Natural Language Processing

Regular Expressions

Definitions

- A language is a set of strings
- String: A sequence of letters
  - Examples: "cat", "dog", "house", ...
  - Defined over an alphabet:

$$\Sigma = \{a, b, c, \ldots, z\}$$

Alphabets and Strings

- We will use small alphabets: $\Sigma = \{a, b\}$
- Strings

$$a$$

$$ab \qquad u = ab$$

$$abba \qquad v = bbbaaa$$

$$baba \qquad w = abba$$

$$aaabbbaabab$$

Regular Expressions

- In computer science, RE is a language used for specifying text search string.
- A *regular expression* is a *formula* in a *special language* that is used for specifying a simple class of *string*.
- Formally, a regular expression is an algebraic notation for characterizing a set of strings.

Basic Regular Expression Patterns

- The use of the brackets [] to specify a disjunction of characters.

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', *or* 'c'	"In uomini, in soldati"
/[1234567890]/	any digit	"plenty of 7 to 5"

- The use of the brackets [] plus the dash - to specify a range.

RE	Match	Example Patterns Matched
/[A-Z]/	an uppercase letter	"we should call it 'Drenched Blossoms'"
/[a-z]/	a lowercase letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

Basic Regular Expression Patterns

- Uses of the caret ^ for negation or just to mean ^

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase letter	"Oyfn pripetchik"
[^Ss]	neither 'S' nor 's'	"I have no exquisite reason for't"
[^\.]	not a period	"our resident Djinn"
[e^]	either 'e' or '^'	"look up ^ now"
a^b	the pattern 'a^b'	"look up a^b now"

- The question-mark ? marks optionality of the previous expression.

RE	Match	Example Patterns Matched
woodchucks?	woodchuck or woodchucks	"woodchuck"
colou?r	color or colour	"colour"

- The use of period . to specify any character

RE	Match	Example Patterns
/beg.n/	any character between *beg* and *n*	begin, beg'n, begun

Aliases for common sets of characters

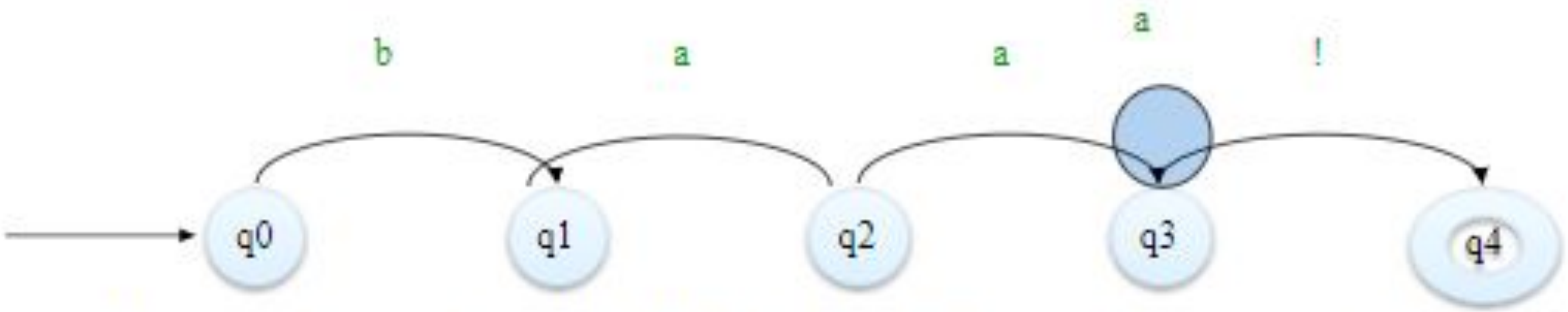
RE	Expansion	Match	Example Patterns
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9␣]	any alphanumeric or space	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Some characters that need to be backslashed

RE	Match	Example Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Would you light my candle?"
\n	a newline	
\t	a tab	

Finite State Automata

- FSA s recognize the regular languages represented by regular expressions
  - SheepTalk: /baa+!/

