Natural Language Processing
DSECL ZG565

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

Session 9-Grammars and Parsing
Date – 10th Feb 2024

These slides are prepared by the instructor, with grateful acknowledgement of Prof. James Allen and many others who made their course materials freely available online.

Session Content
(Ref: Allen James - Chapter 3)

innovate achieve lead

- Grammars and Sentence Structure
- What Makes a Good Grammar
- Parsing
- A Top-Down Parser
- A Bottom-Up Chart Parser
- Top-Down Chart Parsing
- Finite State Models and Morphological Processing.

Ambiguity is Explosive

- "I saw the man with the telescope": 2 parses
- "I saw the man on the hill with the telescope.": 5 parses
- "I saw the man on the hill in Texas with the telescope": 14 parses
- "I saw the man on the hill in Texas with the telescope at noon.": 42 parses
- "I saw the man on the hill in Texas with the telescope at noon on Monday" 132 parses

Some funny examples

innovate achieve lead

– Policeman to little boy: “We are looking for a thief with a bicycle.” Little boy: “Wouldn’t you be better using your eyes.”
– Why is the teacher wearing sun-glasses. Because the class is so bright.

Grammars and Sentence Structure

- **Grammar**- formal specification of the structures allowable in the language, and
- **Parsing technique-** method of analyzing a sentence to determine its structure according to the grammar

What Makes a Good Grammar

- In small grammars, such as those that describe only a few types of sentences, one structural analysis of a sentence may appear as understandable as another, and little can be said as to why one is superior to the other.
- The analysis that retains its simplicity and generality as it is extended is more desirable
- Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammars (CFGs).

Context Free Grammar

N a set of non-terminal symbols (or variables)

Σ a set of terminal symbols (disjoint from N)

R a set of productions or rules of the form $A \rightarrow \beta$, where A is a non-terminal and β is a string of symbols from $(\Sigma \cup N)^*$

S, a designated non-terminal called the start symbol

Categories of Phrases

innovate achieve lead

Noun phrase (NP): Noun acts as the head word. They start with an article or noun.

Verb phrase (VP): Verb acts as the head word. They start with an verb

Adjective phrase (ADJP): Adjective as the head word. They start with an adjective

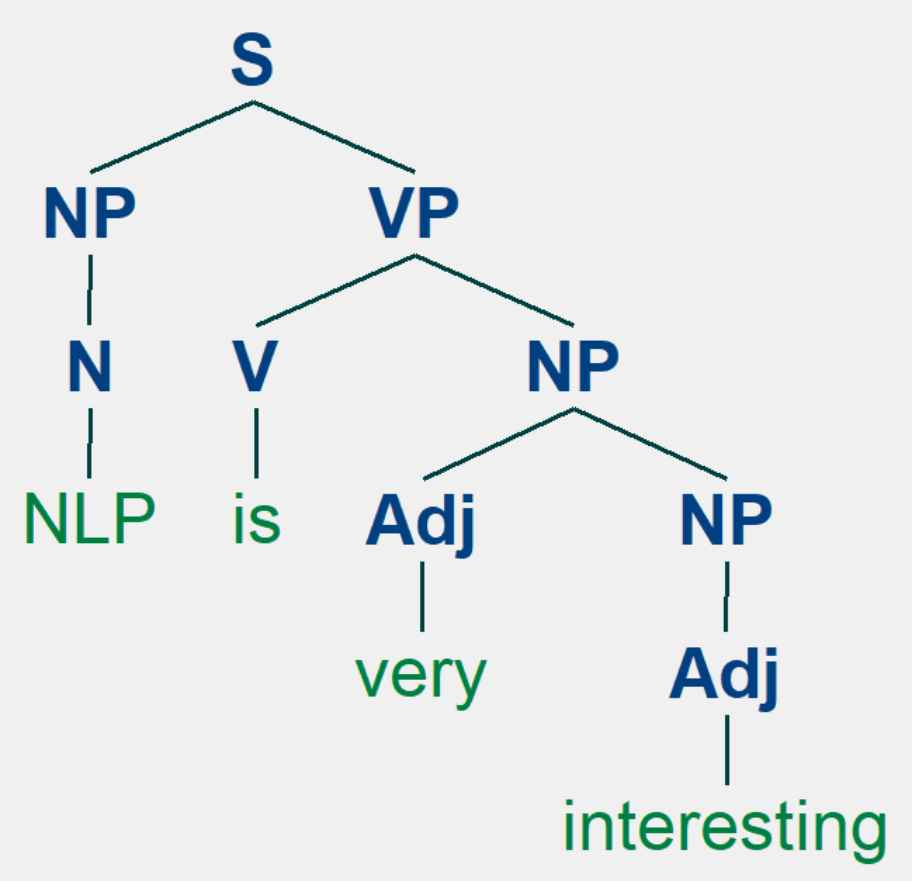
Adverb phrase (ADVP): Adverb acts as the head word. They usually start with an adverb

Prepositional phrase (PP): Preposition as the head word. They start with an preposition.

Simple grammar and Parsing Example

S -> NP VP
VP -> V NP
NP -> N
NP -> Adj NP
NP -> Adj

N -> NLP
V -> is
Adj -> very
Adj -> interesting

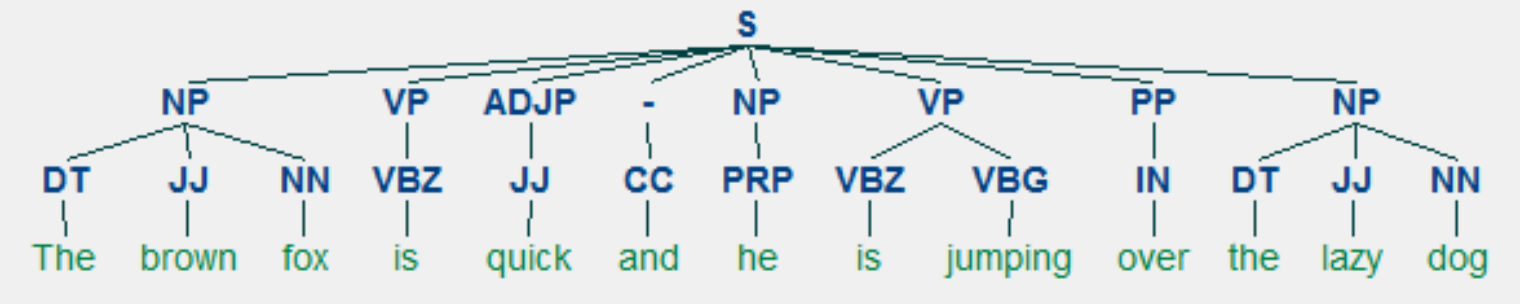


Parsing

Analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words.

Analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases.

This includes POS tags as well as phrases from a sentence.



Applications of Parsing

innovate achieve lead

Sentiment Analysis: Compare "I like Frozen", "I do not like Frozen", and "I like frozen yogurt". The three sentences' words are very similar to each other, yet the first and the second contain inverse statements about the movie "Frozen", while the third is a statement about something else. Parsing here is crucial for understanding.

Relation Extraction: "Rome is the capital of Italy and the region of Lazio". While entity extraction can give us the entities here, we need parsing to see which entity is the capital of which other entities.

Question Answering: When answering "Who was the first man in space?" you need to parse the question and use parsed sentences to build the answer.

Applications of Parsing

Speech Recognition: Parsing scores the strings with either a pass/fail or a likelihood score, to give a powerful language model for speech recognition. Similarly, parsing could help in **spell checking, optical character recognition (OCR), text prediction, handwriting detection** etc.

Machine Translation: Parsing allows us to choose between several possible translations. Also, it makes translation of phrases and terms easier.

Grammar Checking: Parsing can help in checking the grammaticality of a document.

