SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 98
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

Course Name – 19AD601 – Natural Language Processing

III Year / VI Semester

Unit 3 – SYNTACTIC ANALYSIS

Topic 6- Dynamic Programming parsing

Dynamic Programming parsing

Dynamic programming provides a powerful framework for addressing the problems caused by ambiguity in grammars.

Dynamic programming does the following,

1. Describe solution in terms of solution to any sub-problems
2. Determine all the sub-problems you'll need to apply this recursively
3. Solve every sub-problem (once only) in an appropriate order.

The complete table has the solution to all the subproblems needed to solve the problem as a whole.

In the case of syntactic parsing, these subproblems represent parse trees for all the constituents detected in the input. Dynamic programming parsing also called as table parsing or chart parsing.

Dynamic Programming parsing

CKY

- Cocke-Kasami- Younger (CKY) algorithm, the most widely used dynamic-programming based approach to parsing. It is a bottom-up approach start with words.

- A reason for its simplicity is that it only works for grammars in CNF.

- To store intermediate parses, it uses a data structure called chart. It is a set of nodes linked by edges.

- Each word of the input sentence is surrounded by two nodes that are marked by numbers. The edges at their turn mark the constituents.

CKY Algorithm

The CKY algorithm requires grammars to first be in Chomsky Normal Form (CNF). CNF are restricted to rules of the form A □ B C or A □ w.

That is, the right-hand side of each rule must expand either to two non-terminals or to a single terminal.

Dynamic Programming parsing

$\mathcal{L}_1$ Grammar	$\mathcal{L}_1$ in CNF
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

Figure 17.10 $\mathcal{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here, all the original lexical entries from $\mathcal{L}_1$ carry over unchanged as well.

Dynamic Programming parsing

A two-dimensional matrix can be used to encode the structure of an entire tree. For a sentence of length n, we will work with the upper-triangular portion of an (n+1)x(n+1) matrix.

Each cell [I,j] in this matrix contains the set of non-terminals that represent all the constituents that aspan positions i through j of the input.

To make this more concrete, consider the following example with its completed parse matrix,

Book the flight through Houston

The superdiagonal row in the matrix contains the parts of speech for each word in the input.

The subsequent diagonals above that superdiagonal contain constituents that cover all the spans of increasing length in the input.

Dynamic Programming parsing

The outermost loop of the algorithm iterates over the columns, and the second loop iterates over the rows, from the bottom up.

The purpose of the innermost loop is to range over all the places where a substring spanning i to j in the input might be split in two.

As k ranges over the places where the string can be split, the pairs of cells we consider move, in lockstep, to the right along row i and down along column j.

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

  for j ← from 1 to LENGTH(words) do
      for all {A | A → words[j] ∈ grammar}
              table[j − 1, j] ← table[j − 1, j] ∪ A
      for i ← from j − 2 down to 0 do
          for k ← i + 1 to j − 1 do
              for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
                      table[i,j] ← table[i,j] ∪ A
```

Figure 17.12 The CKY algorithm.

Dynamic Programming parsing

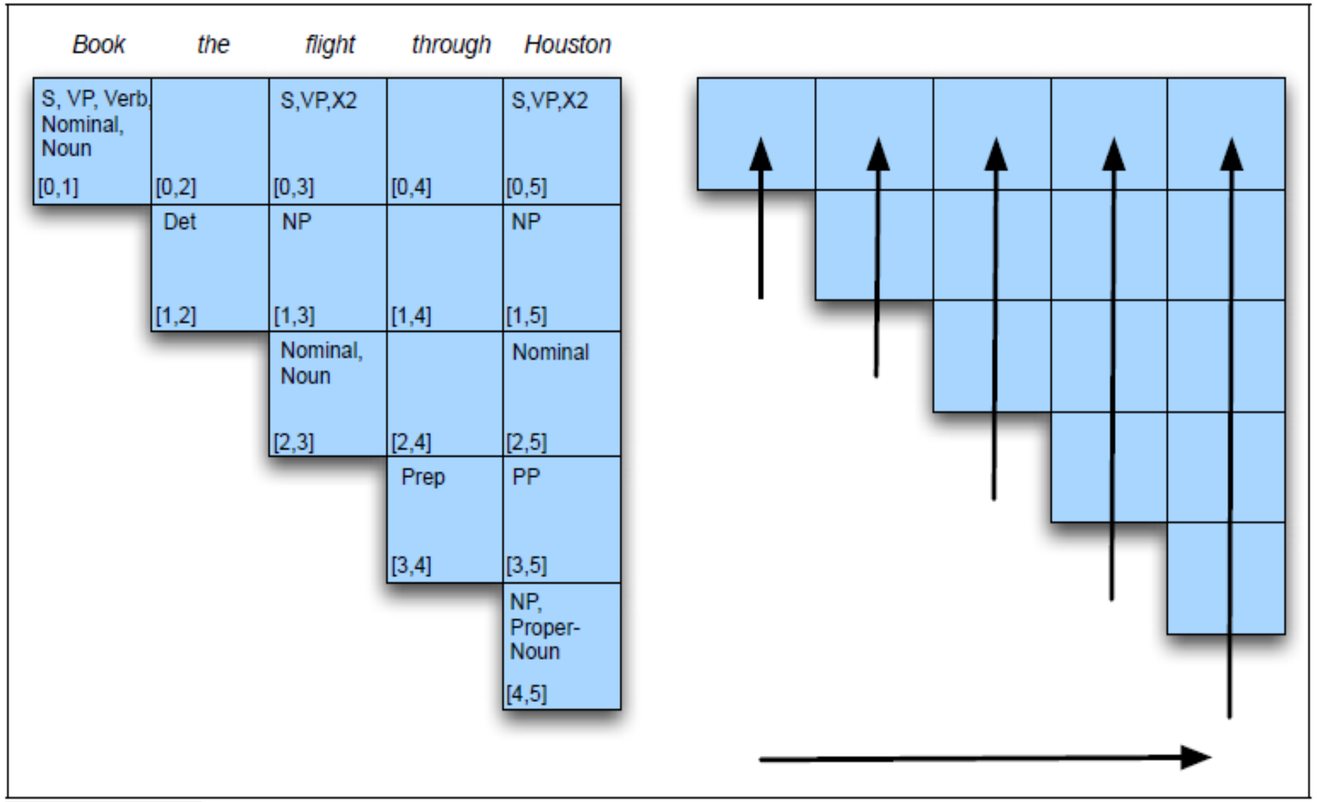

Figure 17.11 Completed parse table for *Book the flight through Houston.*

THANK YOU