V
VP → VP PP
PP → Prp NP
N → duck
N → park
V → duck
V → ducks
V → saw
Prn → she | her
Prp → in | with
Det → a | the

Problems with search procedures

- Top-down search considers productions incompatible with the input, and cannot handle left recursion
- Bottom-up search considers non-terminals that would never lead to S
- Repeated work because of backtracking

→ The result is exponential time complexity in the length of the sentence

> Some of these problems can be solved using *dynamic programming*.

CKY algorithm

- The CKY (Cocke–Kasami–Younger) parsing algorithm is a dynamic programming algorithm (Kasami 1965; Younger 1967; Cocke and Schwartz 1970)
- It processes the input *bottom up*, and saves the intermediate results on a *chart*
- Time complexity for *recognition* is $O(n^3)$
- Space complexity is $O(n^2)$
- It requires the CFG to be in *Chomsky normal form* (CNF)

Chomsky normal form (CNF)

- A CFG is in CNF, if the rewrite rules are in one of the following forms
  - $A \rightarrow B\ C$
  - $A \rightarrow a$

  where A, B, C are non-terminals and a is a terminal
- Any CFG can be converted to CNF
- Resulting grammar is *weakly equivalent* to the original grammar:
  - it generates/accepts the same language
  - but the derivations are different

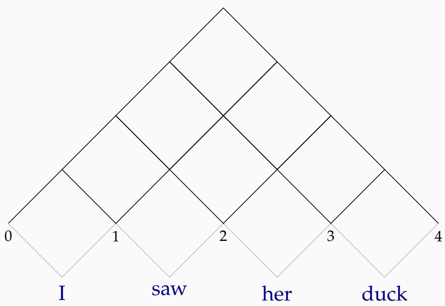
Converting to CNF: example

- For rules with > 2 RHS symbols
  S →Aux NP VP ⇒ S →Aux X
  X →NP VP
- For rules with < 2 RHS symbols
  NP →Prn ⇒ NP → she | her

S → NP VP
S → Aux NP VP
NP → Det N
NP → Prn
NP → NP PP
VP → V NP
VP → V
VP → VP PP
PP → Prp NP
N → duck
N → park
V → duck
V → ducks
V → saw
Prn → she | her
Prp → in | with
Det → a | the

