Natural Language Processing

Assist. Prof. Dr. Tuğba YILDIZ

İSTANBUL BİLGİ UNIVERSITY
Department of Computer Engineering

April 14, 2017

Context-Free Grammar

- the symbols that are used in a CFG are divided into two classes:
  1. terminal symbols:
     - the symbols that correspond to words in the language ("the", "club") are called terminal symbols
     - the lexicon is the set of rules that introduce these terminal symbols
  2. nonterminal symbols:
     - the symbols that express clusters or generalizations of these are called nonterminals
- in each context-free rule, the item to the right of the arrow is an ordered list of one or more terminals and nonterminals
- while to the left of the arrow is a single nonterminal symbol expressing some cluster or generalization

Context-Free Grammar

- a CFG is usually thought of in two ways:
  1. as a device for **generating sentences**
  2. as a device for **assigning a structure** to a given sentence.
- as a generator, we could read the arrow as "rewrite the symbol on the left with the string of symbols on the right"
- rewrite NP as Det Nominal
- Det Nominal
- rewrite Nominal as Noun
- Det Noun
- a flight

Context-Free Grammar

- we say the string **a flight** can be derived from the nonterminal NP
- Thus a CFG can be used to randomly generate a series of strings
- This sequence of rule expansions is called a derivation of the string of words
- It is common to represent a derivation by a parse tree

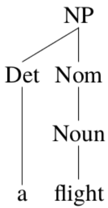


Fig.1 Parse tree of a flight

Context-Free Grammar

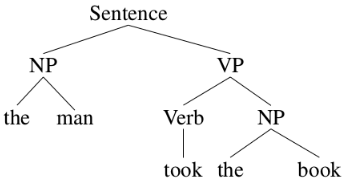


Fig 2. The first context-free grammar parse tree (Chomsky, 1956)

Context-Free Grammar

- the formal language defined by a CFG is the set of strings that are derivable from the designated start symbol.
- each grammar must have one designated start symbol, which is often called S.
- since context-free grammars are often used to define sentences, S is usually interpreted as the "sentence" node

Context-Free Grammar

$$
\begin{aligned}
Noun &\rightarrow flights \mid breeze \mid trip \mid morning \mid \ldots \\
Verb &\rightarrow is \mid prefer \mid like \mid need \mid want \mid fly \\
Adjective &\rightarrow cheapest \mid non-stop \mid first \mid latest \\
&\quad \mid other \mid direct \mid \ldots \\
Pronoun &\rightarrow me \mid I \mid you \mid it \mid \ldots \\
Proper\text{-}Noun &\rightarrow Alaska \mid Baltimore \mid Los\ Angeles \\
&\quad \mid Chicago \mid United \mid American \mid \ldots \\
Determiner &\rightarrow the \mid a \mid an \mid this \mid these \mid that \mid \ldots \\
Preposition &\rightarrow from \mid to \mid on \mid near \mid \ldots \\
Conjunction &\rightarrow and \mid or \mid but \mid \ldots
\end{aligned}
$$

Fig.3 The lexicon for L0 .

Context-Free Grammar

S	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	*Pronoun*	I
	$\mid$	*Proper-Noun*	Los Angeles
	$\mid$	*Det Nominal*	a + flight
Nominal	$\rightarrow$	*Noun Nominal*	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

Fig. 4 The grammar for L0 with example phrases for each rule.

Context-Free Grammar

- We can use this grammar to generate sentences
  "I prefer a morning flight"
- We start with S, expand it to NP VP,
- then choose a random expansion of NP
- (let's say to I), and a random expansion of VP (let's say to Verb NP),
- and so on until we generate the string I prefer a morning flight.

Context-Free Grammar

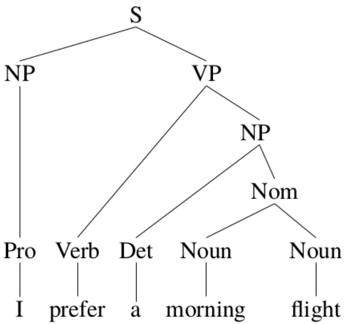


Fig. 5 The parse tree for ‘I prefer a morning flight’ according to grammar L0

Context-Free Grammar

- it is sometimes convenient to represent a parse tree in a more compact format called bracketed notation

$$[_S [_{NP} [_{Pro}\ \mathrm{I}]] [_{VP} [_V\ \mathrm{prefer}] [_{NP} [_{Det}\ \mathrm{a}] [_{Nom} [_N\ \mathrm{morning}] [_N\ \mathrm{flight}]]]]]$$