Parsing

Given a string of non-terminals and a CFG, determine if the string can be generated by the CFG.

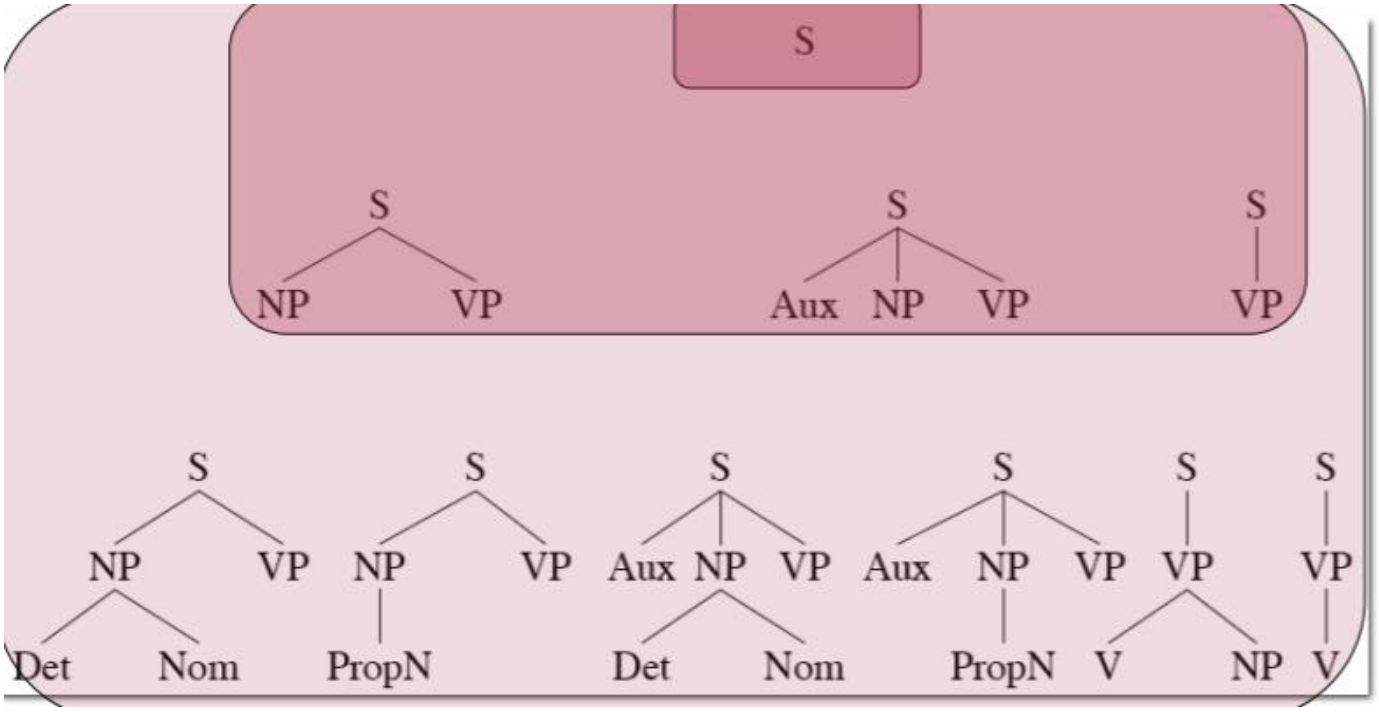
- Also return a parse tree for the string
- Also return all possible parse trees for the string

Must search space of derivations for one that derives the given string.

- Top-Down Parsing: Start searching space of derivations for the start symbol.
- Bottom-up Parsing: Start search space of reverse deivations from the terminal symbols in the string.

Top down Parsing

- **Parse tree is generated in the top to bottom fashion from root to leaves**
- **Search space from the start symbol sentence S on LHS**



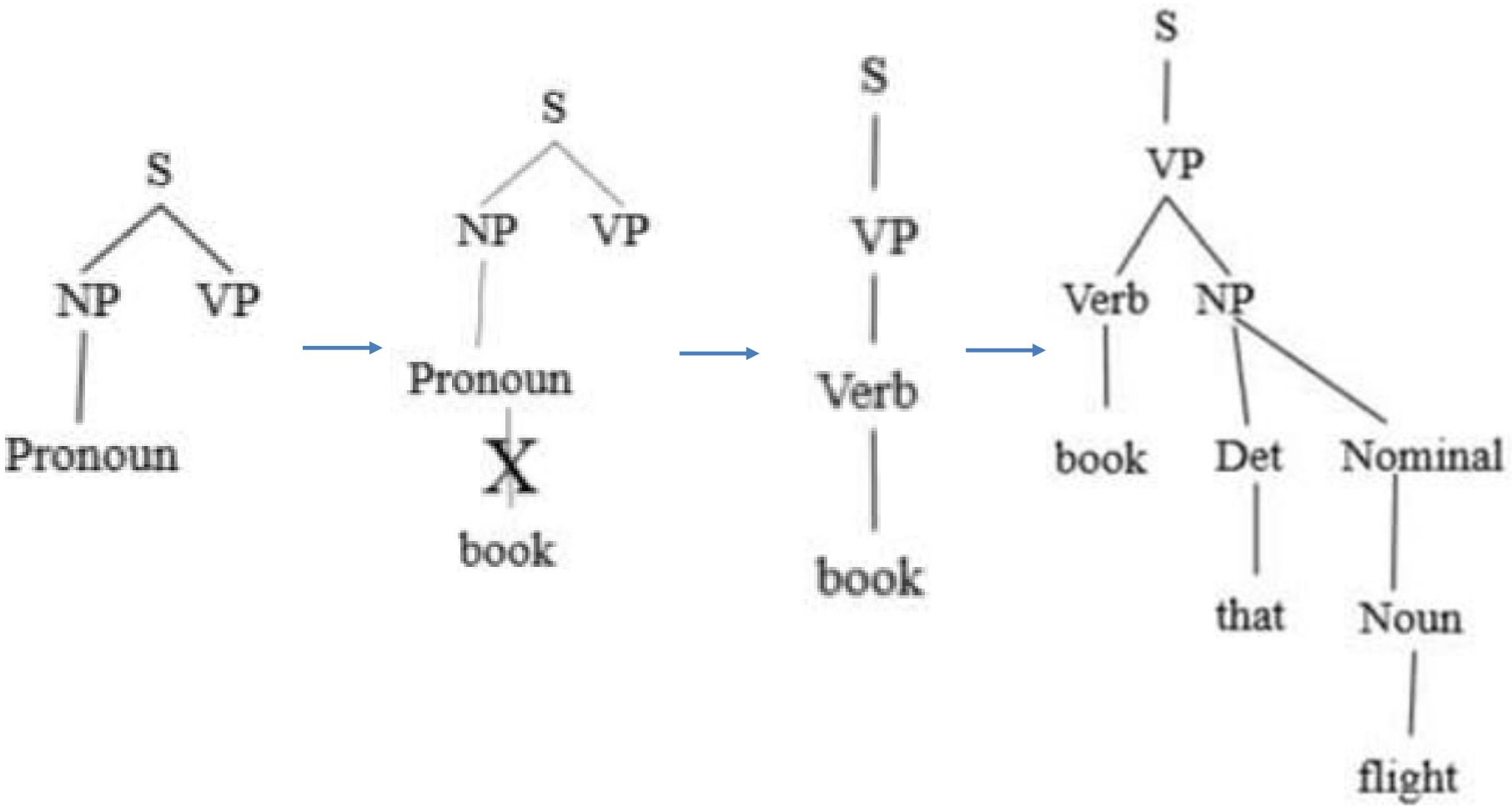
Top down parsing

1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N

4. VP -> V

5. VP -> V NP

Top down parsing



Parsing as a search procedure

- Parsing as a special case of a search problem as defined in AI.

