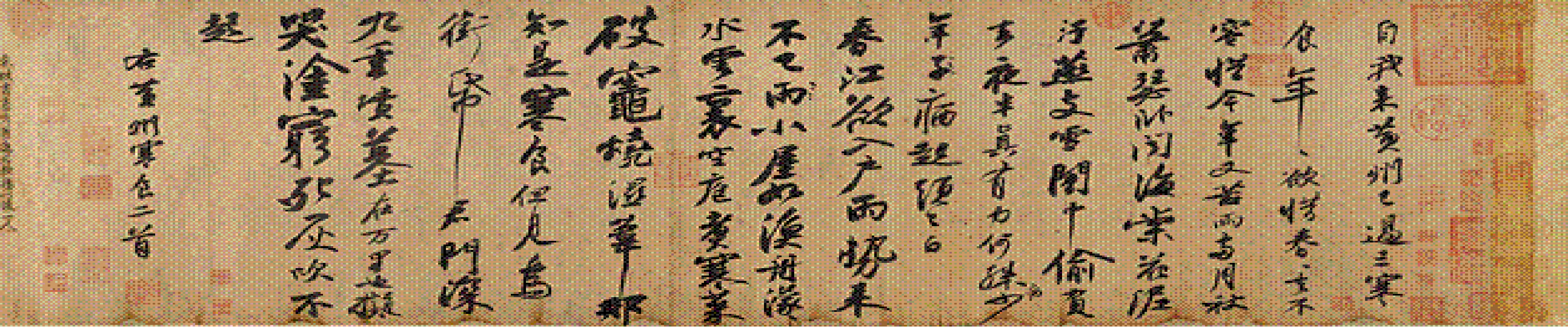


计算语言学
Computational Linguistics

教师：孙茂松

Tel:62781286

Email:sms@tsinghua.edu.cn

TA：林衍凯

Email: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 2)

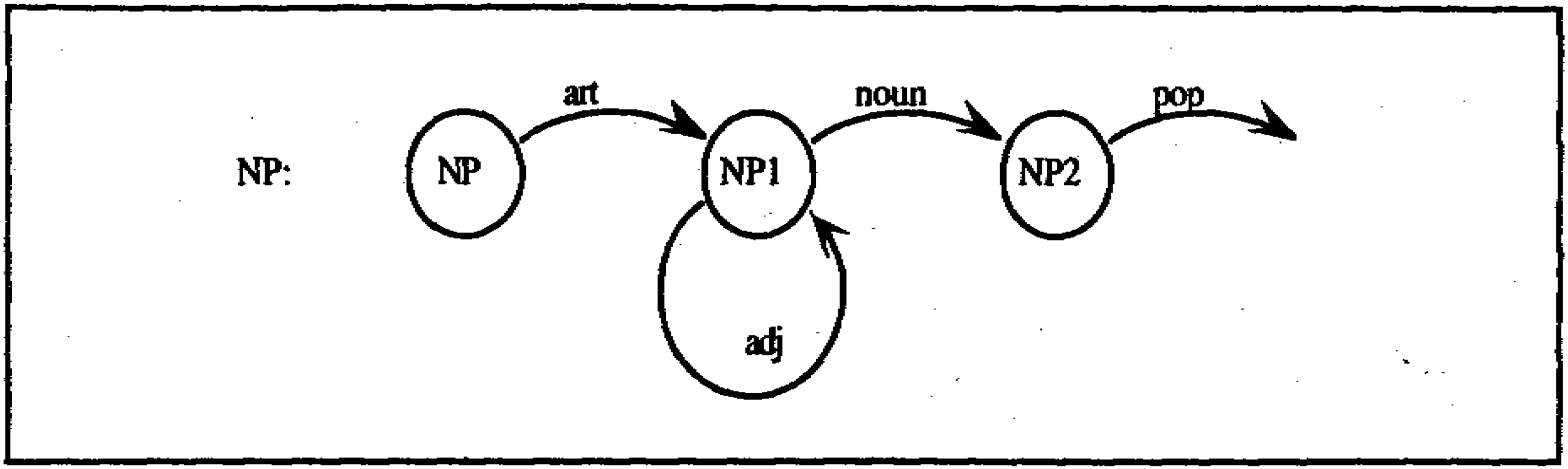
3.6. Transition Network Grammars

```
NP  -> ART NP1
NP1 -> ADJ NP1
NP1 -> N
```