Fig. 6 The parse tree for ‘I prefer a morning flight’ according to grammar L0

Context-Free Grammar

- a CFG like that of L0 defines a formal language
- a formal language is a set of strings (baa!, baaa! ...)
- sentences (strings of words) that can be derived by a grammar are in the formal language defined by that grammar and are called grammatical sentences.
- sentences that cannot be derived by a given formal grammar are not in the language defined by that grammar and are referred to as ungrammatical

Context-Free Grammar

- We conclude this section by way of summary with a quick formal description of a CFG and the language it generates.
- A CFG has four parameters (technically 'is a 4-tuple'):
  1. a set of non-terminal symbols (or "variables") N
  2. a set of terminal symbols $\sum$ (disjoint from N)
  3. a set of productions P, each of the form A $\rightarrow$ α, where A is a non-terminal and α is a string of symbols from the infinite set of strings $(\sum \cup N)*$.
  4. a designated start symbol S
- A language is defined via the concept of derivation.
- One string derives another one if it can be rewritten as the second one via some series of rule applications.

Parsing

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$

Fig. 8 A miniature English grammar and lexicon

Problems with basic parsing

- three problems are:
- left-recursion
- ambiguity
- inefficient reparsing of subtrees

Dynamic Programming

- We want a parsing algorithm (using dynamic programming technique) that fills a table with solutions to subproblems that:
  - Does not do repeated work
  - Does top-down search with bottom-up filtering
  - Solves the left-recursion problem
  - Solves an exponential problem in O(N^3) time.
- The answer is Earley Algorithm.
- The answer is CYK Algorithms

CYK Algorithm

- The membership problem:
- Problem:
- Given a context-free grammar G and a string w
- $G = (V, \sum, P, S)$ where
- V finite set of variables
- $\sum$ (the alphabet) finite set of terminal symbols
- P finite set of rules
- S start symbol (distinguished element of V)
- V and $\sum$ are assumed to be disjoint
- G is used to generate the string of a language
- Question:
  Is w in L(G)?

CYK Algorithm

- J. Cocke
- D. Younger
- T. Kasami
- Independently developed an algorithm to answer this question.
- The Structure of the rules in a Chomsky Normal Form grammar
- Uses a "dynamic programming" or "table-filling algorithm"

CYK Algorithm

- Chomsky Normal Form
- Normal Form is described by a set of conditions that each rule in the grammar must satisfy
- Context-free grammar is in CNF if each rule has one of the following forms:
- A $\rightarrow$ BC
- A $\rightarrow$ a, or
- S $\rightarrow$ λ
- where B,C $\in$ V - {S}

CYK Algorithm

- Chomsky Normal Form (CNF)
- CNF allows only two kinds of right-hand sides:
- Two nonterminals:
- VP $\rightarrow$ ADV VP NP
- VP $\rightarrow$ eat
- Any CFG can be transformed into an equivalent CNF:
- VP $\rightarrow$ ADV VP1
- VP1 $\rightarrow$ VP2 NP
- VP2 $\rightarrow$ eat

CYK Algorithm

- A note about ϵ-productions
- Formally, context-free grammars are allowed to have empty productions (ϵ = the empty string):
  VP $\rightarrow$ V NP
  NP $\rightarrow$ DT Noun
  NP $\rightarrow$ ϵ
- These can always be eliminated without changing the language generated by the grammar and becomes:
  VP $\rightarrow$ V NP
  NP $\rightarrow$ DT Noun
  VP $\rightarrow$ V ϵ
- We will assume that our grammars don't have ϵ-productions
  VP $\rightarrow$ V NP
  NP $\rightarrow$ DT Noun
  VP $\rightarrow$ V

CKY chart parsing algorithm

- Bottom-up parsing:
  start with the words
- Dynamic programming:
  save the results in a table/chart
  re-use these results in finding larger constituents
- Complexity: $O(n^3|G|)$
  n: length of string, $|G|$: size of grammar
- Presumes a CFG in Chomsky Normal Form:
- Rules are all either
  A $\rightarrow$ BC
  A $\rightarrow$ a (with A,B,C nonterminals and a terminal)

CYK Algorithm

- CYK algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do** — Looping over the columns

$\quad table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in gram$ — Filling the bottom cell

$\quad$ **for** $i \leftarrow$ **from** $j-2$ **downto** 0 **do** — Filling row i in column j

$\quad\quad$ **for** $k \leftarrow i+1$ **to** $j-1$ **do** — Looping over the possible split locations between i and j.

