

自我來黃州已過三寒  
食年、欲惜春、意不  
容惜今年又苦雨、多月社  
簫瑟、以聞海棠花、泥  
污遊支雪、閣中偷負  
多夜半、真有力何殊、少  
年、病起頭、白  
春江欲入户、雨勢未  
止、雨小屋如漁舟、濛  
水雲裏、空庭竟寒、寒  
破竈燒酒、華、那  
知是寒食、但見烏  
銜、市、天門深  
九重、清夢在、在、方寸、幾  
哭、淪、窮、所、不、吹、不  
起

右黃州寒食二首

# 计算语言学

## Computational Linguistics

教师：孙茂松

Tel: 62781286


Email: [sms@tsinghua.edu.cn](mailto:sms@tsinghua.edu.cn)

TA：林衍凯

Email: [linyankai423@qq.com](mailto:linyankai423@qq.com)

# 郑重声明

- 此课件仅供选修清华大学计算机系研究生课《计算语言学》(70240052)的学生个人学习使用，所以只允许学生将其下载、存贮在自己的电脑中。未经孙茂松本人同意，任何人不得以任何方式扩散之（包括不得放到任何服务器上）。否则，由此可能引起的一切涉及知识产权的法律责任，概由该人负责。
- 此课件仅限孙茂松本人讲课使用。除孙茂松本人外，凡授课过程中，PTT文件显示此《郑重声明》之情形，即为侵权使用。



# **第三章**

## **句法分析基本算法**

### **(Part 1)**

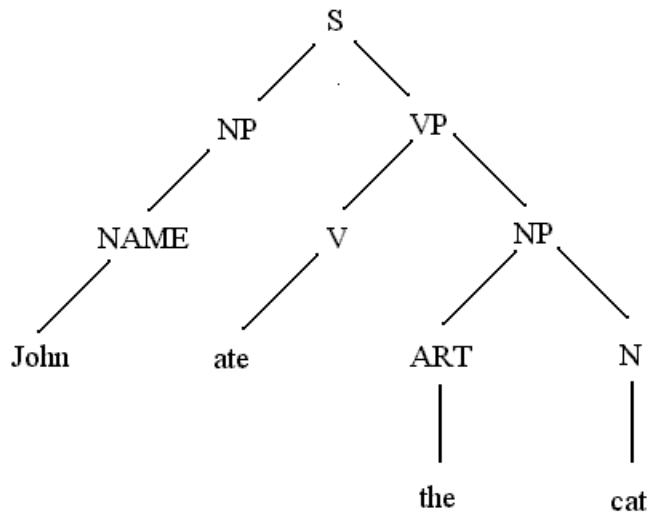
## 3.1. Grammars and Sentence Structure



- **Grammar:** a **formal specification** of the structures allowable (i.e. well-formed structures) in the language.
- **Parsing technique:** the method of analyzing a sentence to determine its structure **according to the grammar.**
- **Parser:** a program for analyzing sentences given a grammar.
  - \* may just give yes/no answer to the question: “Is this sentence legal according to the given grammar?”(an **acceptor**)
  - \* may also produce a structure description ("**parse tree**") for correct sentences.

## 3.1. Grammars and Sentence Structure

**Tree (parse tree):** *John ate the cat.*



**List notation of sentences:**

(S (NP (NAME John))  
(VP (V ate)  
(NP (ART the)  
(N cat))))

or:

(S (NP (NAME John)) (VP (V ate) (NP (ART the) (N cat))))

# 3.1. Grammars and Sentence Structure



- Terms about the tree

**Nodes:** (labeled nodes)

**Links** among nodes: from a parent node to a child node.

**Root** of the tree: the node at the top

**Leaves** of the tree: nodes at the bottom

**Parent** nodes vs. **Child** nodes

**Ancestor** of a node

A node is **dominated** by its ancestor nodes.

# 3.1. Grammars and Sentence Structure

## ● Describing Syntax

A grammar is composed of a set of **rewrite rules**.

1. $S \rightarrow NP VP$	5. $NAME \rightarrow John$
2. $VP \rightarrow V NP$	6. $V \rightarrow ate$
3. $NP \rightarrow NAME$	7. $ART \rightarrow the$
4. $NP \rightarrow ART N$	8. $N \rightarrow cat$

(Grammar 3.1: simple grammar)

**Terminal symbols:** symbols that cannot be decomposed in a grammar.

**Nonterminal symbols:** all other symbols in the grammar.

**Lexical symbols:** the grammatical symbols such as N and V that describe word categories. **a subset of nonterminal symbols.**

**Start symbol:** always S.

## 3.1. Grammars and Sentence Structure

- A grammar is said to **derive** a sentence if there is a sequence of rules that allow to rewrite the start symbol into the sentence (by repeatedly perform rewriting process).

Or, the sentence **is derived** from the grammar with start symbol S.

<b>S</b>	/* generation from S, <b>top-down search</b> strategy*/
=> <b>NP VP</b>	(rewriting S)
=> <b>NAME VP</b>	(rewriting NP)
=> John <b>VP</b>	(rewriting NAME)
=> John <b>V NP</b>	(rewriting VP)
=> John ate <b>NP</b>	(rewriting V)
=> John ate <b>ART N</b>	(rewriting NP)
=> John ate the <b>N</b>	(rewriting ART)
=> John ate the cat	(rewriting N)



# 3.1. Grammars and Sentence Structure

## Bottom-up search strategy:

=> S (rewriting NP VP)  
=> NP VP (rewriting V NP)  
=> NP V NP (rewriting ART N)  
=> NP V ART N (rewriting NAME)  
=> NAME V ART N (rewriting cat)  
=> NAME V ART cat (rewriting the)  
=> NAME V the cat (rewriting ate)  
=> NAME ate the cat (rewriting John)

如何系统地穷尽  
所有的搜索可能性？

**John** ate the cat

1. $S \rightarrow NP VP$	5. $NAME \rightarrow John$
2. $VP \rightarrow V NP$	6. $V \rightarrow ate$
3. $NP \rightarrow NAME$	7. $ART \rightarrow the$
4. $NP \rightarrow ART N$	8. $N \rightarrow cat$

# 3.1. Grammars and Sentence Structure



Grammar 3.1:

$\mathbf{N} = \{ S \text{ NP VP N V ART NAME } \}$

$\mathbf{T} = \{ \textit{ate cat John the} \} \quad \mathbf{S} = \{ S \}$

$\mathbf{R} = \{ S \rightarrow \text{NP VP}, \text{VP} \rightarrow \text{V NP}, \text{NP} \rightarrow \text{ART N}, \text{NP} \rightarrow \text{NAME}, \\ \text{NAME} \rightarrow \text{John}, \text{V} \rightarrow \text{ate}, \text{ART} \rightarrow \text{the}, \text{N} \rightarrow \text{cat} \}$

Language generated by Grammar3.1: total 4 sentences  
{John ate the cat, John ate John, The cat ate John, The cat ate the cat}  
ill-formed: {John ate, Cat ate, The cat ate, The cat ate the John, ...}

## 3.2. What Makes a Good Grammar

- Generality: the range of sentences the grammar analyzes correctly.
- Selectivity: the range of non-sentences it identifies as problematic.
- Understandability: the simplicity of the grammar.

Grammar 3.1 becomes Grammar 3.1\*+Lexicon 3.1

Grammar 3.1\*:

- |                          |                           |
|--------------------------|---------------------------|
| 1. $S \rightarrow NP VP$ | 3. $NP \rightarrow NAME$  |
| 2. $VP \rightarrow V NP$ | 4. $NP \rightarrow ART N$ |

Lexicon 3.1\* (lexical symbols):

John: NAME      ate: V      the: ART      cat: N

### 3.3. A Top-Down Parser



- Just an acceptor NOW: to identify if the input sentence is grammatical or not, but not construct the tree in the parsing process.
- Top-down: starting from S, attempt to rewrite it into a sequence of terminal symbols that matches the input sentence.
- The state of the parse at any given time is represented as a **list** of symbols that are the results applied so far, called the **symbol list**.

## 3.3. A Top-Down Parser

Example: parsing “*John ate the cat*” with grammar 3.1:

Step	Symbol list (state of the parse)
Initial	(S)
after applying the rule $S \rightarrow NP VP$	(NP VP)
After applying the rule $NP \rightarrow ART N$	(ART N VP)
After applying the rule $VP \rightarrow V NP$	(ART N V NP)
After applying the rule $NP \rightarrow ART N$	(ART N V ART N)
After applying the rule ART $\rightarrow$ the, V $\rightarrow$ ate, and N $\rightarrow$ cat	(the cat ate the cat)
Try other combinations of rules .....	

1. $S \rightarrow NP VP$	5. $NAME \rightarrow John$
2. $VP \rightarrow V NP$	6. $V \rightarrow ate$
3. $NP \rightarrow NAME$	7. $ART \rightarrow the$
4. $NP \rightarrow ART N$	8. $N \rightarrow cat$

Low efficiency!

Improvement: A **better** algorithm checks the input as soon as it can!

### 3.3. A Top-Down Parser

- The state of the parse (augmented):

The state of the parse is now defined as a **pair**:

(a **symbol list** as above      a **number** indicating the word position in the sentence)

i.e.: (**symbol list**   **word position**)

**Positions** fall between the words, with 1 being the position before the first word:

1 John 2 ate 3 the 4 cat 5

Example: a typical parse state

((N VP) 2)

indicating the parser needs to find an N followed by a VP, starting at position 2.