transition network: consisting of nodes and labeled arcs

initial state, or start state

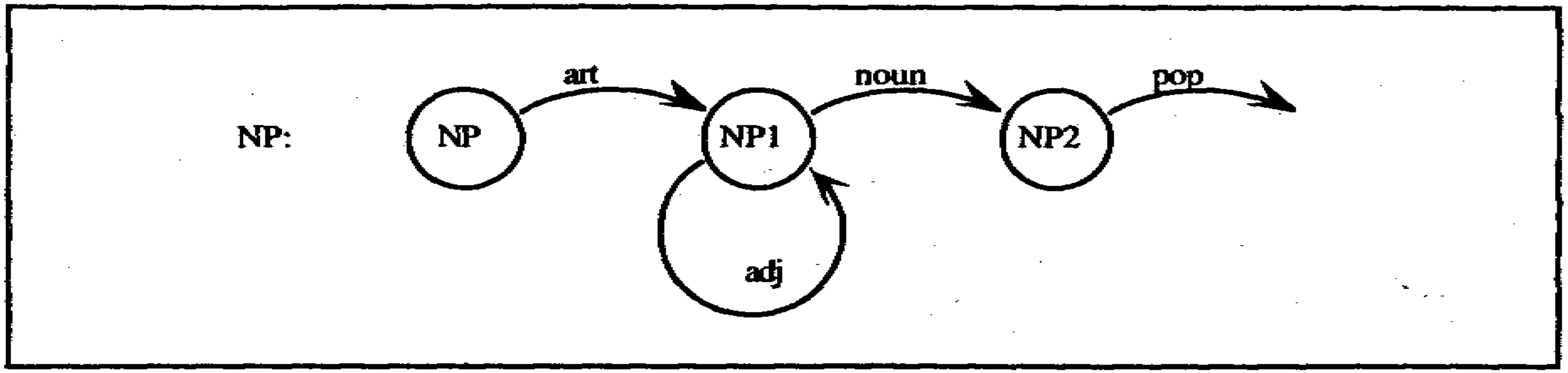
pop arc



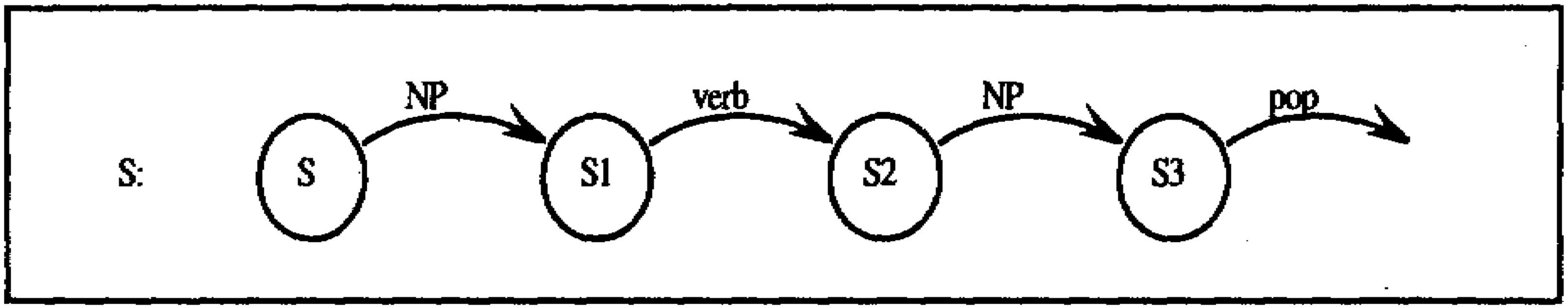
Grammar 3.16

3.6. Transition Network Grammars

recursive transition network (RTN)



Grammar 3.16



Grammar 3.17

3.6. Transition Network Grammars

Arc Type	Example	Arc Type Example How Used
CAT	noun	succeeds only if current word is of the named category
WRD	of	succeeds only if current word is identical to the label
PUSH	NP	succeeds only if named network can be successfully traversed
JUMP	jump	always succeeds
POP	pop	succeeds and signals the successful end of the network

The arc labels for RTNs

3.6. Transition Network Grammars

Top-Down Parsing with Recursive Transition Networks
An algorithm for parsing with RTNs can be developed along the same lines as the algorithms for parsing CFGs. The state of the parse at any moment can be represented by the following:

current position - a pointer to the next word to be parsed.
current node - the node at which you are located in the network.
return points - a stack of nodes in other networks where you will continue if you pop from the current network.