1. Select the first state from the possibilities list (and remove it from the list).
2. Generate the new states by trying every possible option from the selected state (there may be none if we are on a bad path).
3. Add the states generated in step 2 to the possibilities list.

Top down parse of : "$_1$The $_2$ old $_3$ man $_4$ smiled. $_5$"

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(( ) 5)		success!

1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N
4. VP -> V
5. VP -> V NP

Lexicon

The → ART
Old → N, ADJ
Man → N, V
Smiled -> V

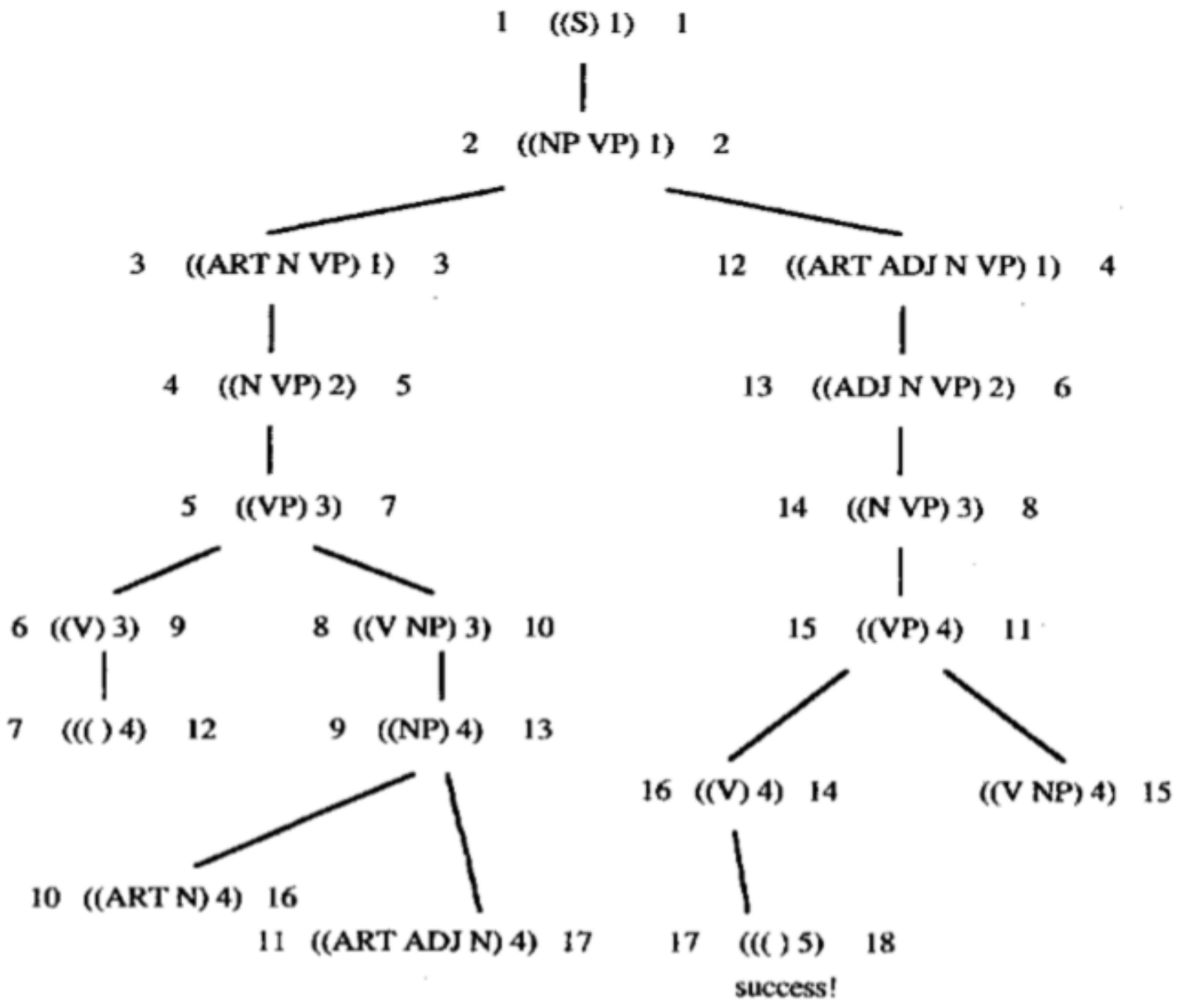


Figure 3.7 *Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)*

Depth-first strategy vs Breadth-first strategy

- For a depth-first strategy, the possibilities list is a stack, yielding a last-in first-out (LIFO) strategy.
- In contrast, in a breadth-first strategy the possibilities list is manipulated as a queue, yielding a first-in first-out (FIFO)) strategy
- With the depth-first strategy, one interpretation is considered and expanded until it fails; only then is the second one considered.
- With the breadth-first strategy, both interpretations are considered alternately, each being expanded one step at a time
- Many parsers built today use the depth-first strategy because it tends to minimize the number of backup states needed and thus uses less memory and requires less bookkeeping

Bottom up parsing

The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules.

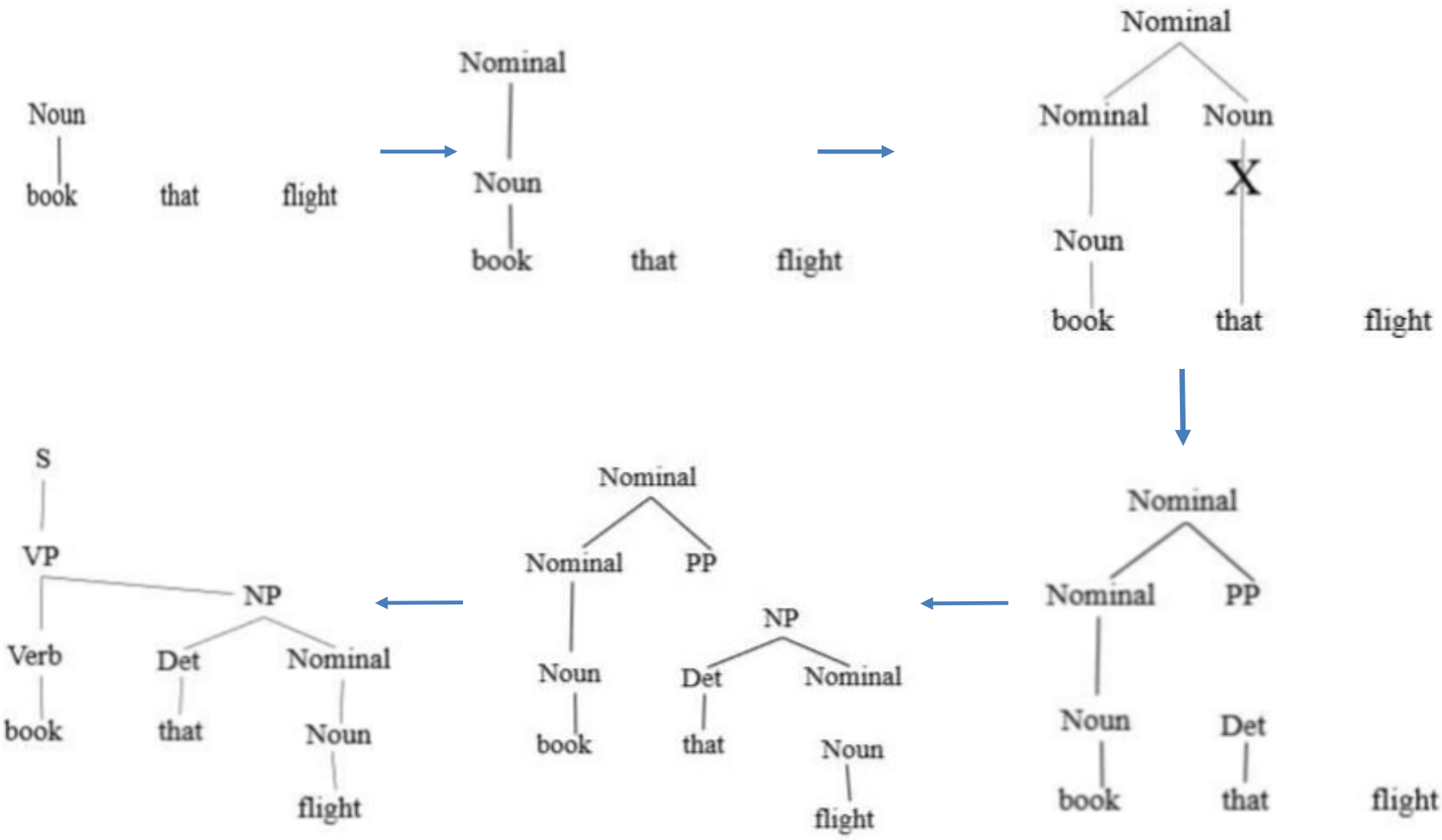
You could build a bottom-up parser simply by formulating this matching process as a search process.

