Artificial Intelligence

Subject Code: 20A05502T

UNIT IV - Natural Language for Communication

SYNTACTIC ANALYSIS - PARSING

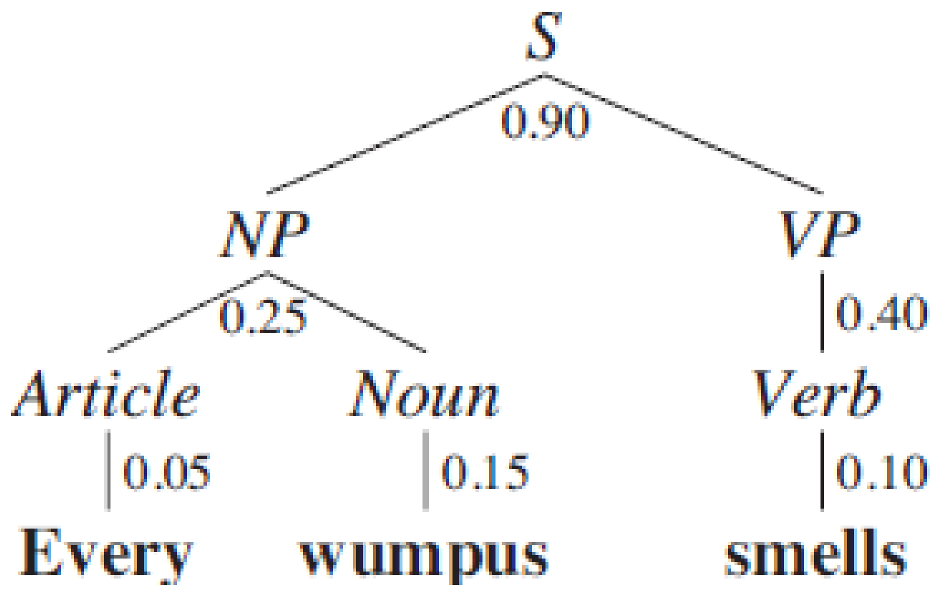
SYNTACTIC ANALYSIS (PARSING)

- **Parsing** is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar.
- Top Down Parsing
- The S, starting symbol and search top down for a tree that has the words as its leaves, or
- Bottom up Parsing
- Start with the words and search bottom up for a tree that culminates in an S.
- Both top-down and bottom-up parsing can be inefficient, because they can end up repeating effort in areas of the search space that lead to dead ends.

List of items	*Rule*
S	
$NP\ VP$	$S \rightarrow NP\ VP$
$NP\ VP\ Adjective$	$VP \rightarrow VP\ Adjective$
$NP\ Verb\ Adjective$	$VP \rightarrow Verb$
$NP\ Verb$ **dead**	$Adjective \rightarrow$ **dead**
NP **is dead**	$Verb \rightarrow$ **is**
$Article\ Noun$ **is dead**	$NP \rightarrow Article\ Noun$
$Article$ **wumpus is dead**	$Noun \rightarrow$ **wumpus**
the wumpus is dead	$Article \rightarrow$ **the**

Parse Tree

- Parse tree for the sentence "Every wumpus smells", according to the grammar E_0.
- Each interior node of the tree is labeled with its probability.
- The probability of the tree as a whole is $0.9 \times 0.25 \times 0.05 \times 0.15 \times 0.40 \times 0.10 = 0.0000675$.
- Since this tree is the only parse of the sentence, that number is also the probability of the sentence.
- The tree can also be written in linear form as
- [S [NP [Article **every**] [Noun **wumpus**]][VP [Verb **smells**]]].



- The E0 grammar generates a wide range of English sentences such as the following:
- John is in the pit
- The wumpus that stinks is in 2 2
- Mary is in Boston and the wumpus is near 3 2

Drawbacks

- Consider the following two sentences:
  - Have the students in section A of Computer Science III take the exam.
  - Have the students in section A of Computer Science III taken the exam?
- Even though they share the first 10 words, these sentences have very different parses, because the first is a command and the second is a question.
- A left-to-right parsing algorithm would have to guess whether
- the **first word** is part of a **command or a question** and
- will not be able to tell if the guess is correct until at least the eleventh word, *take* or *taken.*
- If the algorithm guesses wrong, it will have to backtrack all the way to the first word and reanalyze the whole sentence under the other interpretation.
- To avoid this source of inefficiency we can use dynamic programming.

Dynamic Programming:

- *In dynamic programming, "every time we analyze a substring, store the results so we won't have to reanalyze it later".*
- For example,
- "the students in section B of Computer Science III" is an NP,
- record that result in a data structure as a **chart**.
- Algorithms that do this are called **chart parsers**.
- In context-free grammars, any phrase that was found in the context of one branch of the search space, can work as well in any other branch of the search space.
- There are many types of chart parsers;
- a bottom-up version called the **CYK algorithm**, after its inventors, John Cocke, Daniel Younger, and Tadeo Kasami

CYK algorithm

- Given a sequence of words,
- it finds the most probable derivation for the whole sequence and for each subsequence.
- It returns the whole table, P, in which an entry P[X, start , len] is the probability of the most probable X of length len starting at position start .
- If there is no X of that size at that location, the probability is 0.

function CYK-PARSE(*words*, *grammar*) **returns** P, a table of probabilities

$N \leftarrow$ LENGTH(*words*)
$M \leftarrow$ the number of nonterminal symbols in *grammar*
$P \leftarrow$ an array of size $[M, N, N]$, initially all 0
/* *Insert lexical rules for each word* */
for $i = 1$ **to** N **do**
  for each rule of form ($X \rightarrow words_i\ [p]$) **do**
    $P[X, i, 1] \leftarrow$ p
/* *Combine first and second parts of right-hand sides of rules, from short to long* */
for *length* = 2 **to** N **do**
  for *start* = 1 **to** N – *length* + *1* **do**
    for *len1* = 1 **to** N – 1 **do**
      len2 ← *length* – *len1*
      for each rule of the form ($X \rightarrow Y\ Z\ [p]$) **do**
        $P[X, start, length] \leftarrow$ MAX($P[X, start, length]$,
          $P[Y, start, len1] \times P[Z, start + len1, len2] \times p$)
return P

CYK algorithm

- it requires a grammar with all rules in one of two very specific formats:
  - Lexical rules of the form X → **word**, and
  - Syntactic rules of the form X → Y Z |word
- This grammar format, called **Chomsky Normal Form**,
- any context-free grammar can be automatically transformed into Chomsky Normal Form.
- The CYK algorithm uses
- space of $O(n^2m)$ for the P table,
- time $O(n^3m)$.
- where n is the number of words in the sentence, and
- m is the number of nonterminal symbols in the grammar.

Thank You