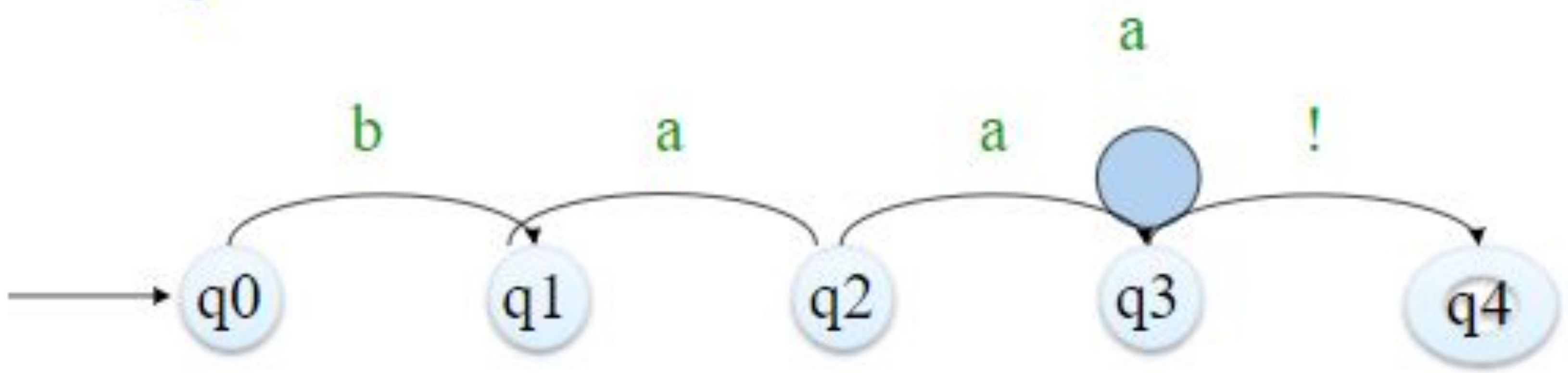


- **Directed graph with labeled nodes and arc transitions**

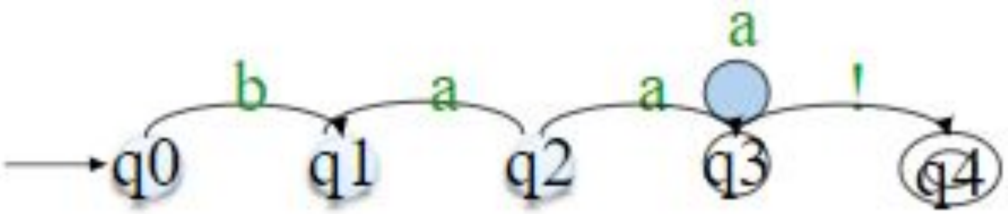
•Five **states:** q0 the **start** state, q4 the **final** state, 5 transitions

Formally

- FSA is a 5-tuple consisting of
  - Q: set of states {q0,q1,q2,q3,q4}
  - Σ: an alphabet of symbols {a,b,!}
  - q0 : A start state
  - F: a set of final states in Q {q4}
  - δ(q,i): a transition function mapping Q x Σ to Q



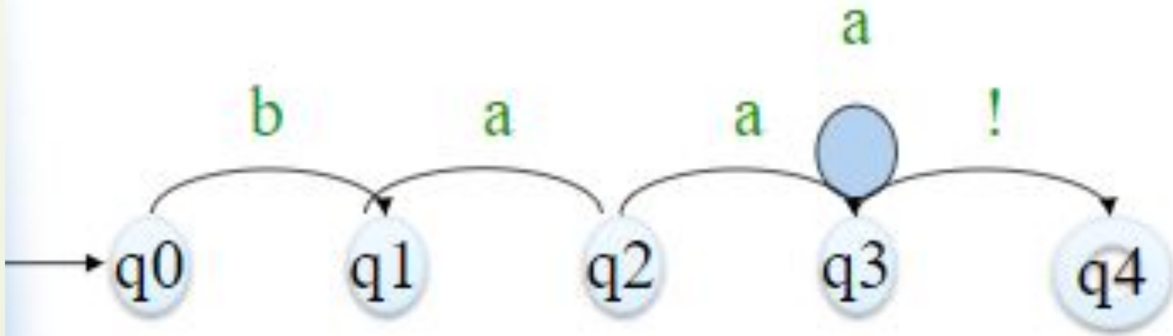
- FSA recognizes (**accepts**) strings of a regular language
  - baa!
  - baaa!
  - baaaa!
  - ...