$\quad\quad\quad table[i,j] \leftarrow table[i,j] \cup$

$\quad\quad\quad\quad \{A \mid A \rightarrow BC \in grammar,$

$\quad\quad\quad\quad\quad B \in table[i,k],$

$\quad\quad\quad\quad\quad C \in table[k,j]\}$

Check the grammar for rules that link the constituents in [i,k] with those in [k,j]. For each rule found store the LHS of the rule in cell [i,j].

CYK Algorithm

- Construct a Triangular Table
- $X_{i,i}$ is the set of variables A such that $A \to w_i$ is a production of G
- Compare at most n pairs of previously computed sets:
- $(X_{i,i}, X_{i+1,j})$, $(X_{i,i+1}, X_{i+2,j})$ ... $(X_{i,j-1}, X_{j,j})$

CYK Algorithm

- Construct a Triangular Table

$X_{1,5}$				
$X_{1,4}$	$X_{2,5}$			
$X_{1,3}$	$X_{2,4}$	$X_{3,5}$		
$X_{1,2}$	$X_{2,3}$	$X_{3,4}$	$X_{4,5}$	
$X_{1,1}$	$X_{2,2}$	$X_{3,3}$	$X_{4,4}$	$X_{5,5}$
w_1	w_2	w_3	w_4	w_5

CYK Algorithm- Example1

- Show the CYK Algorithm with the following example:
- CNF grammar G
  - $S \rightarrow AB \mid BC$
  - $A \rightarrow BA \mid a$
  - $B \rightarrow CC \mid b$
  - $C \rightarrow AB \mid a$
- w is baaba
- Question is baaba in L(G)?

CYK Algorithm

- Construct a Triangular Table

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- $X1,2 = (Xi,i, Xi{+}1,j) = (X1,1\ ,\ X2,2)$
- $\{B\}\{A,C\} = \{BA,BC\}$
- Steps:
  - Look for production rules to generate BA or BC
  - There are two: S and A
  - $X1,2 = \{S, A\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK Algorithm

- Construct a Triangular Table

{S, A}				
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- X2,3 = (Xi,i, Xi+1,j) = (X2,2 , X3,3)
- {A,C}{A,C} = {AA,AC,CA,CC} = Y
- Steps:
  - Look for production rules to generate Y
  - There are two: B
  - X2,3 = {B}

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK Algorithm

- Construct a Triangular Table

{S, A}	{B}			
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- $X3,4 = (Xi,i,\ Xi+1,j) = (X3,3\ ,\ X4,4)$
- $\{A,C\}\{B\} = \{AB,CB\} = Y$
- Steps:
  - Look for production rules to generate Y
  - There are two: S and C
  - $X3,4 = \{S,C\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK Algorithm

- Construct a Triangular Table

{S, A}	{B}	{S, C}		
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- $X4,5 = (Xi,i, Xi+1,j) = (X4,4 , X5,5)$
- $\{B\}\{A,C\} = \{BA,BC\} = Y$
- Steps:
  - Look for production rules to generate Y
  - There are two: S and A
  - $X4,5 = \{S,A\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK Algorithm

- Construct a Triangular Table

{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- X1,3 = (Xi,i, Xi+1,j)(Xi,i+1, Xi+2,j)
- = (X1,1 , X2,3),(X1,2 , X3,3)
- $\{B\}\{B\} \cup \{S,A\}\{A,C\} = \{BB,SA,SC,AA,AC\} = Y$
- Steps:
    - Look for production rules to generate Y
    - There are NONE
    - X1,2 = 0
    - no elements in this set (empty set)

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$

CYK Algorithm

- Construct a Triangular Table

∅				
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- $X2,4 = (Xi,i,\ Xi+1,j)(Xi,i+1,\ Xi+2,j)$
- $= (X2,2\ ,\ X3,4),(X2,3\ ,\ X4,4)$
- $\{A,C\}\{S,C\} \cup \{B\}\{B\} = \{AS,AC,CS,CC,BB\} = Y$
- Steps:
  - Look for production rules to generate Y
  - There is one : B
  - $X2,4 = \{B\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK Algorithm

- Construct a Triangular Table

Ø	{B}			
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	**a**	**a**	**b**	**a**

CYK Algorithm

- $X3,5 = (Xi,i,\ Xi{+}1,j)(Xi,i{+}1,\ Xi{+}2,j)$
- $= (X3,3\ ,\ X4,5),(X3,4\ ,\ X5,5)$
- $\{A,C\}\{S,A\} \cup \{S,C\}\{A,C\} = \{AS,AA,CS,CA,SA,\ SC,\ CA,\ CC\} = Y$
- Steps:
  - Look for production rules to generate Y
  - There is one : B
  - $X3,5 = \{B\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK Algorithm

- Construct a Triangular Table

Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- Construct a Triangular Table