## 3.3. A Top-Down Parser

- Example: parsing “*John ate the cat*” with grammar 3.1

Step	Current state	Input sentence
Initial	((S) 1)	1 John 2 ate 3 the 4 cat 5 ↑
after applying the rule $S \rightarrow NP VP$	((NP VP) 1)	1 John 2 ate 3 the 4 cat 5 ↑
after applying the rule $NP \rightarrow NAME$	((NAME VP) 1)	1 John 2 ate 3 the 4 cat 5 ↑
after applying the rule $NAME \rightarrow John$	((VP) 2)	1 John 2 ate 3 the 4 cat 5 ↑
After applying the rule $VP \rightarrow V NP$	((V NP) 2)	1 John 2 ate 3 the 4 cat 5 ↑
After applying the rule $V \rightarrow ate$	((NP) 3)	1 John 2 ate 3 the 4 cat 5 ↑
After applying the rule $NP \rightarrow ART N$	((ART N) 3)	1 John 2 ate 3 the 4 cat 5 ↑
After applying the rule $ART \rightarrow the$	((N) 4)	1 John 2 ate 3 the 4 cat 5 ↑
After applying the rule $N \rightarrow cat$	(( ) 5)	1 John 2 ate 3 the 4 cat 5 ↑
Success! The input sentence is grammatical.		

### 3.3. A Top-Down Parser



- Algorithm(informal)

**IF** the first symbol in the current state is a lexical symbol  
**THEN** check it with the word( in input sentence) indicated  
by the word position  
**IF** succeeded  
**THEN** update the state by removing the first  
symbol and updating the word position  
**ELSE** (i.e., the first symbol is non-terminal symbol)  
rewrite it by using a rule from the grammar



## 3.3. A Top-Down Parser

- A parser must **systematically explore every possible new state.**

**Backtracking:** parsing “*John ate the cat*” with grammar 3.1

Step	Current state	Input sentence
1. Initial	((S) 1)	1 John 2 ate 3 the 4 cat 5 ↑
2. after applying: S → NP VP	((NP VP) 1)	1 John 2 ate 3 the 4 cat 5 ↑
3. after applying: NP → NAME	((NAME VP) 1)	1 John 2 ate 3 the 4 cat 5 ↑
4. after applying: NAME → John	((VP) 2)	1 John 2 ate 3 the 4 cat 5 ↑
5. after applying: VP → V NP	((V NP) 2)	1 John 2 ate 3 the 4 cat 5 ↑
6. after applying: V → ate	((NP) 3)	1 John 2 ate 3 the 4 cat 5 ↑
7. after applying: NP → NAME	((NAME) 3)	1 John 2 ate 3 the 4 cat 5 ↑
Fail in matching the word indicated by the word position. How about the next step? So, we need backtracking!		

### 3.3. A Top-Down Parser

At stage 7, we need to generate all possible new states: one is as the current state, others will be as backup states for later use.

**Backup states:** ((ART N) 3) (as the result of applying the rule  $NP \rightarrow ART\ N$  at word position 3)

If the algorithm ever reach a situation where the current state cannot lead to a solution, then simply pick a new state current state from the list of backup states.

8. After applying: NP → ART N	((ART N) 3)	1 John 2 ate 3 the 4 cat 5 <div style="text-align: center;">↑</div>
9. After applying: ART → the	((N) 4)	1 John 2 ate 3 the 4 cat 5 <div style="text-align: right; margin-right: 10%;">↑</div>
10. After applying: N → cat	((()) 5)	1 John 2 ate 3 the 4 cat 5 <div style="text-align: right;">↑</div>

..... Success! The input sentence is grammatical.

## 3.3. A Top-Down Parser

### ● Data structure for Top-Down Parser

in Backus-Naur Form(BNF forms):

```
<possibilities list>:=(<current state><backup states>)  
<current state>:=(<symbol list><current word position>)  
<backup states>:=(<symbol list><current word position>)*  
<symbol-list>:=(<non-terminal symbol string>)  
<current-position>:=<integer>
```

Conventions of Backus-Naur Form (BNF):

1. ‘::=’: definition
2. between ‘<’ and ‘>’: items
3. ‘\*’: items that can repeat zero or more times
4. vertical line ‘|’: separates alternatives
5. ....

## 3.3. A Top-Down Parser

### ● Algorithm of Top-Down Parser(formal)

The algorithm starts with the initial state ((S) 1) and no backup states as the possibilities list.

1. Select the current state: Take the first state off the possibilities list and call it C. If C is empty, then the algorithm fails (that is, no successful parse is possible).
2. If C consists of an empty symbol list and the word position is at the end of the sentence, then the algorithm succeeds.
3. Otherwise, generate the next possible states.
  - 3.1. If the first symbol on the symbol list of C is a lexical symbol, and the **current** word in the sentence can be in that class, then create a new state by removing the first symbol from the symbol list and updating the word position, and add it to the possibilities list.
  - 3.2. Otherwise, if the first symbol on the symbol list of C is a non-terminal, generate a new state **for each rule in the grammar** that can rewrite that non-terminal symbol and add them all to the possibilities list.
4. Go to 1

## 3.3. A Top-Down Parser



- Example: Top-down depth-first parsing for “The old man cried”

Grammar 3.2:

R1.  $S \rightarrow NP VP$

R4.  $VP \rightarrow V$

R2.  $NP \rightarrow ART N$

R5.  $VP \rightarrow V NP$

R3.  $NP \rightarrow ART ADJ N$

Lexicon (Lexical symbol):

the: ART

old: ADJ, N

man: N, V

cried: V

### 3.3. A Top-Down Parser

Step	Current state	Backup states	Input sentence
1. Initial	((S) 1)		1 The 2 old 3 man 4 cried 5 ↑
2. R1	((NP VP) 1)		1 The 2 old 3 man 4 cried 5 ↑
3. R2 R3	((ART N VP) 1)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ↑
4. ART <i>the</i>	(( N VP) 2)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART ↑
5. N old	((VP) 3)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N ↑
6. R4 R5	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N ↑
7. V <i>man</i>	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑

### 3.3. A Top-Down Parser

8. use backup	((V NP) 3)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N ↑
9. N <i>man</i>	((NP) 4)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑
10. R2 R3	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑
11. use backup	((ART ADJ N) 4)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑
12. use backup	((ART ADJ N VP) 1)		1 The 2 old 3 man 4 cried ↑
13. ART <i>The</i>	((ADJ N VP) 2)		1 The 2 old 3 man 4 cried 5 ART ↑
14. ADJ <i>old</i>	((N VP) 3)		1 The 2 old 3 man 4 cried 5 ART ADJ ↑
15. N <i>man</i>	((VP) 4)		1 The 2 old 3 man 4 cried 5 ART ADJ N ↑
16. R4 R5	((V) 4)	((V NP) 4)	1 The 2 old 3 man 4 cried 5 ART ADJ N ↑
17. V <i>cried</i>	(( ) 5)	((V NP) 4)	1 The 2 old 3 man 4 cried 5 ART ADJ N V ↑

Succeeded in step 17! The input sentence is grammatical.

## 3.3. A Top-Down Parser

### ● Parsing as a **Search** Procedure

Step 1. Select the first state from the possibilities list (and remove it from the list).

Step 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).

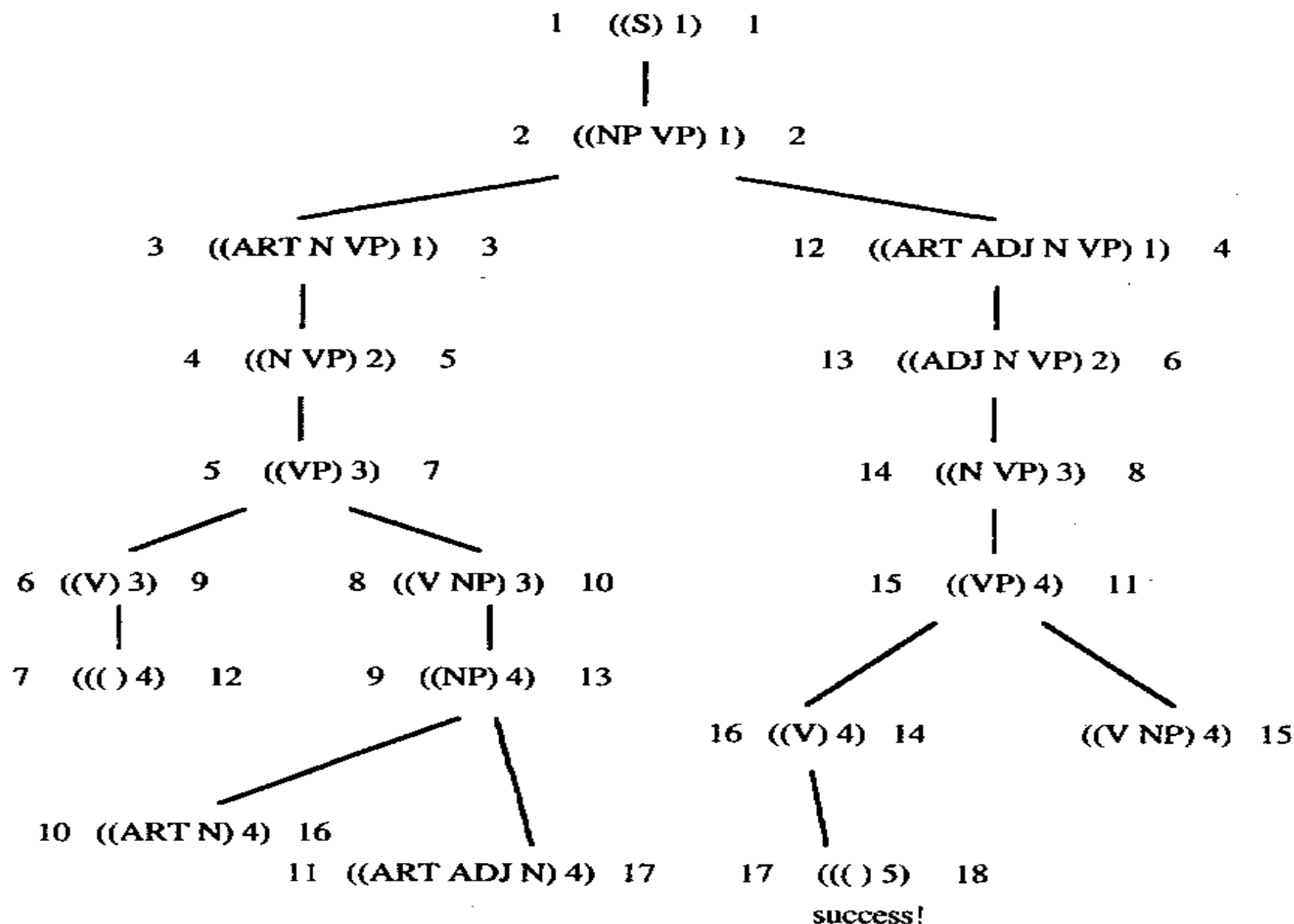
**Note:** If there are more than one grammar rules with the same symbol on the left hand side, then the order of using these rules must be kept unchanged during the whole procedure of parsing.