The state would simply consist of a symbol list, starting with the words in the sentence.

Successor states could be generated by exploring all possible ways to

- Rewrite a word by its possible lexical categories
- Replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol

Bottom up parsing



Top Down vs Bottom Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

Chart parsing

- Parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily.
- To avoid this problem, a data structure called a chart is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Chart Parsing

- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to "match"
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

Chart Parsing

- Assume you are parsing a sentence that starts with an ART.
- With this ART as the key, rules 2 and 3 are matched because they start with ART.
- To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART.
- You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record

  2'. NP -> ART o ADJ N

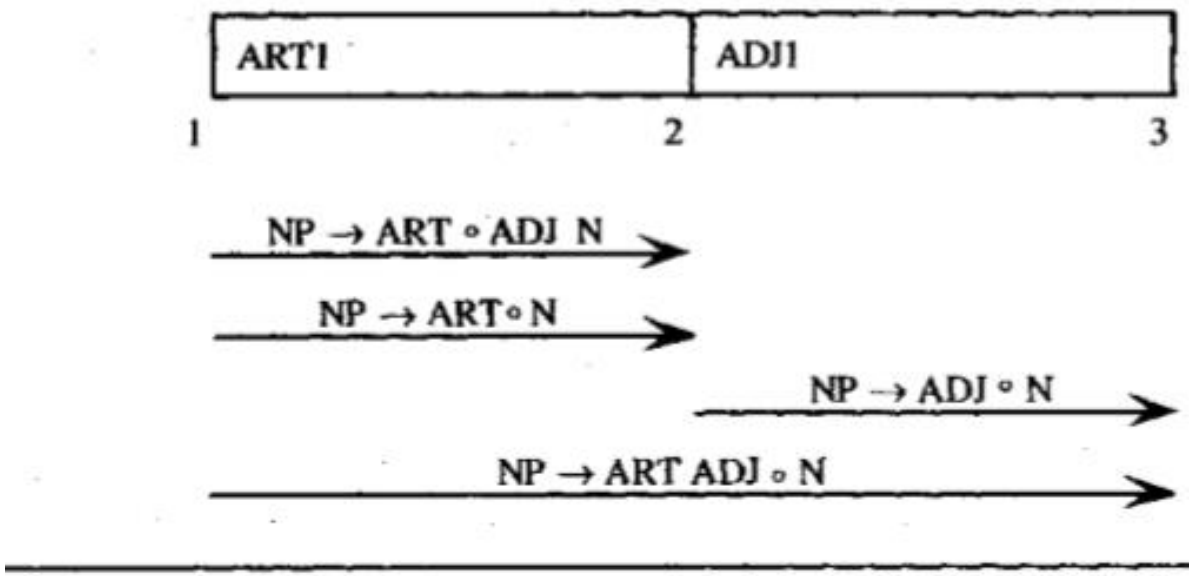
  3'. NP -> ART o N

- If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2 may be extended to give

  2''. NP -> ART ADJ o N

Chart- active arcs

- The chart maintains the record of all the constituents derived from the sentence so far in the parse.
- Maintains the record of rules that have matched partially but are not complete. These are called the active arcs.



1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N

4. VP -> V

5. VP -> V NP

Figure 3.9 The chart after seeing an ADJ in position 2

Chart Parsing

Extending Active Arcs with a Key:

- Each **Active Arc** has the form: $[A \rightarrow X_1 ... \bullet C ... X_m](p_i, p_j)$
- A Key constituent has the form: $C(p_i, p_j)$
- When processing the Key $C(p_1, p_2)$, we search the active arc list for an arc $[A \rightarrow X_1 ... \bullet C ... X_m](p_0, p_1)$, and then create a new active arc $[A \rightarrow X_1 ... C \bullet ... X_m](p_0, p_2)$
- If the new active arc is a completed rule: $[A \rightarrow X_1 ... C \bullet](p_0, p_2)$, then we add $A(p_0, p_2)$ to the Agenda
- After "using" the key to extend all relevant arcs, it is entered into the Chart

Chart Parsing

Steps in the Process:

- Input is processed left-to-right, one word at a time

1. Find all POS of word (terminal-level)
2. Initialize Agenda with all POS of the word
3. Pick a Key from the Agenda
4. Add all grammar rules that start with the Key as active arcs
5. Extend any existing active arcs with the Key
6. Add LHS constituents of newly completed rules to the Agenda
7. Add the Key to the Chart
8. If Agenda not empty - goto (3), else goto (1)

Chart parsing example

The input: "x = The large can can hold the water"

POS of Input Words:

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

$$
\begin{aligned}
(1)\ S &\rightarrow NP\ VP \\
(2)\ NP &\rightarrow ART\ ADJ\ N \\
(3)\ NP &\rightarrow ART\ N \\
(4)\ NP &\rightarrow ADJ\ N \\
(5)\ VP &\rightarrow AUX\ VP \\
(6)\ VP &\rightarrow V\ NP
\end{aligned}
$$

Chart parsing example

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

The input: "$x =$ **The** large can can hold the water"