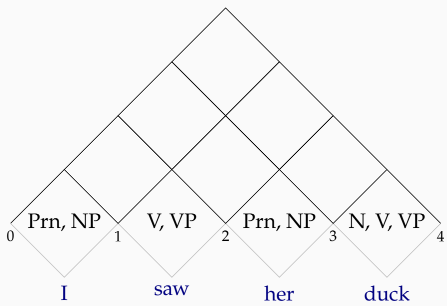
CKY demonstration

an ambiguous example



CKY demonstration

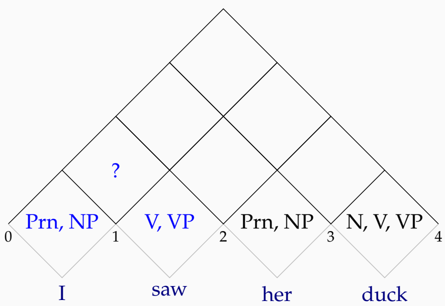
an ambiguous example



CKY demonstration

an ambiguous example

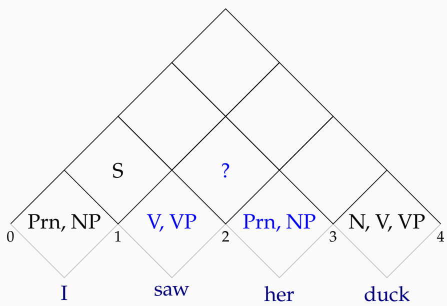
S → NP VP



CKY demonstration

an ambiguous example

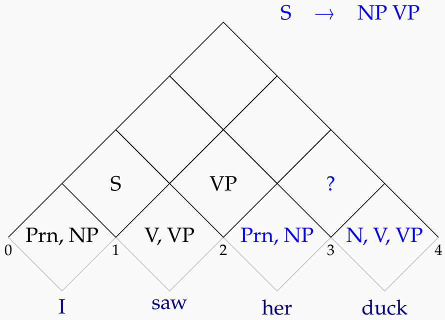
VP → V NP



CKY demonstration

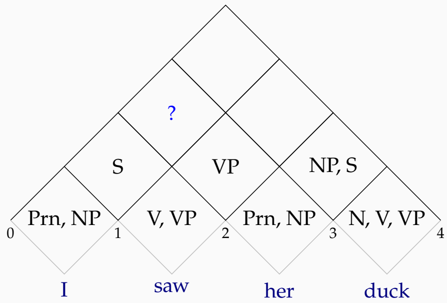
an ambiguous example

NP → Prn N
S → NP VP



CKY demonstration

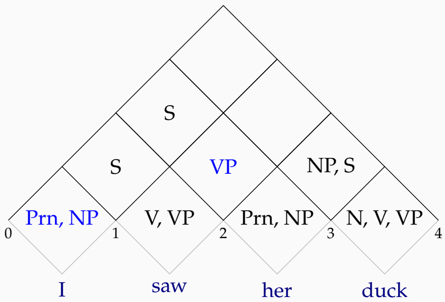
an ambiguous example



CKY demonstration

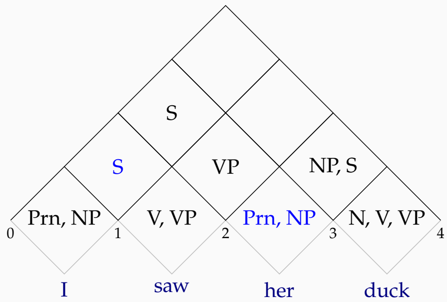
an ambiguous example

S → NP VP



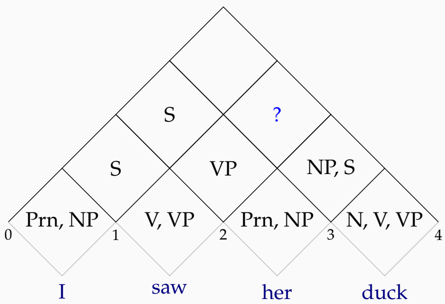
CKY demonstration

an ambiguous example



CKY demonstration

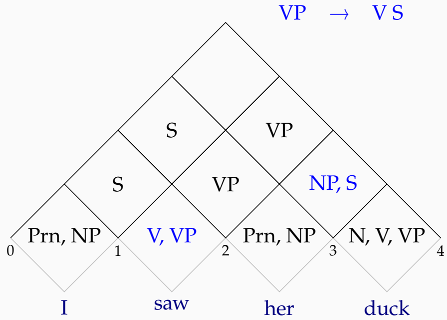
an ambiguous example



CKY demonstration

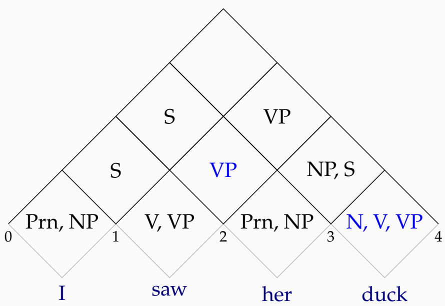
an ambiguous example

VP → V NP
VP → V S



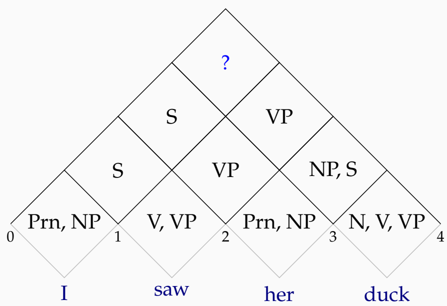
CKY demonstration

an ambiguous example



CKY demonstration

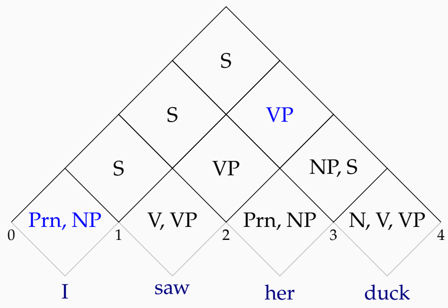
an ambiguous example



CKY demonstration

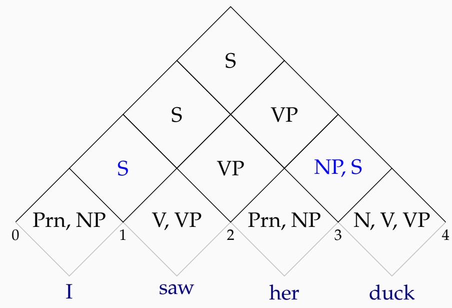
an ambiguous example

S → NP VP



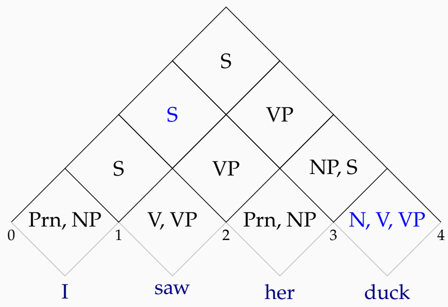
CKY demonstration

an ambiguous example



CKY demonstration

an ambiguous example



CKY demonstration: the chart

NP, Prn	S	S	S
	V, VP	VP	VP
		Prn	NP, S
			V, N, NP

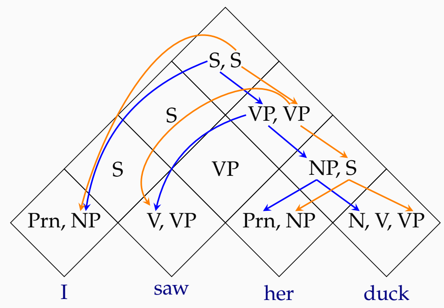
0 she 1 saw 2 her 3 duck 4

Chart is a 2-dimensional array: $O(n^2)$ space complexity.

Parsing vs. recognition

- We went through a recognition example
- Recognition accepts or rejects a sentence based on a grammar
- For parsing, we want to know the derivations that yielded a correct parse
- To recover parse trees, we
  - we follow the same procedure as recognition
  - add back links to keep track of the derivations

Chart parsing example (CKY parsing)



The chart stores a *parse forest* efficiently.

CKY summary

+ CKY avoids re-computing the analyses by storing the earlier analyses (of sub-spans) in a table
– It still computes lower level constituents that are not allowed by the grammar