Step 3. Add the states generated in step 2 to the possibilities list.

**Depth-first strategy:** the possibilities list is a **stack**.

**Breadth-first strategy:** the possibilities list is a **queue**.





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

### 3.3. A Top-Down Parser

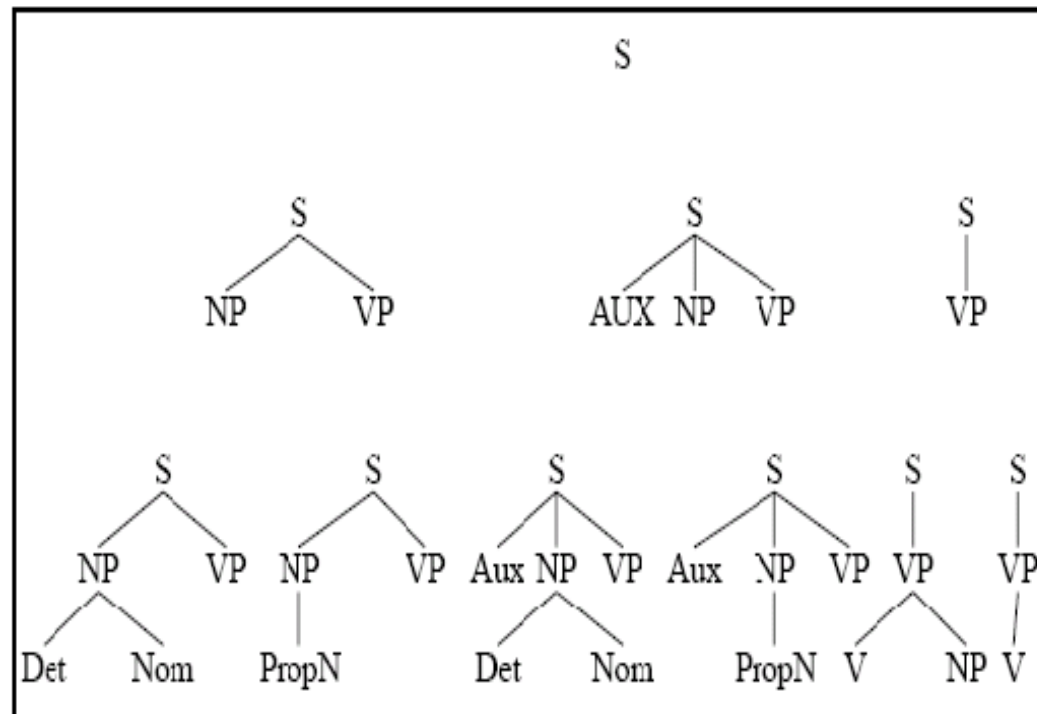
- **优点:** 不会浪费时间去搜索一个不可能以S为根的树.即:绝不会搜索那些在以S为根的树中找不到位置的子树.

- **缺点:**

(1) 花费大量努力去孳生那些与输入不一致的根为S的树.

book that flight

MiniEngGrammar:



$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Noun Nominal$

$NP \rightarrow Proper-Noun$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$Det \rightarrow that | this | a$

$Noun \rightarrow book | flight | meal | money$

$Verb \rightarrow book | include | prefer$

$Aux \rightarrow does$

$Prep \rightarrow from | to | on$

$Proper-Noun \rightarrow Houston | TWA$

$Nominal \rightarrow Nominal PP$

### 3.3. A Top-Down Parser

#### (2) 左递归

直接左递归:

NP  $\rightarrow$  NP PP

VP  $\rightarrow$  VP PP

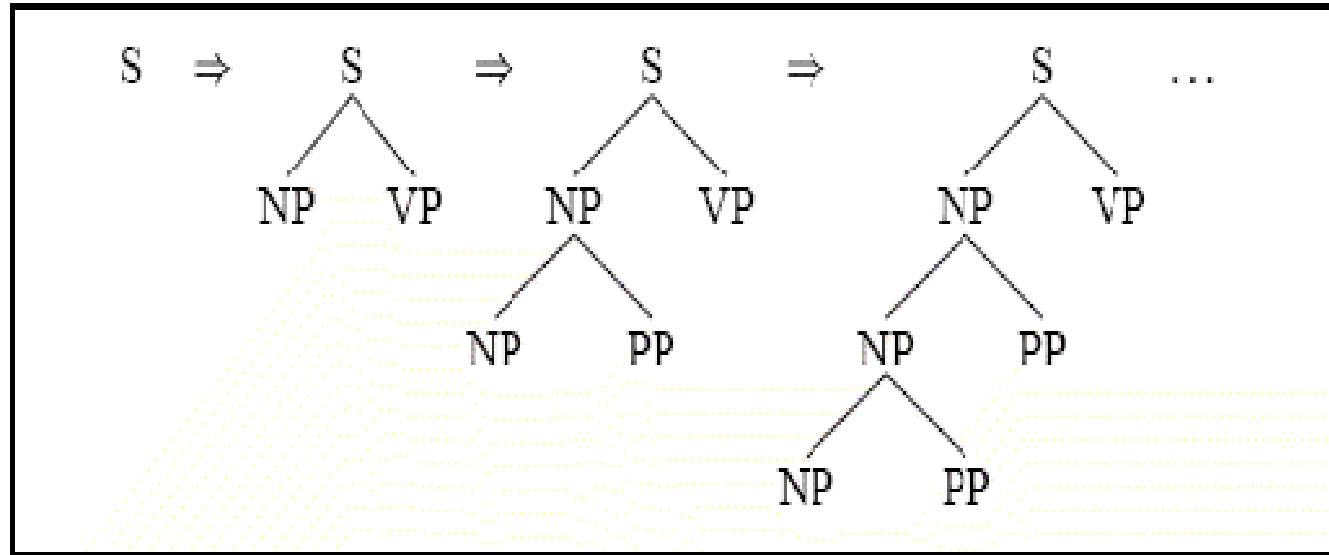
S  $\rightarrow$  S and S

间接左递归:

NP  $\rightarrow$  Det Nominal

Det  $\rightarrow$  NP's

可能导致无限扩展.



对策:

(1) 改左递归为右递归

(2) 限制扩展的深度

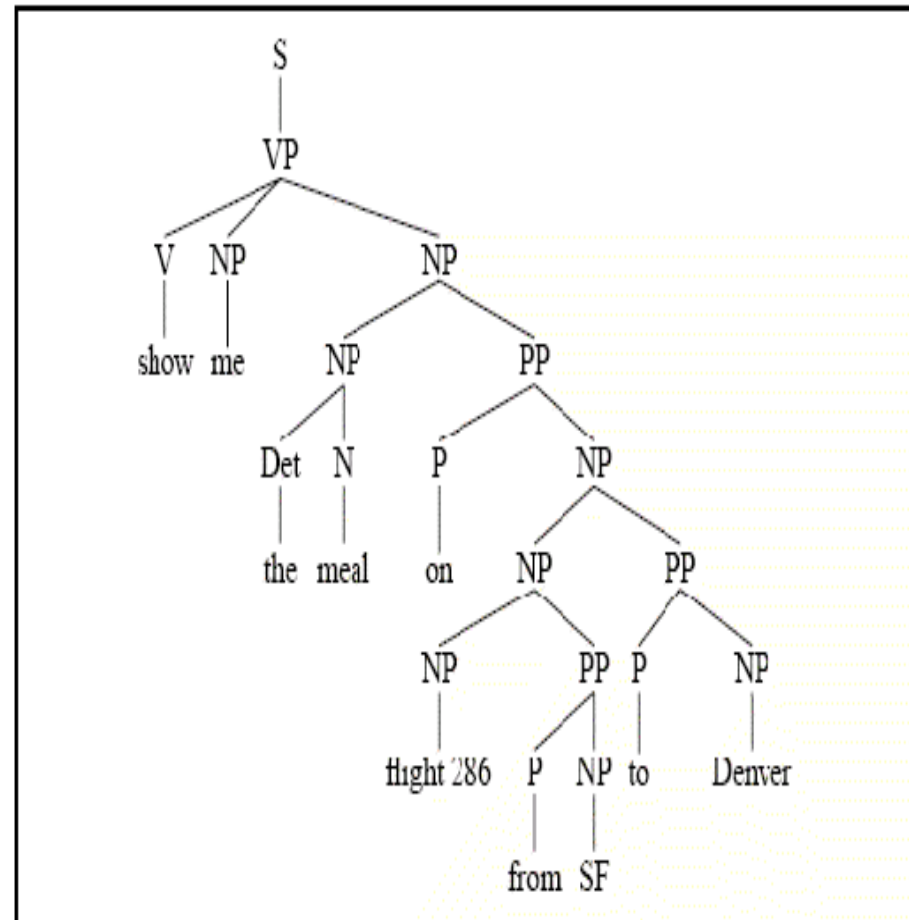
### 3.3. A Top-Down Parser

(3) 广泛存在的歧义可能会导致搜索的组合爆炸

加入两条规则至  
MiniEngGrammar:  
VP → VP PP  
NP → NP PP

*Show me the meal on Flight UA  
386 from San Francisco to Denver.*

PP Attachment



# 3.3. A Top-Down Parser

此类 句子的分析树的数目以所谓的Catalan数按指数增长:

$$C_n = \frac{1}{n+1} \binom{2n}{n}, n \text{ 为 } PP \text{ 数}$$

PP数目    NP分析数目

2	2
3	5
4	14
5	132
6	469
7	1430
8	4867

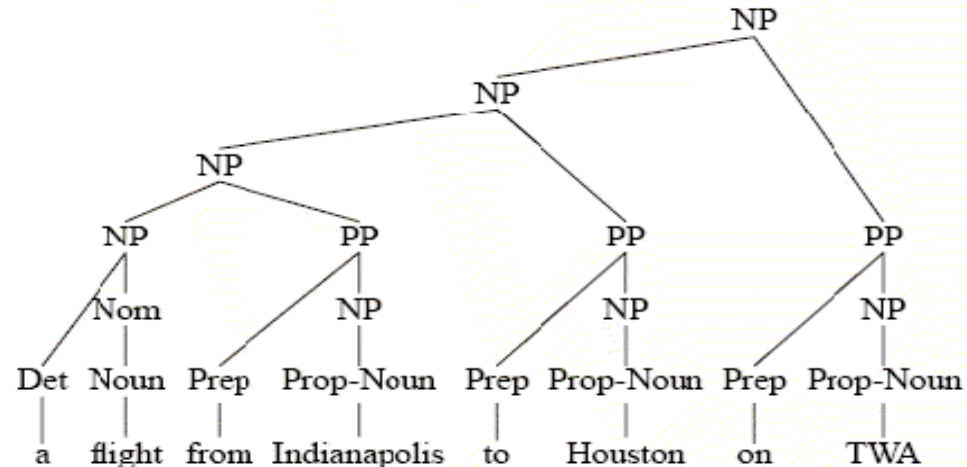
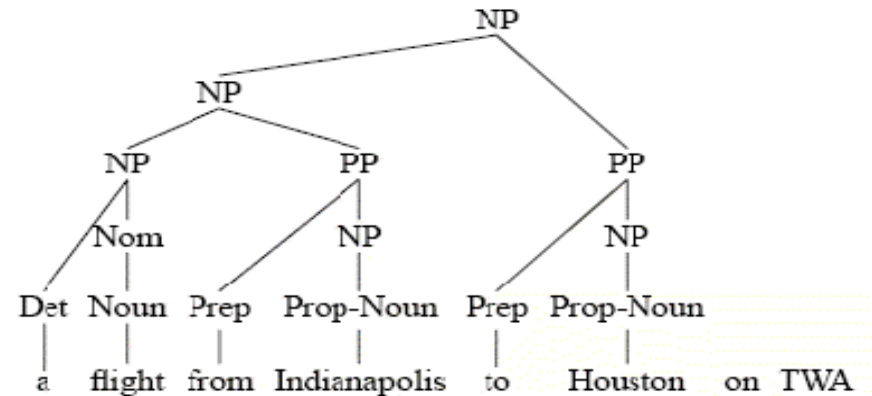
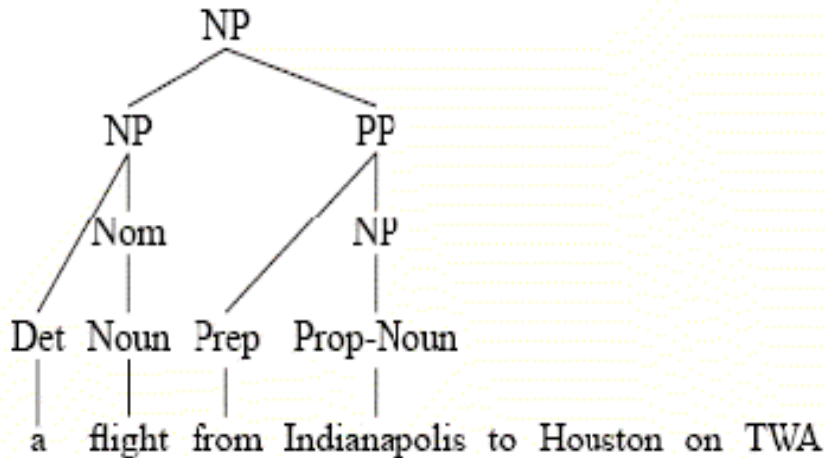
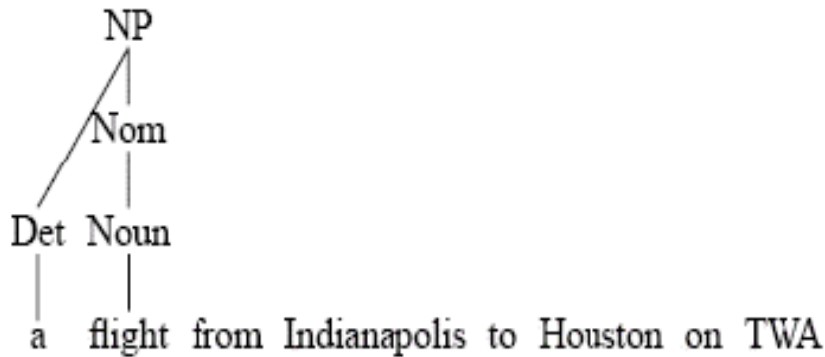
森林?如何表示?

8. use backup	((V NP) 3)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N ↑
9. N man	((NP) 4)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑
10. R2 R3	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑
11. use backup	((ART ADJ N) 4)	((ART ADJ N VP) 1)	1 The 2 old 3 man 4 cried 5 ART N V ↑
12. use backup	((ART ADJ N VP) 1)		1 The 2 old 3 man 4 cried ↑
13. ART The	((ADJ N VP) 2)		1 The 2 old 3 man 4 cried 5 ART ↑
14. ADJ old	((N VP) 3)		1 The 2 old 3 man 4 cried 5 ART ADJ ↑
15. N man	((VP) 4)		1 The 2 old 3 man 4 cried 5 ART ADJ N ↑
16. R4 R5	((V) 4)	((V NP) 4)	1 The 2 old 3 man 4 cried 5 ART ADJ N ↑
17. V cried	(( ) 5)	((V NP) 4)	1 The 2 old 3 man 4 cried 5 ART ADJ N V ↑
Succeeded in step 17! The input sentence is grammatical.			

### 3.3. A Top-Down Parser

#### (4) 子树的重复分析

*A flight from Indianapolis to Houston on TWA*



## 3.4. A Bottom-Up Chart Parser

### Bottom-up:

=> S

1. S → NP VP

5. NAME → John

2. VP → V NP

6. V → ate

=> NP VP

3. NP → NAME

7. ART → the

4. NP → ART N

8. N → cat

=> NP V NP

=> NP V ART N

加入规则 VP->V

=> NAME V ART N

=> NAME V ART **cat**

有回溯的需要

=> NAME V **the** cat

=> NAME **ate** the cat

**John** ate the cat

**shift and reduce**

## 3.4. A Bottom-Up Chart Parser

### Bottom-up的优点:

绝不会搜索那些不是以实际的输入为基础的树.  
可克服左递归

### 缺点:

那些没有希望导致S的子树, 会大量孳生出来.