3.6. Transition Network Grammars

Case 1: If arc names word category and next word in sentence is in that category,
Then (1) update *current position* to start at the next word;
(2) update *current node* to the destination of the arc.

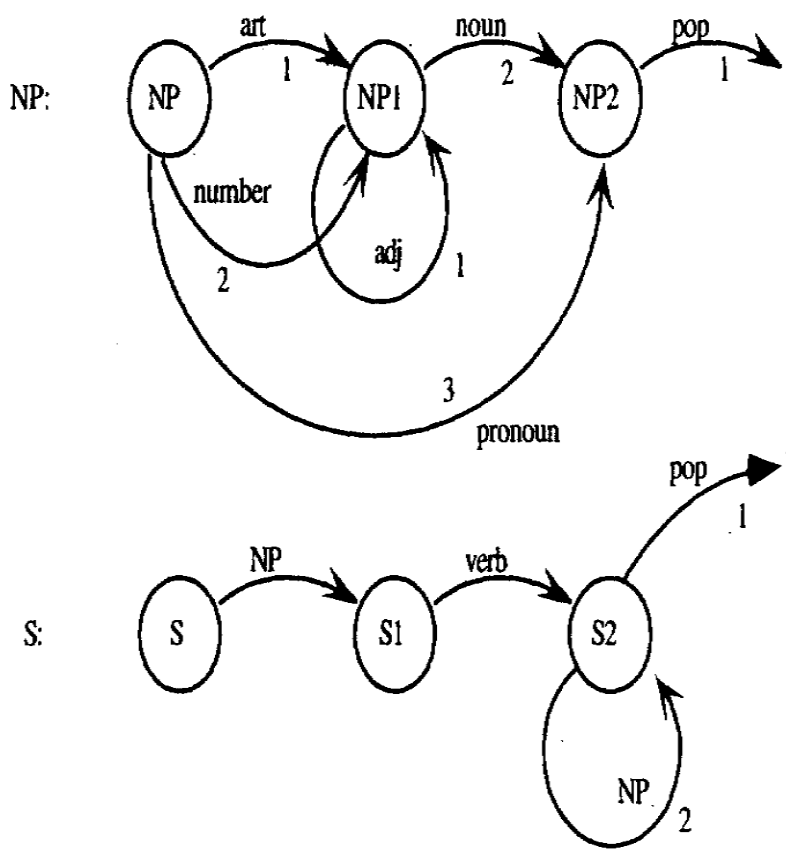
Case 2: If arc is a push arc to a network X,
Then (1) add the destination of the arc onto *return points*;
(2) update *current node* to the starting node in network X.

Case 3: If arc is a pop arc and *return points* list is not empty,
Then (1) remove first *return point* and make it *current node*.

Case 4: If arc is a pop arc, *return points* list is empty and there are no words left,
Then (1) parse completes successfully.