- Tape Input: a rejected input

	a	b	a	!	b	

State Transition Table for SheepTalk



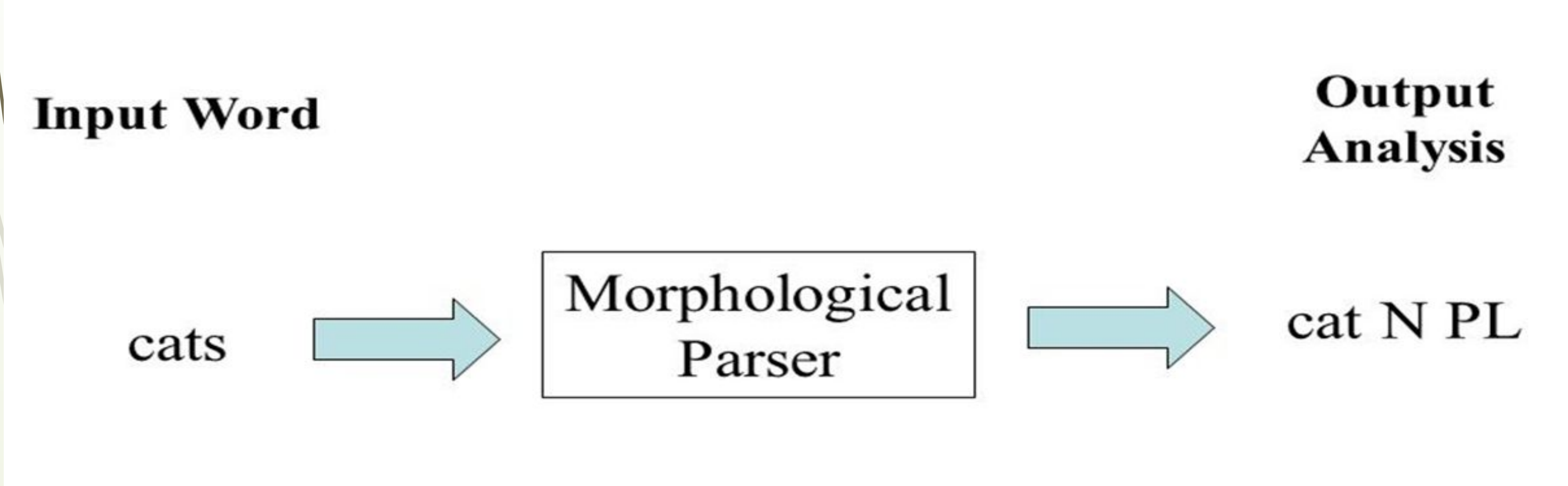
State	Input		
	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4	Ø	Ø	Ø

Morphology

- Morpheme = "minimal meaning-bearing unit in a language"
- Morphology handles the formation of words by using morphemes
  - base form (stem,lemma), e.g., *believe*
  - affixes (suffixes, prefixes, infixes), e.g., *un-, -able, -ly*
- Morphological parsing = the task of recognizing the morphemes inside a word
  - e.g., *hands, foxes, children*

Morphological Parsing

- The process of determining the morphemes from which a given word

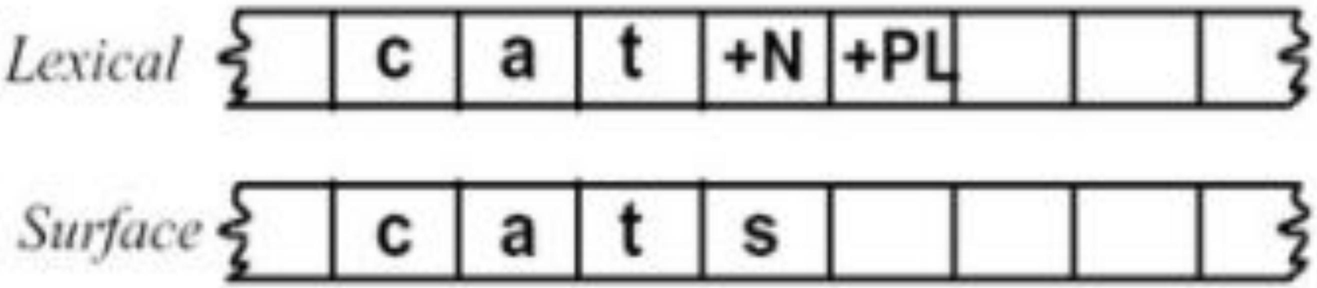


Role of a morphological parser

A	make+V+Past $\longrightarrow$ made	
B	cat+N+Pl $\longrightarrow$ cats ⋮ make+V+Past $\longrightarrow$ maked	
A .P. B	cat+N+Pl $\longrightarrow$ cats ⋮ make+V+Past $\longrightarrow$ made	