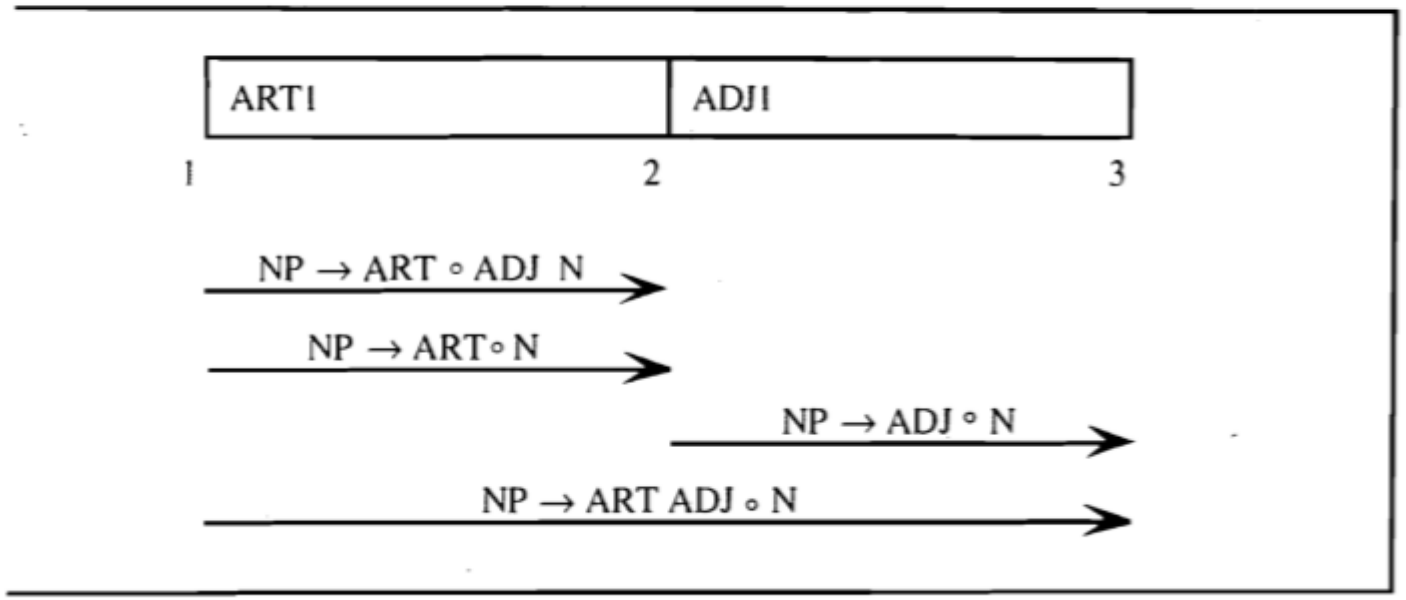


Figure 3.9 The chart after seeing an ADJ in position 2

Chart parsing example

The input: "x = **The large can** can hold the water"

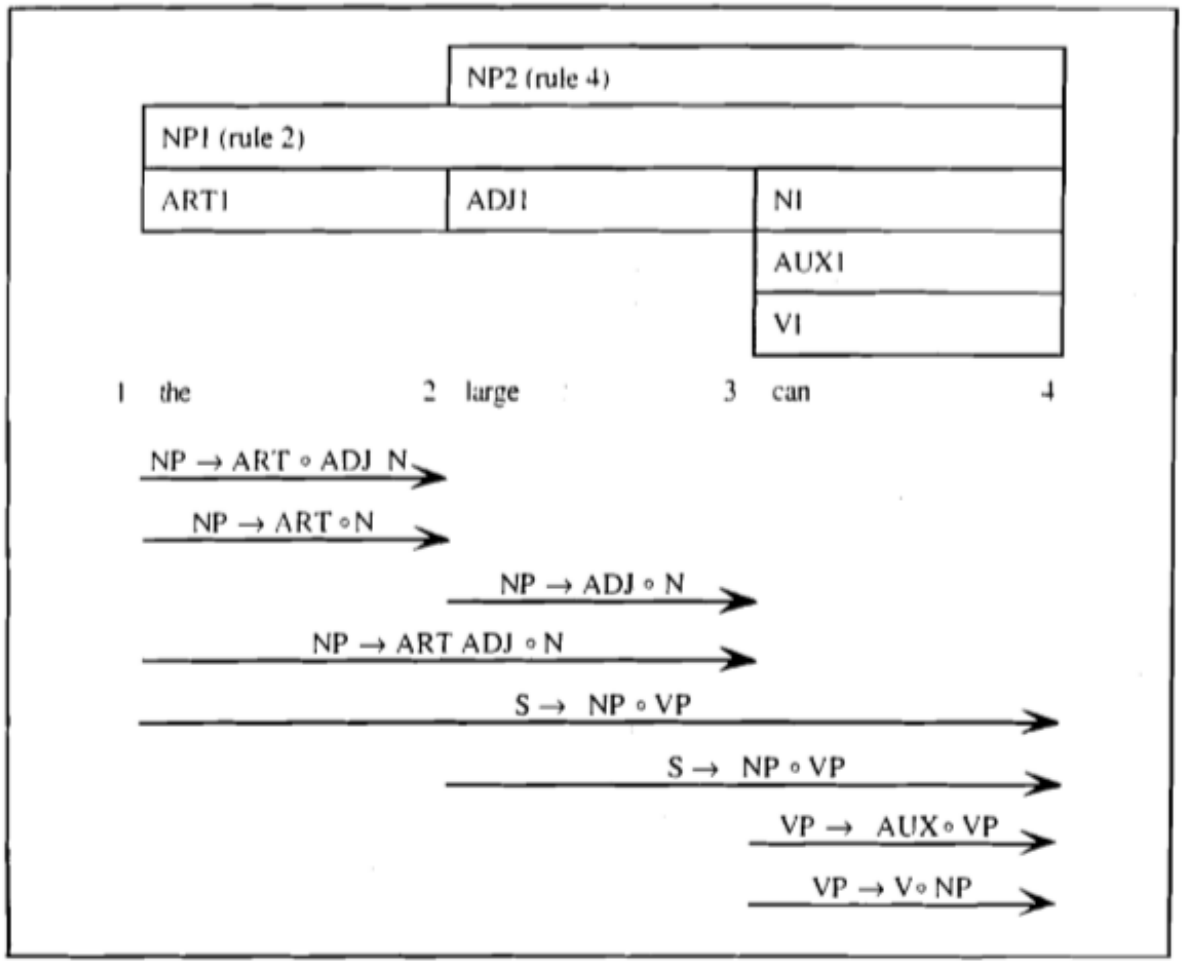


Figure 3.12 After parsing *the large can*

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

Chart parsing example

The input: "$x =$ **The large can can hold** the water"

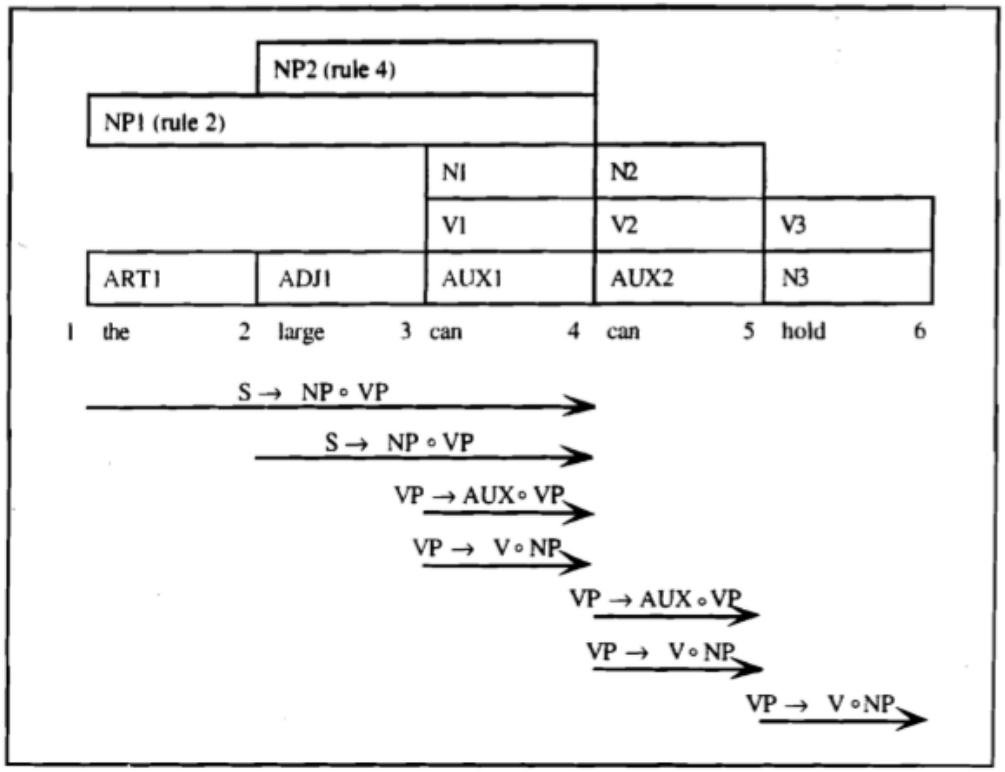


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

(1) $S \rightarrow NP\ VP$

(2) $NP \rightarrow ART\ ADJ\ N$

(3) $NP \rightarrow ART\ N$

(4) $NP \rightarrow ADJ\ N$

(5) $VP \rightarrow AUX\ VP$

(6) $VP \rightarrow V\ NP$

Chart parsing example

The input: "x = **The large can can hold the water**"