策略: 通常会利用Top-down, Bottom-up的某种结合

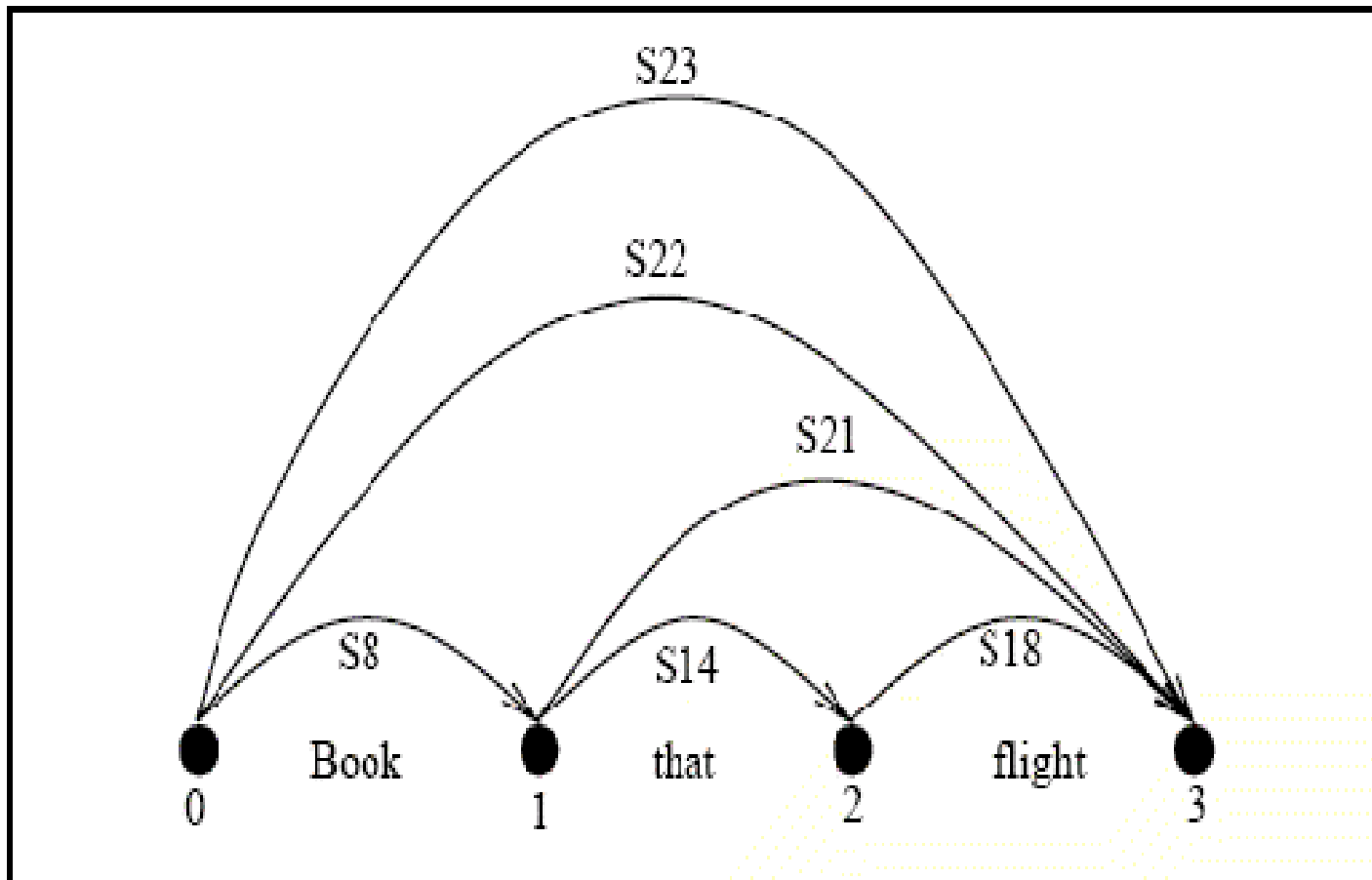
### Basic Idea:

**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** 可保持森林: 歧义



### 3.4. A Bottom-Up Chart Parser



## 3.4. A Bottom-Up Chart Parser



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

*The large can can hold the water.*

## 3.4. A Bottom-Up Chart Parser

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. This is denoted by writing the rule with a **dot (o)**, indicating what has been seen so far:

**2'. NP -> ART o ADJ N**

**3'. NP -> ART o N**

**Note: also called “left-corner parsing algorithm”**

If the next input key is an ADJ, then rule 4 may be started:

**4'. NP -> ADJ o N**

and the modified rule 2 may be extended to give

**2''. NP -> ART ADJ o N**

### 3.4. A Bottom-Up Chart Parser

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

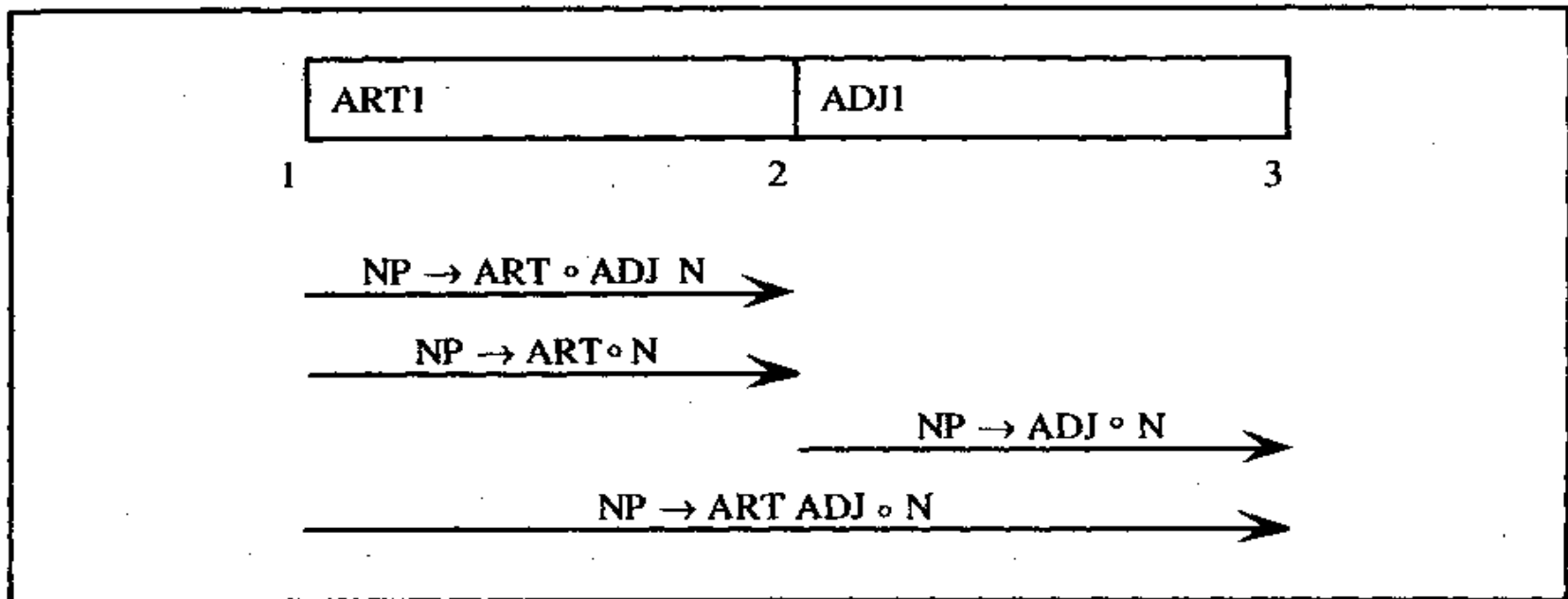


Figure 3.9 The chart after seeing an ADJ in position 2

## 3.4. A Bottom-Up Chart Parser



The basic operation of a chart-based parser involves combining an **active arc** with a **completed constituent**. The result is **either a new completed constituent or a new active arc that is an extension of the original active arc**. **New completed constituents** are maintained on a list called the **agenda** until they themselves are added to the chart.

**Data structure:**

**chart:**

Completed constituents that enter the chart + active arcs

**agenda:**

New completed constituents

## 3.4. A Bottom-Up Chart Parser



### The bottom-up chart parsing algorithm:

Do until there is no input left:

1. If the agenda is empty, look up the interpretations for the next word in the input and add them to the **agenda**.
2. Select a constituent from the agenda (let's call it constituent  $C$  from position  $p1$  to  $p2$ ).
3. For each rule in the grammar of form  $\mathbf{X} \rightarrow \mathbf{C} \mathbf{x1} \dots \mathbf{x_n}$ , add an **active arc** of form  $\mathbf{X} \rightarrow \mathbf{C} \circ \mathbf{x1} \dots \mathbf{x_n}$  from position  $p1$  to  $p2$  into the **chart**.
4. Add  $C$  to the **chart** using the arc extension algorithm.