– CKY requires the grammar to be in CNF
- CKY has $O(n^3)$ recognition complexity
- For parsing we need to keep track of backlinks
- CKY can efficiently store all possible parses in a chart
- Enumerating all possible parses have exponential complexity (worst case)

Earley algorithm

- Earley algorithm is a top down (and left-to-right) parsing algorithm (Earley 1970)
- It allows arbitrary CFGs
- Keeps record of constituents that are
  - predicted using the grammar (top-down)
  - in-progress with partial evidence
  - completed based on input seen so far

  at every position in the input string
- Time complexity is $O(n^3)$

Earley chart entries (states or items)

Earley chart entries are CF rules with a 'dot' on the RHS representing the state of the rule

- $A \rightarrow \bullet\alpha[i,i]$ predicted without any evidence (yet)
- $A \rightarrow \alpha \bullet \beta[i,j]$ partially matched
- $A \rightarrow \alpha\beta \bullet [i,j]$ completed, the non-terminal A is found in the given span

Earley algorithm: an informal sketch

1. Start at position 0, predict S
2. Predict all possible states (rules that apply)
3. Read a word
4. Update the table, advance the dot if possible
5. Go to step 2
6. If we have a completed S production at the end of the input, the input it recognized

Earley algorithm: three operations

Predictor adds all rules that are possible at the given state

Completer adds states from the earlier chart entries that match the completed state to the chart entry being processed, and advances their dot

Scanner adds a completed state to the next chart entry if the current category is a POS tag, and the word matches

Earley parsing example (chart[0])

$_0$ she $_1$ saw $_2$ a $_3$ duck $_4$

state	rule	position	operation
0	$\gamma \rightarrow \bullet$S	[0,0]	initialization
1	S $\rightarrow \bullet$NP VP	[0,0]	predictor
2	S $\rightarrow \bullet$Aux NP VP	[0,0]	predictor
3	NP $\rightarrow \bullet$Det N	[0,0]	predictor
4	NP $\rightarrow \bullet$NP PP	[0,0]	predictor
5	NP $\rightarrow \bullet$Prn	[0,0]	predictor

Note: the chart[0] is independent of the input.