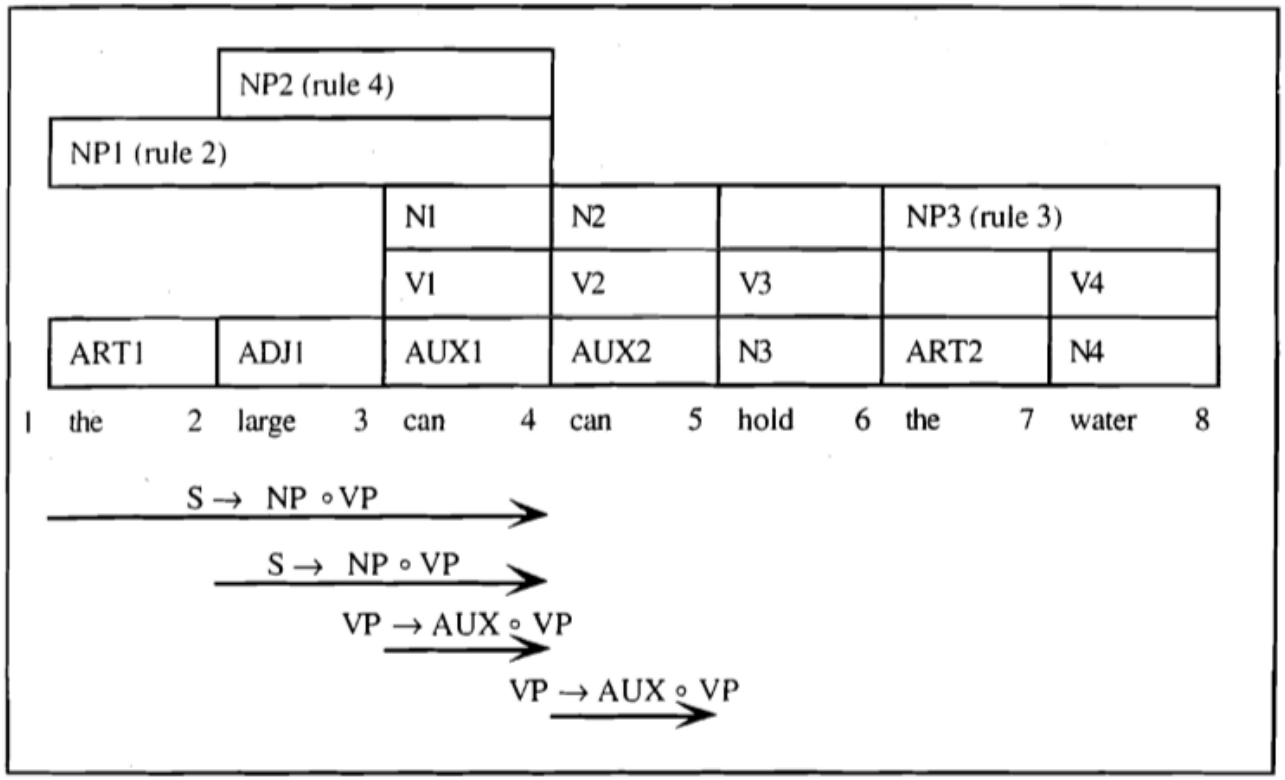


Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

Final Chart

The input: "$x =$ **The large can can hold the water**"

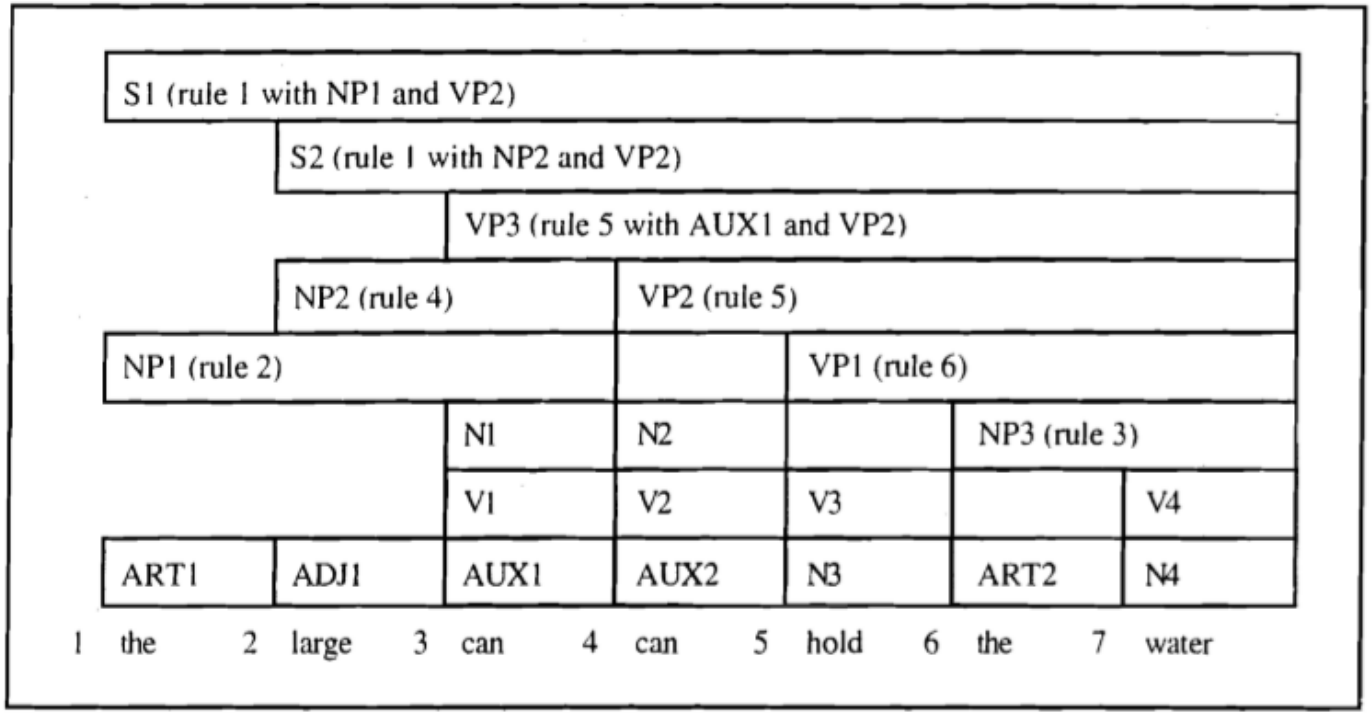


Figure 3.15 The final chart

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

Top down chart parsing

- Top-down methods have the advantage of being highly predictive.
- A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered.

Top down arc induction

- Consider this new algorithm operating with the same grammar on “The large can can hold the water.”
- In the initialization stage, an arc labeled S -> o NP VP is added. Then, active arcs for each rule that can derive an NP are added: NP -> o ART ADJ N, NP -> o ART N,

Top-Down Arc Introduction Algorithm

To add an arc S -> C_1 ... o C_1 ... C_n ending at position j, do the following:

For each rule in the grammar of form C_i -> X_1 ... X_k, recursively add the new arc C_i -> o X_1 ... X_k from position j to j.

Top down chart parsing

Initialization: For every rule in the grammar of form S -> X_1 ... X_k, add an arc labeled S -> o X_1 ... X_k using the arc introduction algorithm.