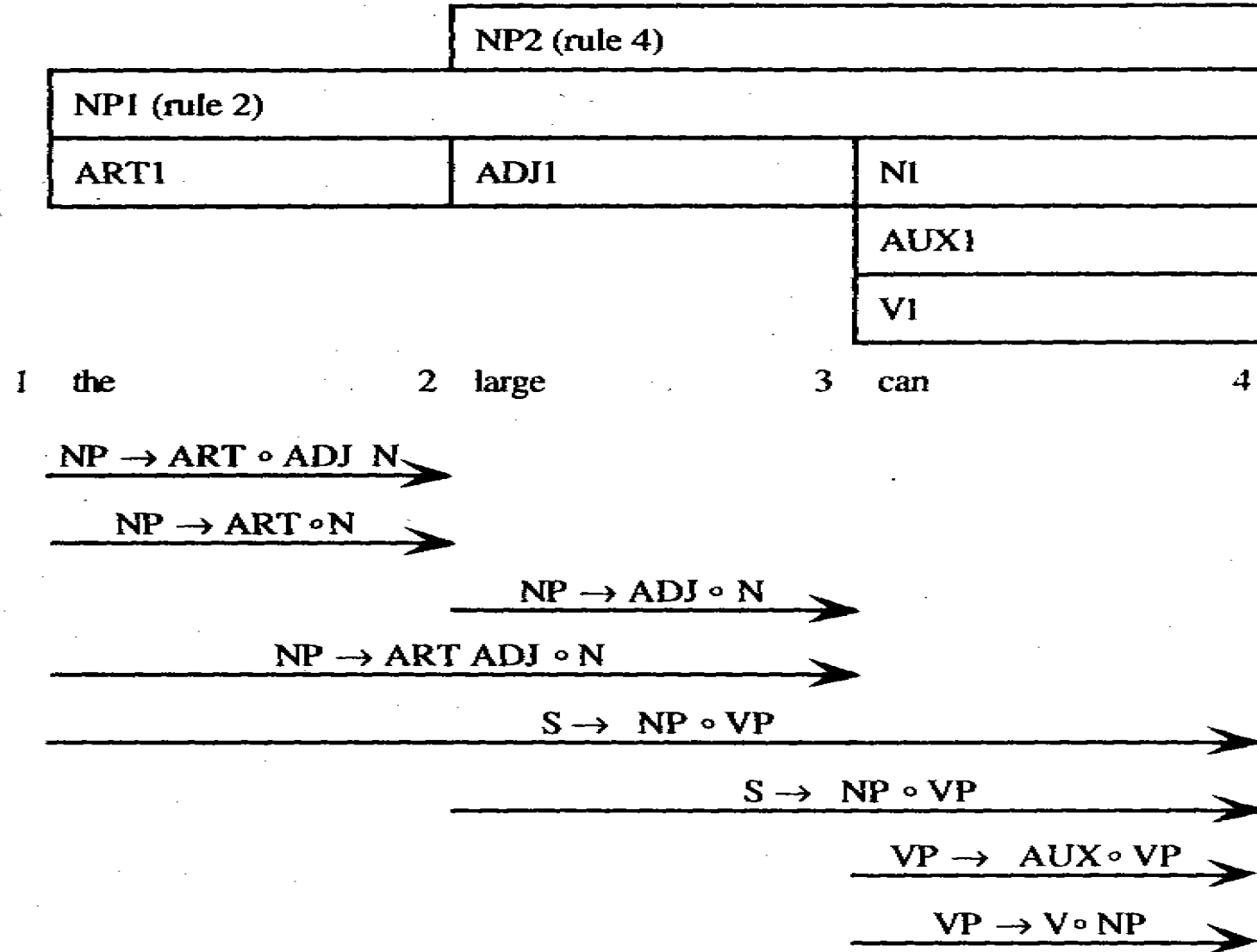
## 3.4. A Bottom-Up Chart Parser

To add a constituent  $C$  from position  $p_1$  to  $p_2$ :

1. Insert  $C$  into the chart from position  $p_1$  to  $p_2$ .
2. For any active arc of the form  $X \rightarrow X_1 \dots \circ C \dots X_n$  from position  $p_0$  to  $p_1$ , add a new active arc  $X \rightarrow X_1 \dots C \circ \dots X_n$  from position  $p_0$  to  $p_2$ .
3. For any active arc of the form  $X \rightarrow X_1 \dots X_n \circ C$  from position  $p_0$  to  $p_1$ , then add a new constituent of type  $X$  from  $p_0$  to  $p_2$  to the agenda.