3.6. Transition Network Grammars

1 The 2 old 3 man 4 cried 5 .



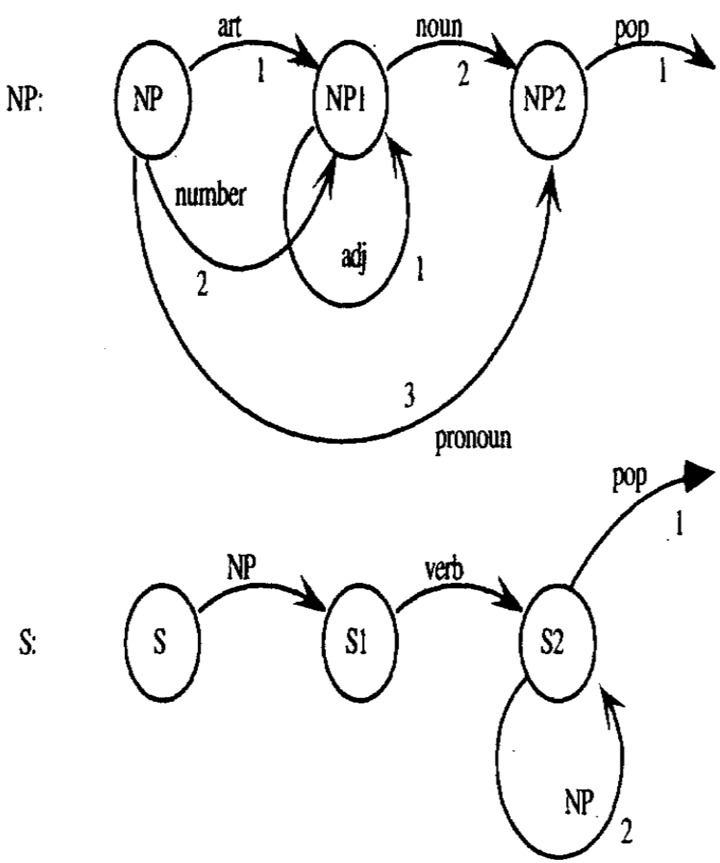
Grammar 3.19

Step	Current Node	Current Position	Return Points	Arc to be Followed	Comments
1.	(S,	1,	NIL)	S/1	initial position
2.	(NP,	1,	(S1))	NP/1	followed push arc to NP network, to return ultimately to S1
3.	(NP1,	2,	(S1))	NP1/1	followed arc NP/1 (*the*)
4.	(NP1,	3,	(S1))	NP1/2	followed arc NP1/1 (*old*)
5.	(NP2,	4,	(S1))	NP2/2	followed arc NP1/2 (*man*) since NP1/1 is not applicable
6.	(S1,	4,	NIL)	S1/1	the pop arc gets us back to S1
7.	(S2,	5,	NIL)	S2/1	followed arc S2/1 (*cried*)
8.					parse succeeds on pop arc from S2

Figure 3.20 A trace of a top-down parse

3.6. Transition Network Grammars

1 One 2 saw 3 the 4 man 5 .



Step	Current State	Arc to be Followed	Backup States
1.	(S, 1, NIL)	S/1	NIL
2.	(NP, 1, (S1))	NP/2 (& NP/3 for backup)	NIL
3.	(NP1, 2, (S1))	NP1/2	(NP2, 2, (S1))
4.	(NP2, 3, (S1))	NP2/1	(NP2, 2, (S1))
5.	(S1, 3, NIL)	no arc can be followed	(NP2, 2, (S1))
6.	(NP2, 2, (S1))	NP2/1	NIL
7.	(S1, 2, NIL)	S1/1	NIL
8.	(S2, 3, NIL)	S2/2	NIL
9.	(NP, 3, (S2))	NP/1	NIL
10.	(NP1, 4, (S2))	NP1/2	NIL
11.	(NP2, 5, (S2))	NP2/1	NIL
12.	(S2, 5, NIL)	S2/1	NIL
13.	parse succeeds		NIL

3.6. Transition Network Grammars

An RTN parser can be constructed to use a chart-like structure to gain the advantages of chart parsing. In RTN systems, the chart is often called the **well-formed substring table (WFST).** Each time a pop is followed, the constituent is placed on the WFST, and every time a push is found, the WFST is checked before the subnetwork is invoked. If the chart contains constituent(s) of the type being pushed for, these are used and the subnetwork is not reinvoked. **An RTN using a WFST** has the same complexity as the chart parser described in the last section: $K*n^3$, where n is the length of the sentence.

3.7. Finite State Models and Morphological Processing

finite state transducers (FSTs)