Parsing: Do until there is no input left:

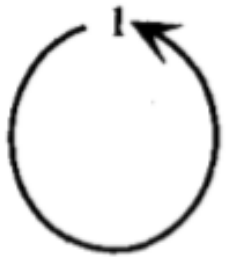
1. If the agenda is empty, look up the interpretations of the next word and add them to the agenda.

2. Select a constituent from the agenda (call it constituent C).

3. Using the arc extension algorithm, combine C with every active arc on the chart. Any new constituents are added to the agenda.

4. For any active arcs created in step 3, add them to the chart using the top-down arc introduction algorithm.

Top down chart parsing example



S → ° NP VP
NP → ° ADJ N
NP → ° ART N
NP → ° ART ADJ N

Figure 3.23 The initial chart

Top down chart parsing example

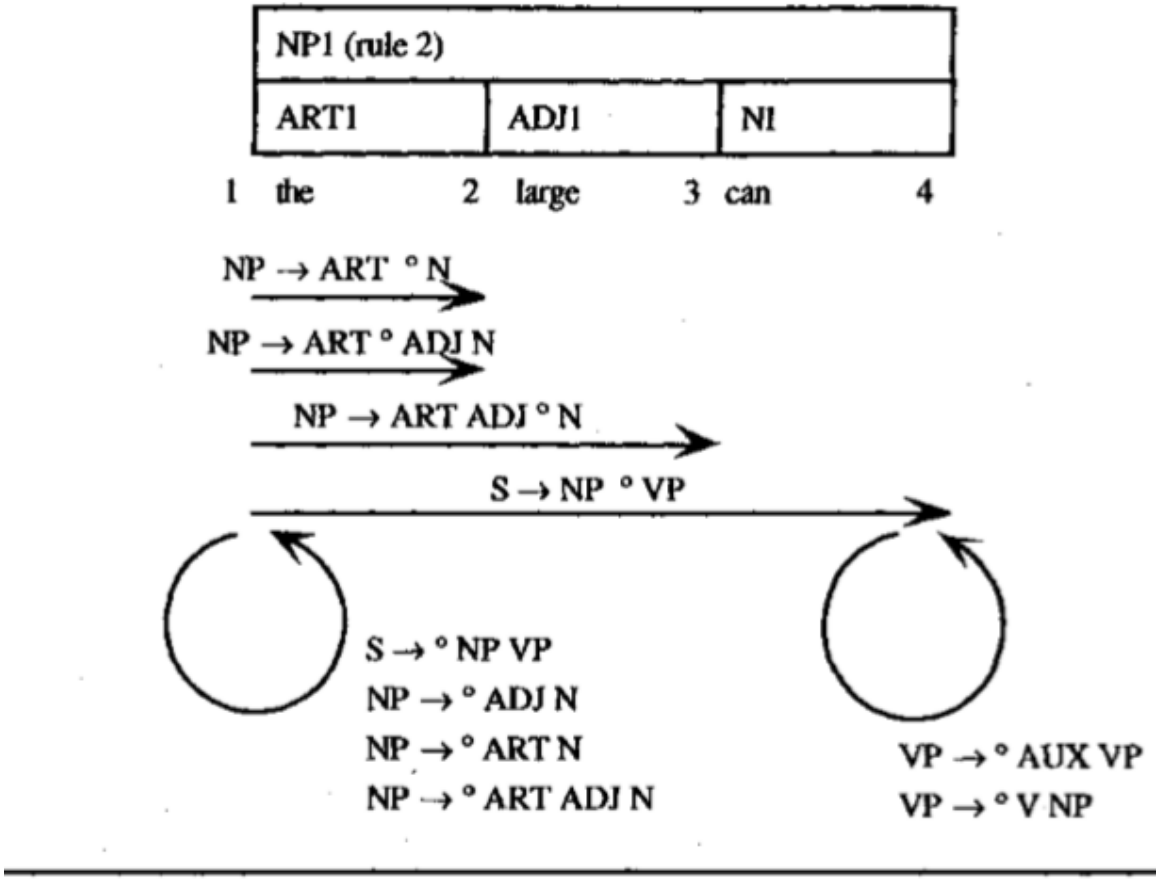


Figure 3.24 The chart after building the first NP

Top down chart parsing example

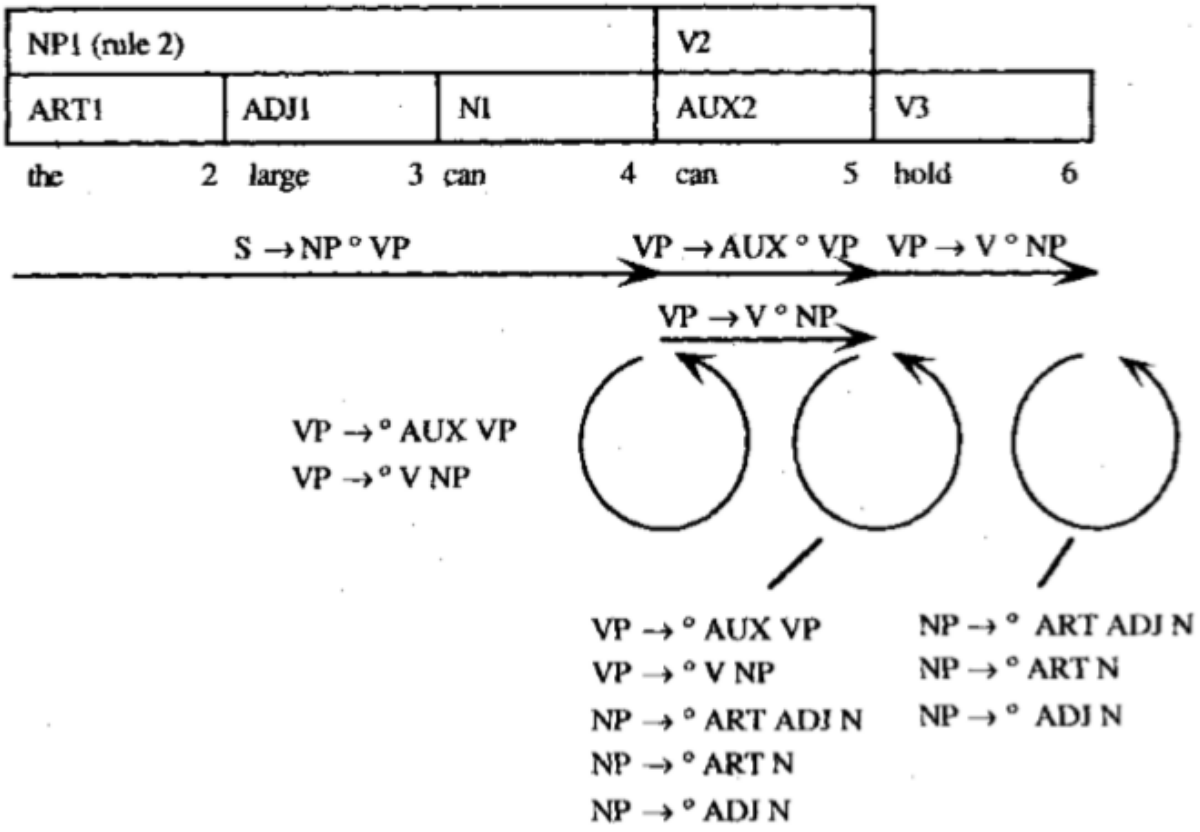


Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

Top down chart parsing example

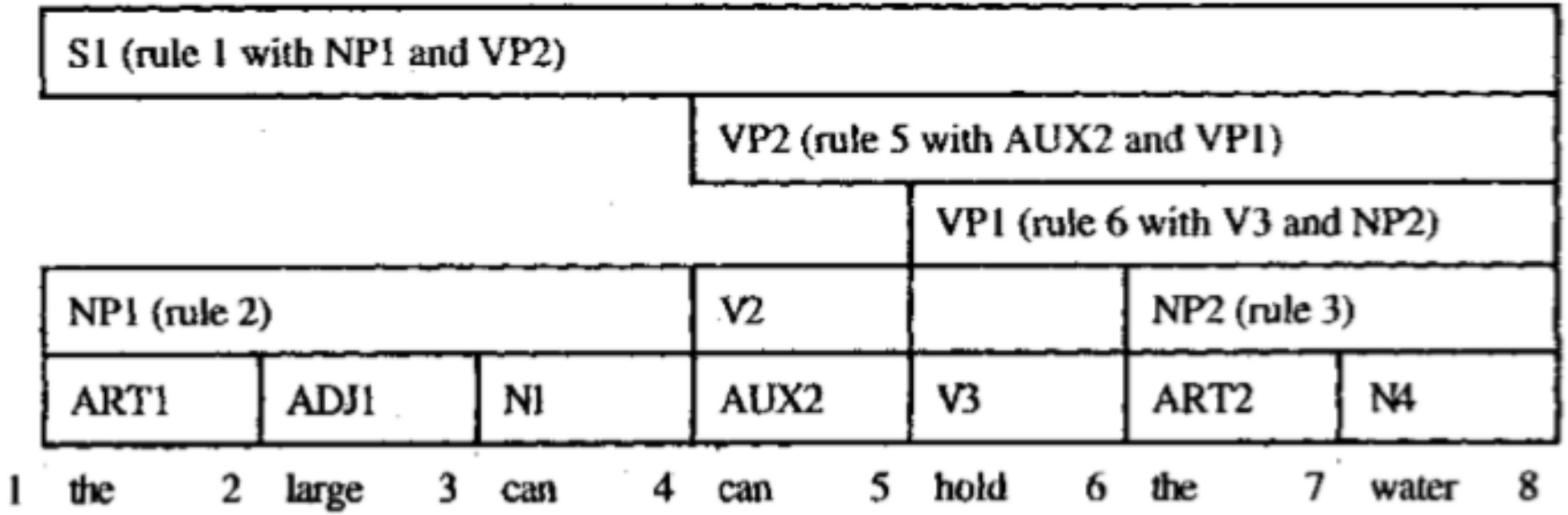


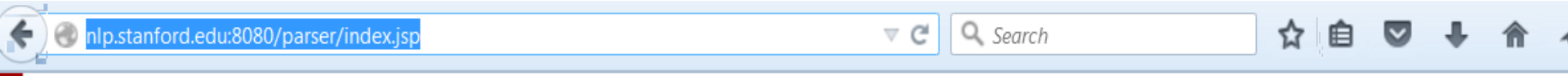
Figure 3.26 The final chart for the top-down filtering algorithm

Chart Parsing

- When a chart parser begins parsing a text, it creates a new (empty) chart, spanning the text.
- It then incrementally adds new edges to the chart.
- A set of "chart rules" specifies the conditions under which new edges should be added to the chart.
- Once the chart reaches a stage where none of the chart rules adds any new edges, parsing is complete.

Advantages of Chart Parsing

- No repeated computation of same sub problem
- Deals well with left-recursive grammars
- Deals well with ambiguity
- No backtracking necessary



Stanford Parser

Please enter a sentence to be parsed:

I love natural language processing class

Language: English Sample Sentence Parse

Your query