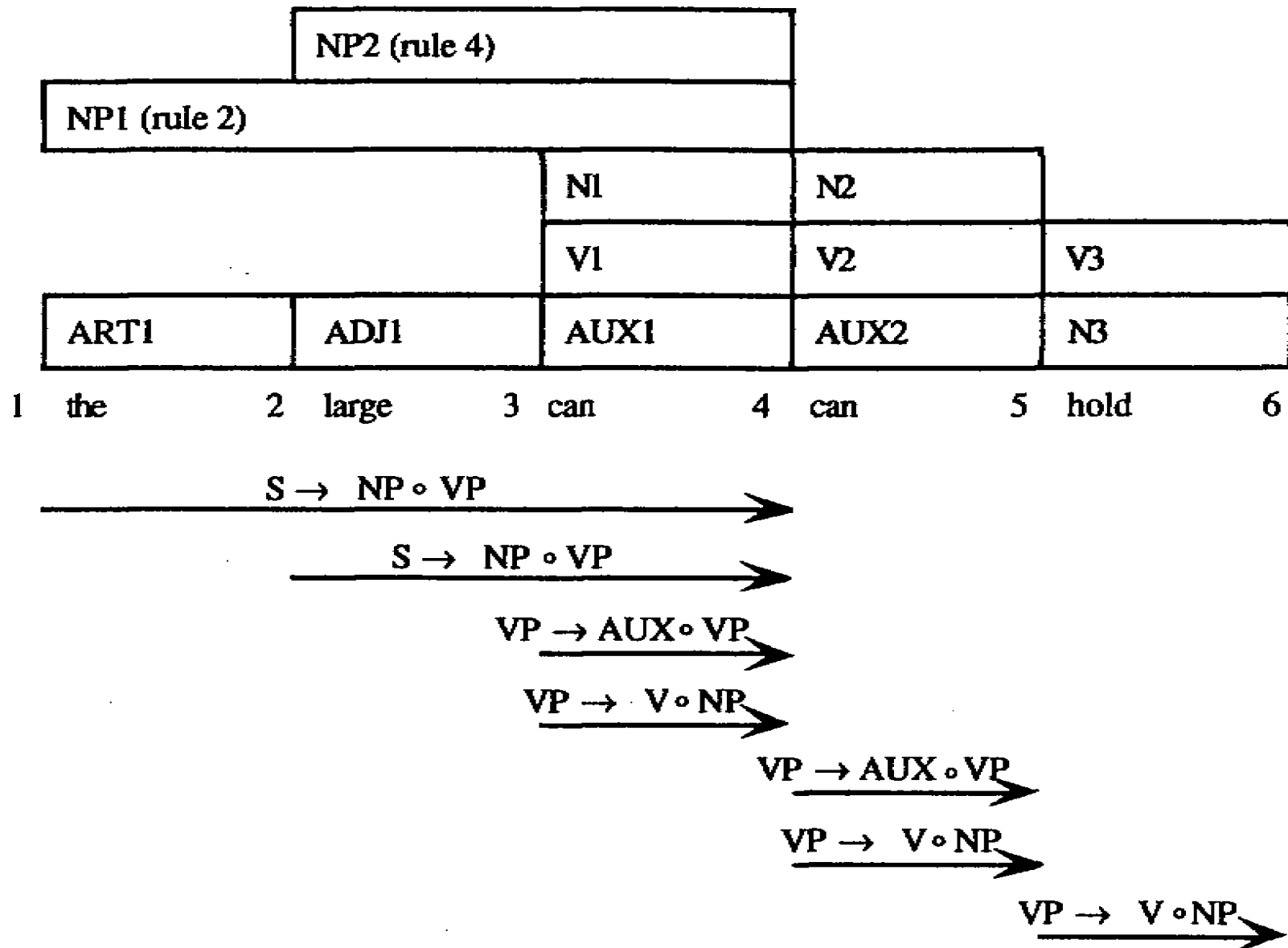
Figure 3.10 The arc extension algorithm

# *The large can can hold the water*

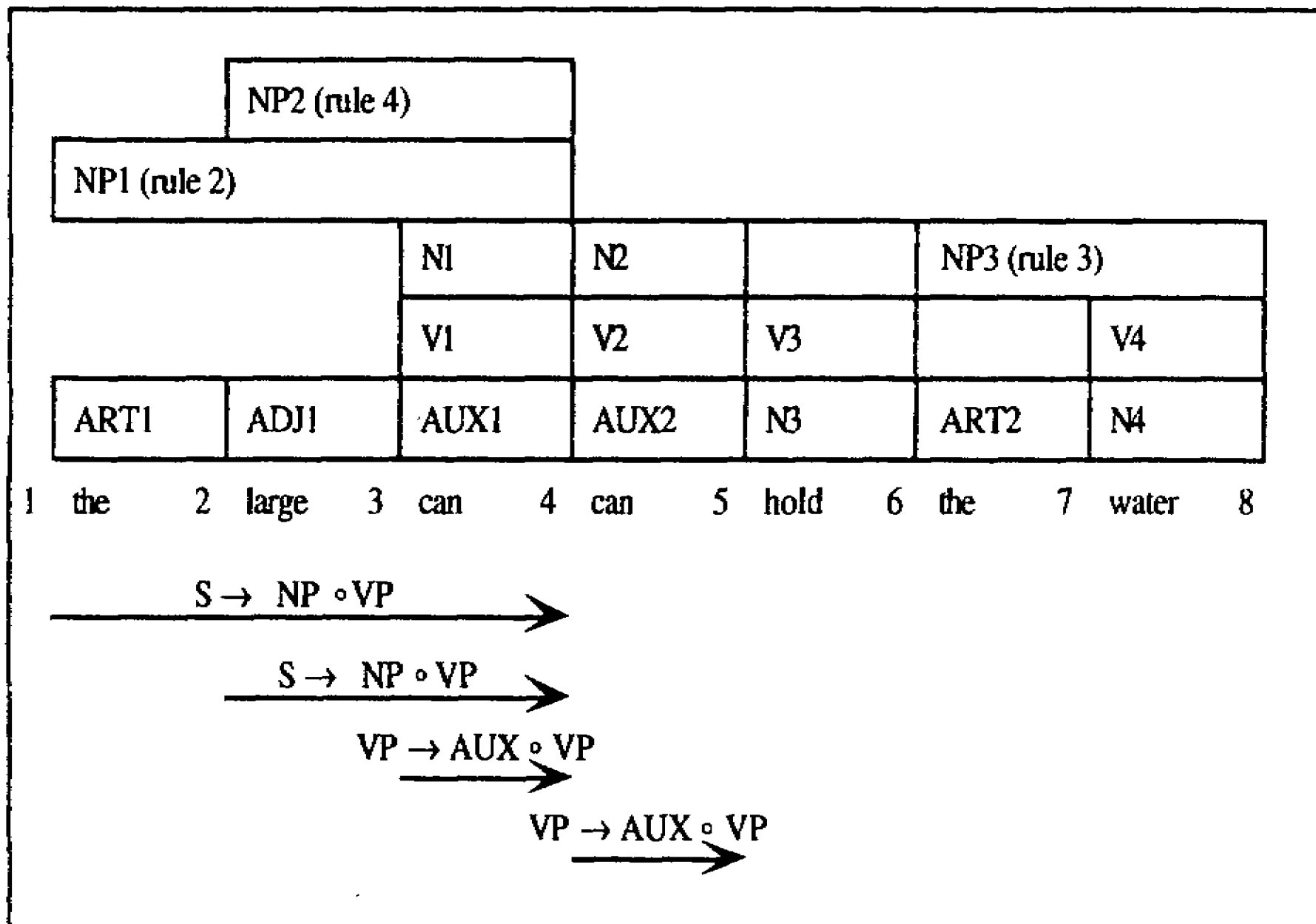


**Figure 3.12** After parsing *the large can*

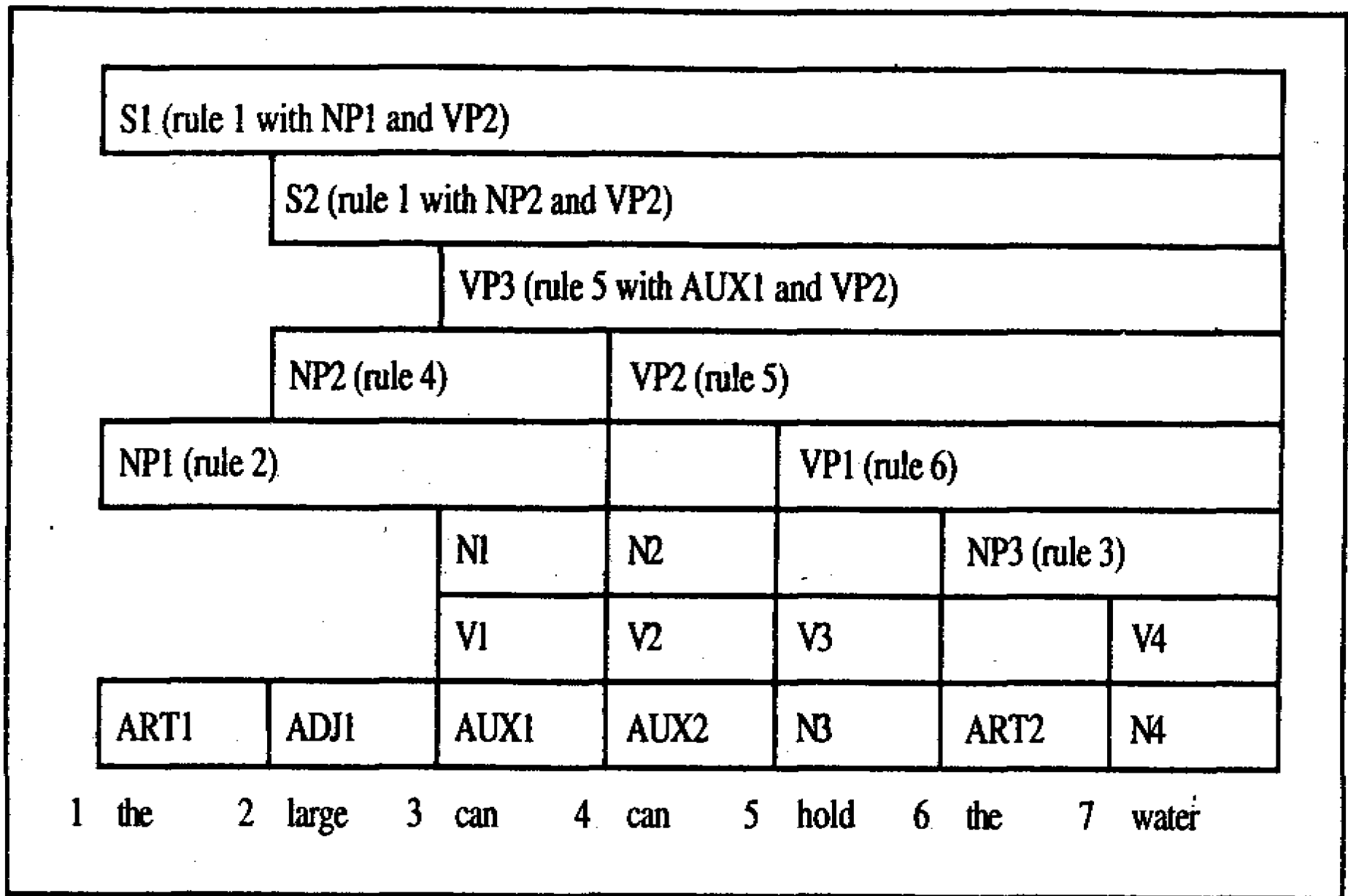




**Figure 3.13** The chart after adding *hold*, omitting arcs generated for the first NP



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



**Figure 3.15** The final chart

## 3.4. A Bottom-Up Chart Parser



A pure top-down or bottom-up search strategy:  $C^n$  operations to parse a sentence of length  $n$ , where  $C$  is a constant that depends on the specific algorithm used.

A chart-based parser: a worst-case complexity of  $K * n^3$ , where  $n$  is the length of the sentence and  $K$  is a constant depending on the algorithm.

A chart parser involves more work in each step, so  $K$  will be larger than  $C$ .

To contrast the two approaches, assume that  $C$  is 10 and that  $K$  is a hundred times worse, 1000. Given a sentence of 12 words, the brute force search might take  $10^{12}$  operations (that is, 1,000,000,000,000), whereas the chart parser would take  $1000 * 12^3$  (that is, 1,728,000). Under these assumptions, the chart parser would be up to 500,000 times faster than the brute force search on some examples!

## 3.5. Top-Down Chart Parsing (Earley Algorithm)

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

范畴: 左角

S: Det, Proper-Noun, Aux, Verb

NP: Det, Proper-Noun

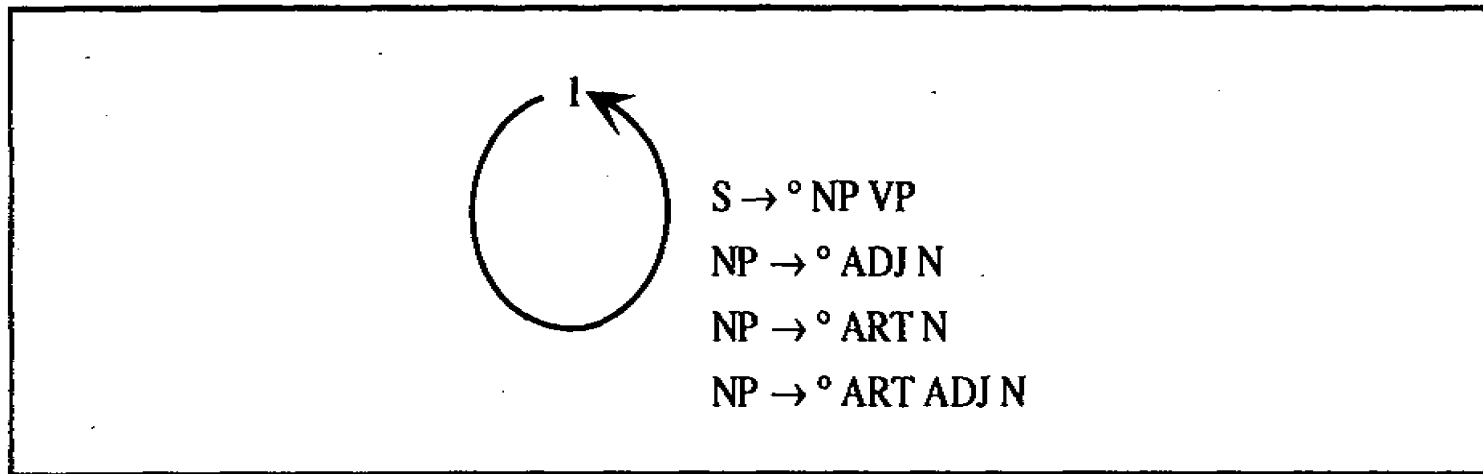
Nominal: Noun

VP: Verb

## 3.5. Top-Down Chart Parsing (Earley Algorithm)

- |                               |              |
|-------------------------------|--------------|
| 1. $S \rightarrow NP VP$      | the: ART     |
| 2. $NP \rightarrow ART ADJ N$ | large: ADJ   |
| 3. $NP \rightarrow ART N$     | can: N,AUX,V |
| 4. $NP \rightarrow ADJ N$     | hold: N,V    |
| 5. $VP \rightarrow AUX VP$    | water: N, V  |
| 6. $VP \rightarrow V NP$      |              |

*The large can can hold the water.*



**Figure 3.23** The initial chart

## 3.5. Top-Down Chart Parsing (Earley Algorithm)

### Top-Down Arc Introduction Algorithm

To add an arc  $S \rightarrow C_1 \dots o C_i \dots C_n$  ending at position  $j$ , do the following:

For each rule in the grammar of form  $C_i \rightarrow X_1 \dots X_k$ , recursively add the new arc  $C_i \rightarrow o X_1 \dots X_k$  from position  $j$  to  $j$ .

### Top-Down Chart Parsing Algorithm

Initialization: For every rule in the grammar of form  $S \rightarrow X_1 \dots X_k$ , add an arc labeled  $S \rightarrow o X_1 \dots X_k$  using the top-down 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  $C$ ).