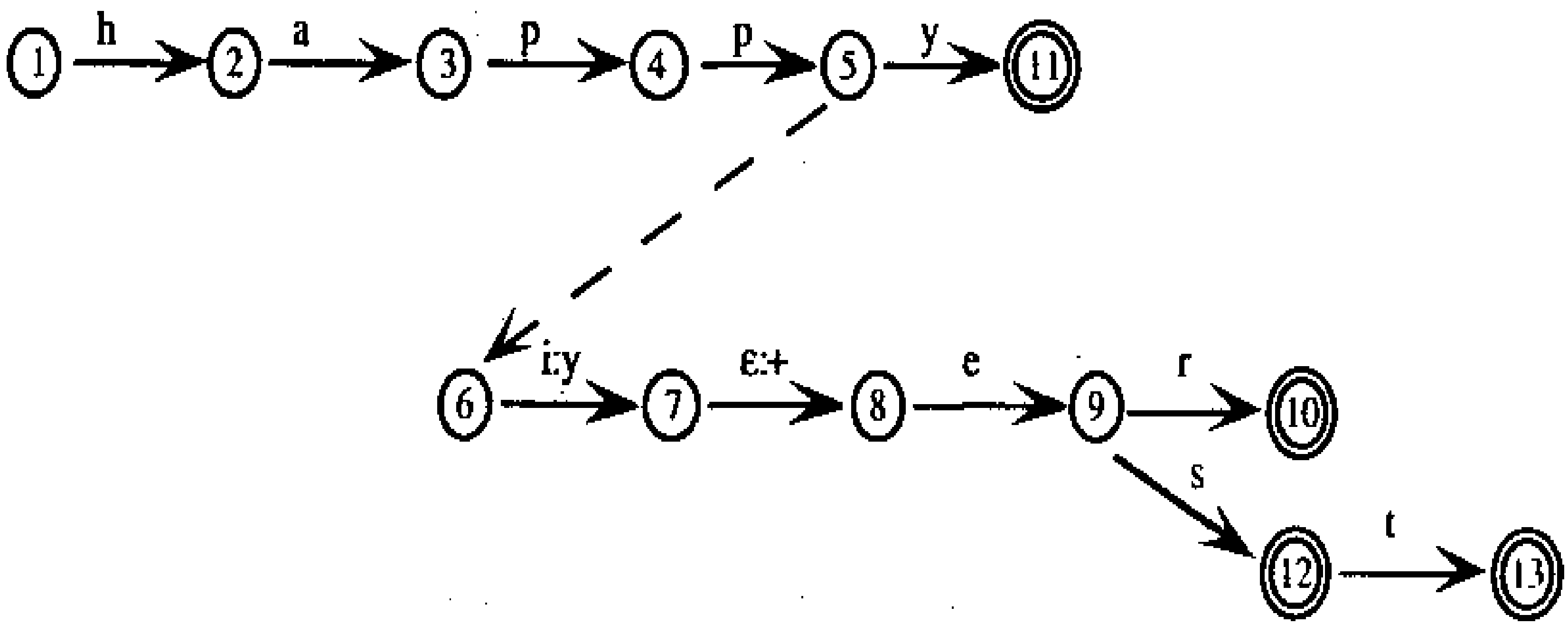


Figure 3.27 A simple FST for the forms of *happy*

3.7. Finite State Models and Morphological Processing

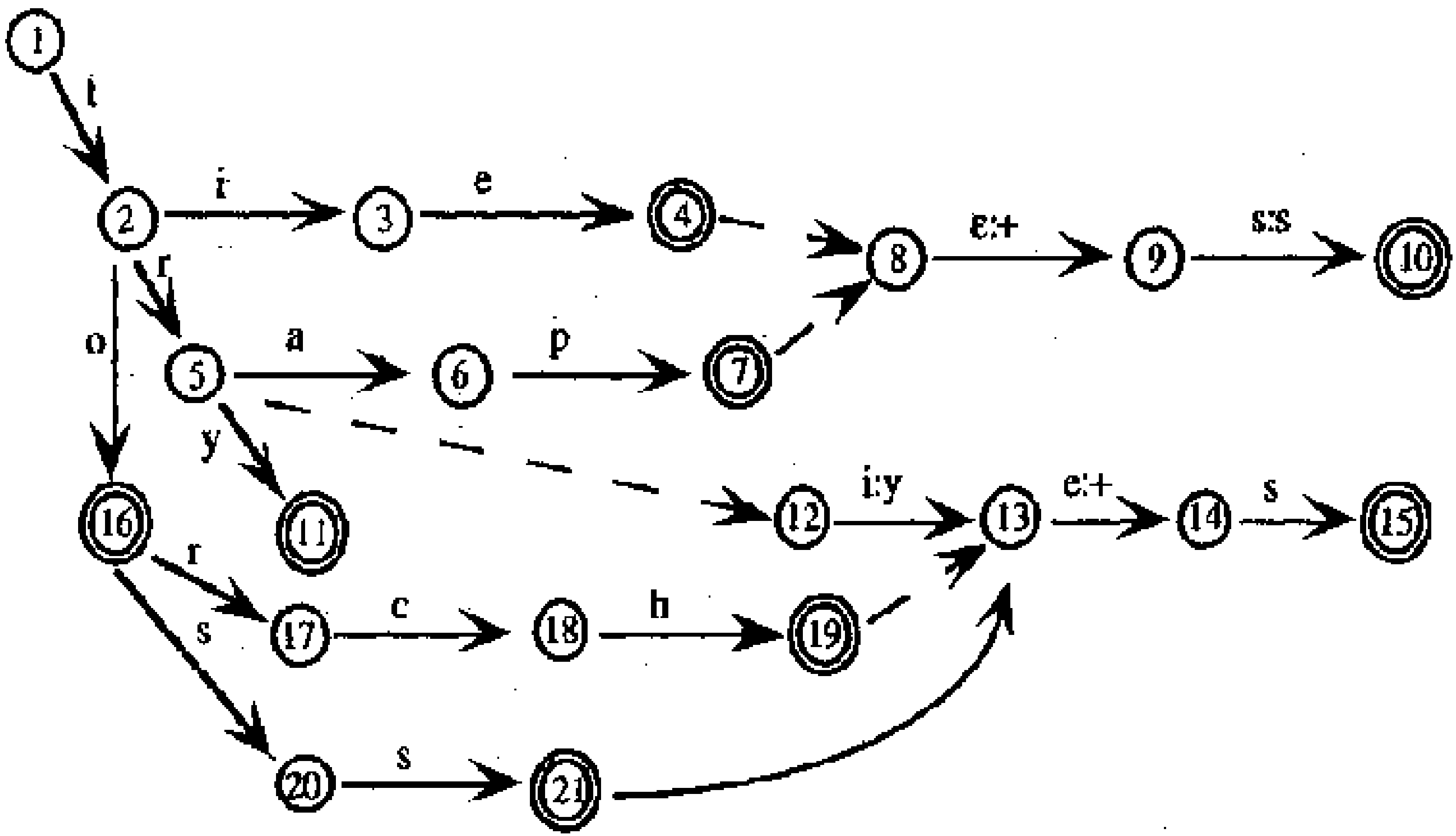


Figure 3.28 A fragment of an FST defining some nouns (singular and plural)

3.8. CYK Algorithm

The **Cocke-Younger-Kasami (CYK) algorithm**

- One of the earliest recognition and parsing algorithms
- Assumes the grammar is in CNF (and depends on this!)
- Based on a "dynamic programming" approach:
  - Build solutions compositionally from sub-solutions
  - Store sub-solutions and re-use them whenever necessary

- Input is $w = x_1 x_2 \ldots x_n$
- We denote $w_{ij} = x_i x_{i+1} \ldots x_{i+j-1}$, the substring of w of length j, starting with x_i
- For every w_{ij} and for every variable A in the grammar, the algorithm determines if $A \stackrel{*}{\Rightarrow} w_{ij}$

3.8. CYK Algorithm

The CYK Parsing Algorithm

- The algorithm works on substrings of increasing length:
- We start with substrings of length 1: $w_{i1} = x_i$, for $1 \leq i \leq n$
  - $A \stackrel{*}{\Rightarrow} w_{i1}$ if $A \rightarrow x_i$ is a rule in the grammar
- We then continue with substrings of length 2,3,...
- For a substring w_{ij}, we consider all possible ways of breaking it into two parts w_{ik} and $w_{i+k\ j-k}$
- $A \stackrel{*}{\Rightarrow} w_{ij}$ if:
  1. $A \rightarrow B\ C$ is a rule in the grammar
  2. $B \stackrel{*}{\Rightarrow} w_{ik}$