I love natural language processing class

Tagging

```
I/PRP  love/VBP  natural/JJ  language/NN  processing/NN  class/NN
```

Parse

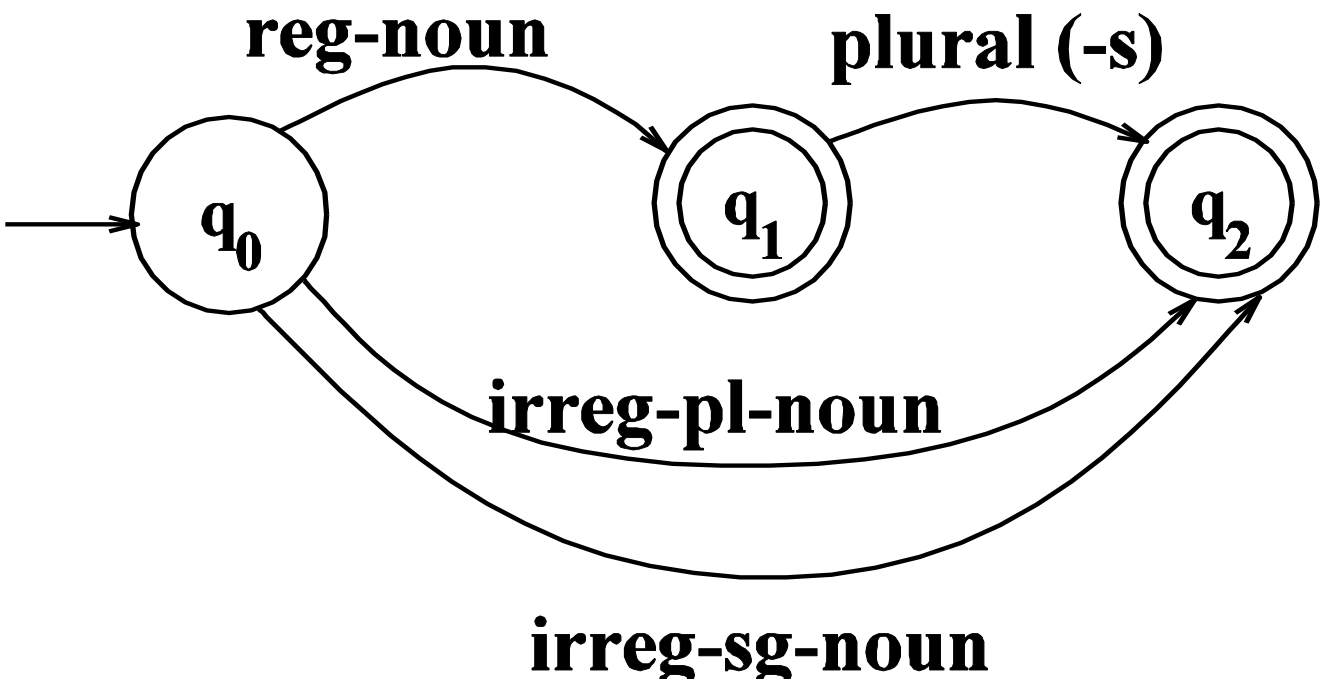
```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBP love)
      (NP (JJ natural) (NN language) (NN processing) (NN class)))))
```

Parsing Implementation Demo

Morphological Processing

- Not only are there a large number of words, but each word may combine with affixes to produce additional related words.
- One way to address this problem is to preprocess the input sentence into a sequence of morphemes.
- A word may consist of single morpheme, but often a word consists of a root form plus an affix (prefix or suffix).
- For instance, the word eaten consists of the root form eat and the suffix -en, which indicates the past participle form.
- Without any pre-processing, a lexicon would have to list all the forms of eat, including eats, eating, ate, and eaten.
- With preprocessing, there would be one morpheme eat that may combine with suffixes such as -ing, -s, and -en, and one entry for the irregular form ate. Thus the lexicon would only need to store two entries (eat and ate) rather than four.

Simple Rules



Parsing/Generation

- Usually if we find some string in the language we need to find the structure in it (parsing)
- Or we have some structure and we want to produce a surface form (production/generation)

- Example
  - From "`cats`" to "`cat +N +PL`" and back

➔**Morphological analysis**

References

- http://www.ai.mit.edu/courses/6.863/lecture7-03.pdf
- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin
- Natural language understanding by James Allen
- https://nlp.stanford.edu/software/lex-parser.shtml
- https://www.nltk.org/api/nltk.parse.html

Thank you for your time!!