(NOT: 3. For each rule in the grammar of form  $X \rightarrow C X_1 \dots X_n$ , add an active arc of form  $X \rightarrow C o X_1 \dots X_n$  from position  $p_1$  to  $p_2$  into the agenda. )

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.

## 3.5. Top-Down Chart Parsing (Earley Algorithm)

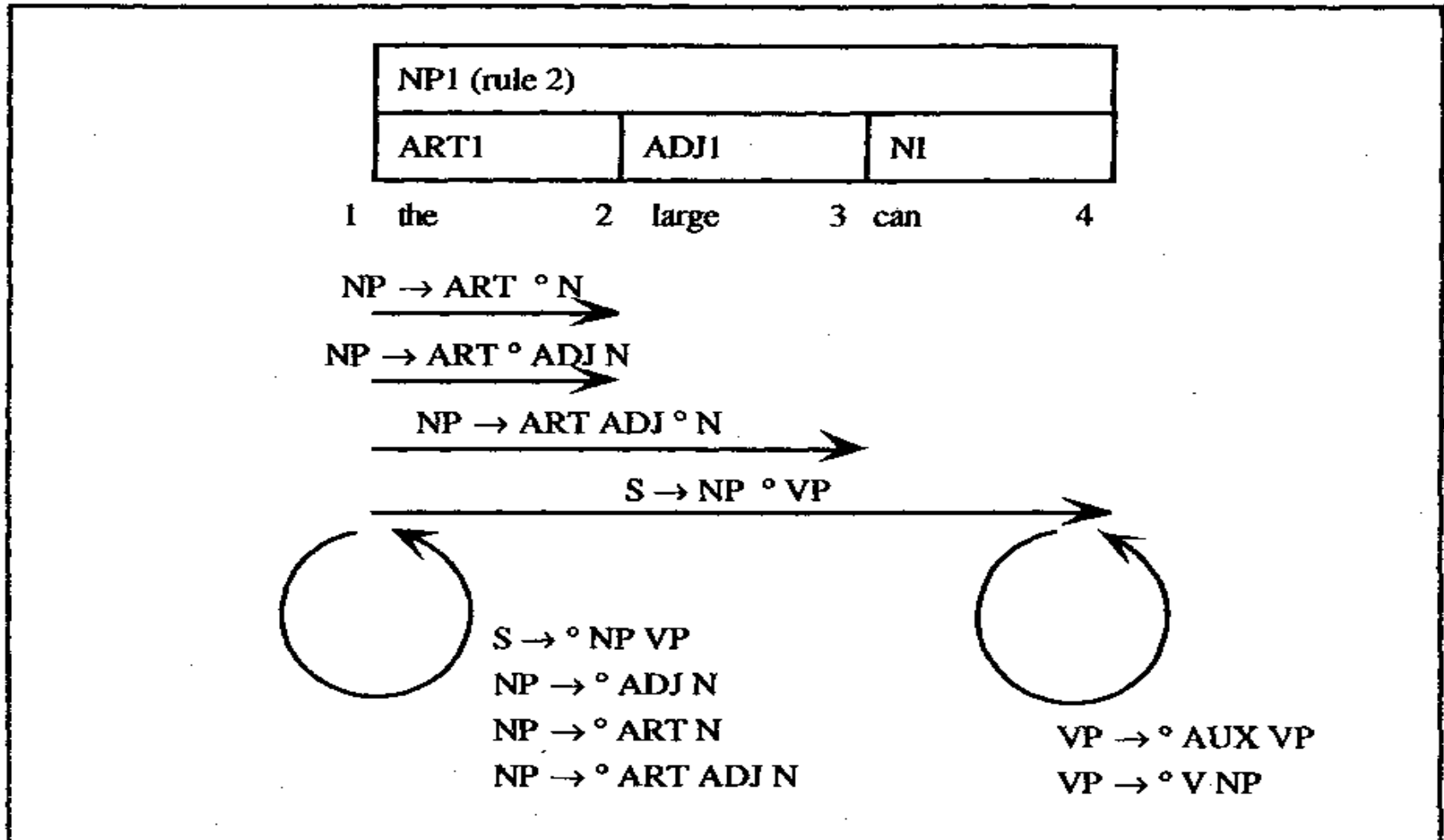


Figure 3.24 The chart after building the first NP



NP1 (rule 2)			V2	
ART1	ADJ1	N1	AUX2	V3

the                      2   large                      3   can                      4   can                      5   hold                      6

$S \rightarrow NP^\circ VP$

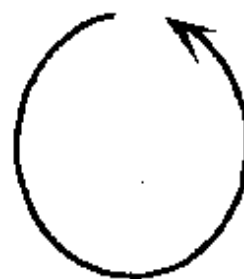
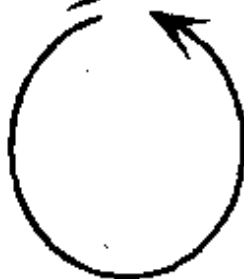
$VP \rightarrow AUX^\circ VP$

$VP \rightarrow V^\circ NP$

$VP \rightarrow V^\circ NP$

$VP \rightarrow^\circ AUX VP$

$VP \rightarrow^\circ V NP$



$VP \rightarrow^\circ AUX VP$

$VP \rightarrow^\circ V NP$

$NP \rightarrow^\circ ART ADJ N$

$NP \rightarrow^\circ ART N$

$NP \rightarrow^\circ ADJ N$

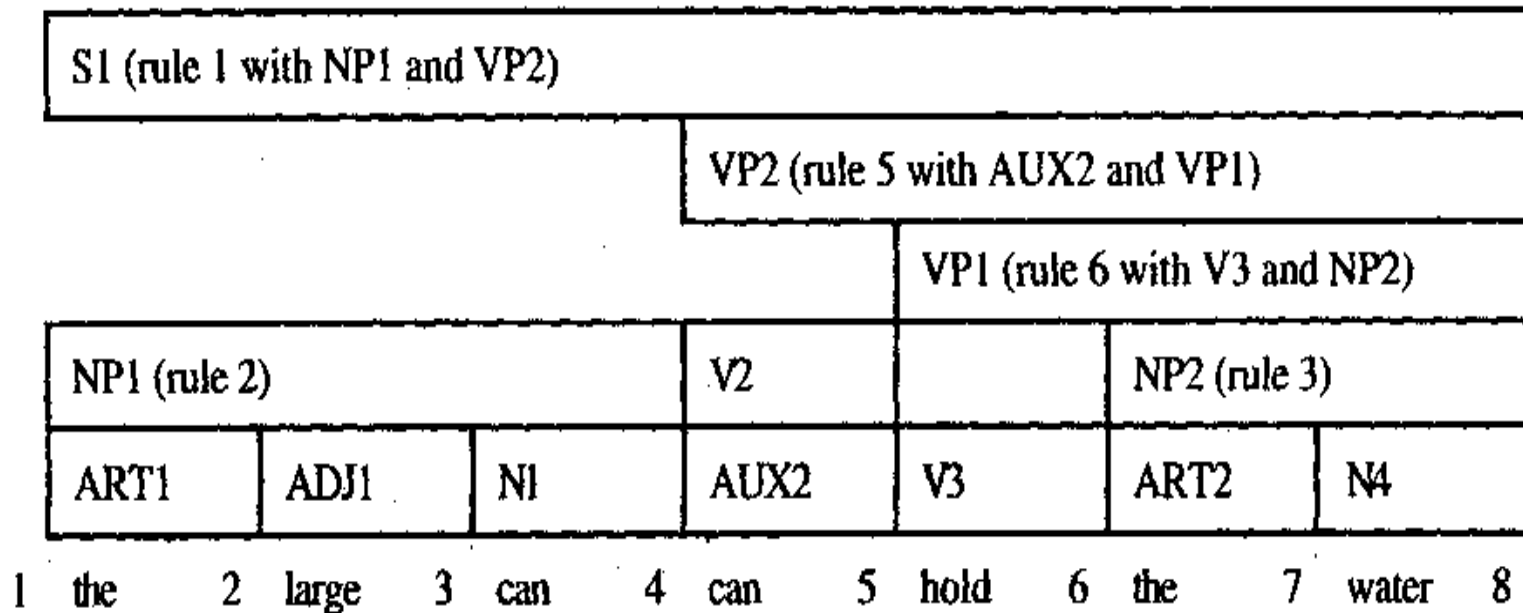
$NP \rightarrow^\circ ART ADJ N$

$NP \rightarrow^\circ ART N$

$NP \rightarrow^\circ ADJ N$

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

## 3.5. Top-Down Chart Parsing (Earley Algorithm)



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

worst-case complexity of  $K \cdot n^3$ , in practice, more efficient than a pure chart-based parser.

S1 (rule 1 with NP1 and VP2)													
			S2 (rule 1 with NP2 and VP2)										
		VP3 (rule 5 with AUX1 and VP2)											
NP2 (rule 4)			VP2 (rule 5)										
NP1 (rule 2)				VP1 (rule 6)									
		N1	N2		NP3 (rule 3)								
		V1	V2	V3		V4							
ART1	ADJ1	AUX1	AUX2	N3	ART2	N4							
1	the	2	large	3	can	4	can	5	hold	6	the	7	water

**Figure 3.15** The final chart

S1 (rule 1 with NP1 and VP2)														
				VP2 (rule 5 with AUX2 and VP1)										
				VP1 (rule 6 with V3 and NP2)										
NP1 (rule 2)			V2		NP2 (rule 3)									
ART1	ADJ1	N1	AUX2	V3	ART2	N4								
1	the	2	large	3	can	4	can	5	hold	6	the	7	water	8

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