{S, A, C}	← $X_{1,5}$			
Ø	{S, A, C}			
Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

CYK Algorithm

- Is baaba in L(G)?
- Yes
- We can see the S in the set X1n where "n" = 5
- We can see the table the cell $X15 = (S, A, C)$ then
- if $S \in X15$ then baaba $\in L(G)$

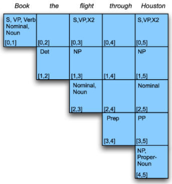
CYK Algorithm

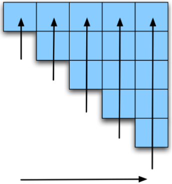
- The CYK Algorithm correctly computes Xij for all i and j; thus w is in L(G) if and only if S is in X1n

CYK Algorithm

- CYK-Example

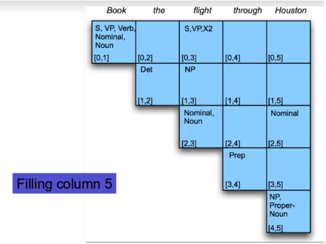
Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	





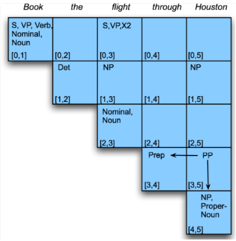
CYK Algorithm

- CYK-Example



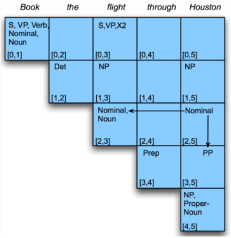
CYK Algorithm

- CYK-Example



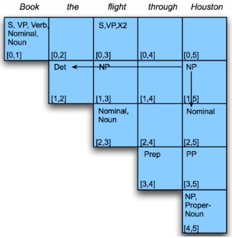
CYK Algorithm

- CYK-Example



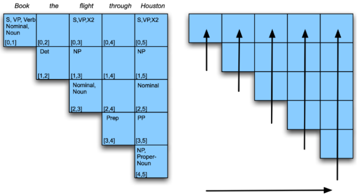
CYK Algorithm

- CYK-Example



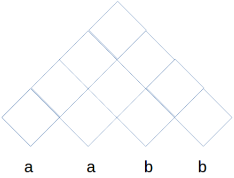
CYK Algorithm

- CYK-Example



CYK Algorithm

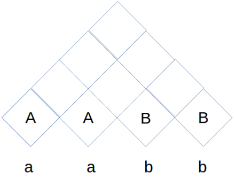
- CYK-Example



$S \rightarrow \epsilon \mid AB \mid XB$
$T \rightarrow AB \mid XB$
$X \rightarrow AT$
$A \rightarrow a$
$B \rightarrow b$

CYK Algorithm

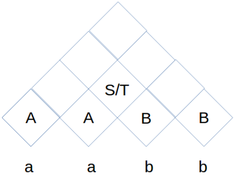
- CYK-Example



S → ϵ | AB | XB
T → AB | XB
X → AT
A → a
B → b

CYK Algorithm

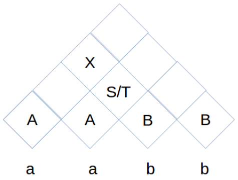
- CYK-Example



$S \rightarrow \epsilon \mid AB \mid XB$
$T \rightarrow AB \mid XB$
$X \rightarrow AT$
$A \rightarrow a$
$B \rightarrow b$

CYK Algorithm

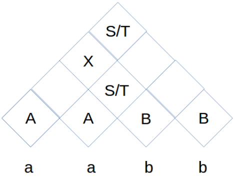
- CYK-Example



$S \rightarrow \epsilon \mid AB \mid XB$
$T \rightarrow AB \mid XB$
$X \rightarrow AT$
$A \rightarrow a$
$B \rightarrow b$

CYK Algorithm

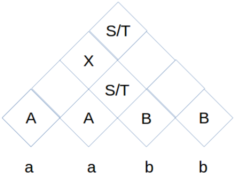
- CYK-Example



S → ϵ | AB | XB
T → AB | XB
X → AT
A → a
B → b

CYK Algorithm

- CYK-Example
- S in top square? Yes, a a b b belongs to the language :-)



S → ϵ | AB | XB
T → AB | XB
X → AT
A → a
B → b

References

- Speech and Language Processing (3rd ed. draft) by D. Jurafsky & J. H. Martin (web.stanford.edu)
- David Rodriguez - Velazquez CS – 6800 Summer I - 2009
- Julia Hockenmaier, Lecture 12:The CKY parsing algorithm
- Miguel Ballesteros, CKY Algorithm, Algorithms for NLP Course.