  3. $C \stackrel{*}{\Rightarrow} w_{i+k\ j-k}$
- Finally, since $w - w_{1n}$, we need to verify that $S \stackrel{*}{\Rightarrow} w_{1n}$

CYK table

S						
	VP					
S						
	VP			PP		
S		NP			NP	
Pron, NP	V, VP	Det.	N, NP	P	Det	N, NP
she	eats	a	fish	with	a	fork

3.8. CYK Algorithm

- We keep the results for every w_{ij} in a table V_{ij}
- Each table entry V_{ij} contains the set of variables A such that $A \stackrel{*}{\Rightarrow} w_{ij}$
- Note that we only need to fill in entries up to the diagonal - the longest substring starting at i is of length $n - i + 1$

---

begin

1) **for** $i := 1$ **to** n **do**

2) $V_{i1} := \{A \mid A \rightarrow a$ is a production and the ith symbol of x is $a\}$;

3) **for** $j := 2$ **to** n **do**

4) **for** $i := 1$ to $n - j + 1$ **do**

begin

5) $V_{ij} := \varnothing$;

6) **for** $k := 1$ **to** $j - 1$ **do**

7) $V_{ij} := V_{ij} \cup \{A \mid A \rightarrow BC$ is a production, B is in V_{ik} and C is in $V_{i+k,j-k}\}$

end

end

3.8. CYK Algorithm

Time Complexity of CYK

- We have three nested loops, each with ranges of at most 1 to n
- The internal individual steps (2) (5) (7) each require constant time, since the grammar is fixed, and thus the size of V_{ij} is a constant
- Step (7) is the most nested and thus gets executed $O(n^3)$ times
- Thus, the entire algorithm requires $O(n^3)$ time

Membership problem for CFGs

backpointers.