S → NP VP
S → Aux NP VP
NP → Det N
NP → Prn
NP → NP PP
VP → V NP
VP → V
VP → VP PP
PP → Prp NP
N → duck
N → park
V → duck
V → ducks
V → saw
Prn → she | her
Prp → in | with
Det → a | the
Aux → does | has

Earley parsing example (chart[1])

$_0$ she $_1$ saw $_2$ a $_3$ duck $_4$

state	rule	position	operation
6	Prn $\rightarrow$she $\bullet$	[0,1]	scanner
7	NP $\rightarrow$Prn $\bullet$	[0,1]	completer
8	S $\rightarrow$NP $\bullet$VP	[0,1]	completer
9	NP $\rightarrow$NP $\bullet$PP	[0,1]	completer
10	VP $\rightarrow \bullet$V NP	[1,1]	predictor
11	VP $\rightarrow \bullet$V	[1,1]	predictor
12	VP $\rightarrow \bullet$VP PP	[1,1]	predictor
13	PP $\rightarrow \bullet$Prp NP	[1,1]	predictor

Earley parsing example (chart[2])

$_0$ she $_1$ saw $_2$ a $_3$ duck $_4$

state	rule	position	operation
14	V $\rightarrow$saw $\bullet$	[1,2]	scanner
15	VP $\rightarrow$V $\bullet$NP	[1,2]	completer
16	VP $\rightarrow$V $\bullet$	[1,2]	completer
17	NP $\rightarrow \bullet$Det N	[2,2]	predictor
18	NP $\rightarrow \bullet$NP PP	[2,2]	predictor
19	NP $\rightarrow \bullet$Prn	[2,2]	predictor
20	S $\rightarrow$NP VP $\bullet$	[0,2]	predictor

Earley parsing example (chart[3])

$_0$ she $_1$ saw $_2$ a $_3$ duck $_4$

state	rule	position	operation
21	Det $\rightarrow$a $\bullet$	[2,3]	scanner
22	NP $\rightarrow$Det $\bullet$N	[2,3]	completer

Earley parsing example (chart[4])

$_0$ she $_1$ saw $_2$ a $_3$ duck $_4$

state	rule	position	operation
23	N $\rightarrow$duck $\bullet$	[3,4]	scanner
24	V $\rightarrow$duck $\bullet$	[3,4]	scanner
25	NP $\rightarrow$Det N $\bullet$	[2,4]	completer
26	VP $\rightarrow$V NP $\bullet$	[1,4]	completer
27	S $\rightarrow$NP VP $\bullet$	[0,4]	completer

Earley parsing: summary

- Top-down approach with bottom-up filtering (better filtering may be achived with lookahead)
- It can process any CFG (no need for CNF)
- Complexity is the same as CKY
  - time complexity : $O(n^3)$
  - space complexity: $O(n^2)$
- Our examples show recognition, we need to maintain backlinks for parsing
- Again, Earley chart stores a parse forest compactly, but extracting all trees may require exponential time

An exercise

Construct the CKY and Earley charts for the following sentence

`The duck she saw is in the park`

Recommended grammar:

S	→	NP VP	PP	→	Prp NP
NP	→	Det N	N	→	park
NP	→	Prn	N	→	duck
NP	→	NP PP	V	→	duck
NP	→	NP S	V	→	saw
VP	→	V NP	Prn	→	she
VP	→	V	Prp	→	in
VP	→	VP PP	Det	→	the

Summary: context-free parsing algorithms

- Naive search for parsing is intractable
- Dynamic programming algorithms allow polynomial time recognition
- Parsing may still be exponential in the worse case
- Charts represent ambiguity, but cannot say anything about which parse is the best

Pretty little girl's school

Natural languages and ambiguity

Cartoon Theories of Linguistics, SpecGram Vol CLIII, No 4, 2008. `http://specgram.com/CLIII.4/school.gif`

Some more examples

- Lexical ambiguity
  - *She is looking for a match*
  - *We saw her duck*
- Attachment ambiguity
  - *I saw the man with a telescope*
  - *Panda eats bamboo shoots and leaves*
- Local ambiguity (garden path sentences)
  - *The horse raced past the barn fell*
  - *The old man the boats*
  - *Fat people eat accumulates*

We use statistical methods for dealing with ambiguity (not in this course).

References / additional reading material

- Jurafsky and Martin (2009, Chapter 13)

References / additional reading material (cont.)

Cocke, John and J. T. Schwartz (1970). *Programming languages and their compilers: preliminary notes*. Tech. rep. Courant Institute of Mathematical Sciences, NYU.

Earley, Jay (Feb. 1970). "An Efficient Context-free Parsing Algorithm". In: *Commun. ACM* 13.2, pp. 94–102. ISSN: 0001-0782. DOI: `10.1145/362007.362035`. URL: `http://doi.acm.org/10.1145/362007.362035`.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.

Kasami, Tadao (1965). *An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages*. Tech. rep. DTIC Document.

Younger, Daniel H (1967). "Recognition and parsing of context-free languages in time n 3". In: *Information and control* 10.2, pp. 189–208.