Finite State Transducer

- Given the input, for example, *cats*, we would like to produce `cat +N +PL`.
- Two-level morphology, by Koskenniemi (1983)
  - Representing a word as a correspondence between a **lexical level**
    - Representing a simple concatenation of morphemes making up a word, and
  - The **surface level**
    - Representing the actual spelling of the final word.
- Morphological parsing is implemented by building mapping rules that maps letter sequences like *cats* on the surface level into morpheme and features sequence like `cat +N +PL` on the lexical level.



- The automaton we use for performing the mapping between these two levels is the **finite-state transducer** or **FST**.
  - A transducer maps between one set of symbols and another;
  - An FST does this via a finite automaton.
- Thus an FST can be seen as a two-tape automaton which **recognizes** or **generates** ***pairs*** of strings.

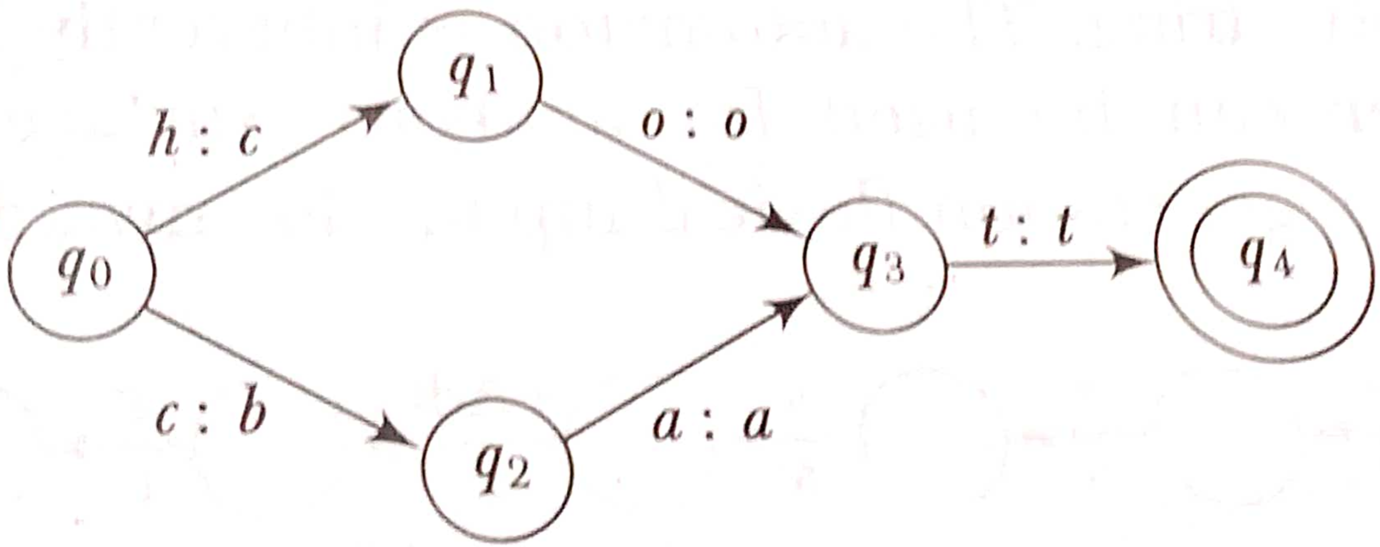


Figure 3.6 Finite-state transducer

- A formal definition of FST (based on the **Mealy machine** extension to a simple FSA):
  - Q: a finite set of N states $q_0, q_1, \ldots, q_N$
  - Σ: a finite alphabet of complex symbols. Each complex symbol is composed of an input-output pair $i : o$; one symbol I from an input alphabet I, and one symbol o from an output alphabet O, thus $\Sigma \subseteq I \times O$. I and O may each also include the epsilon symbol ε.
  - q_0: the start state
  - F: the set of final states, $F \subseteq Q$
  - $\delta(q, i{:}o)$: the transition function or transition matrix between states. Given a state $q \in Q$ and complex symbol $i{:}o \in \Sigma$, $\delta(q, i{:}o)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q.

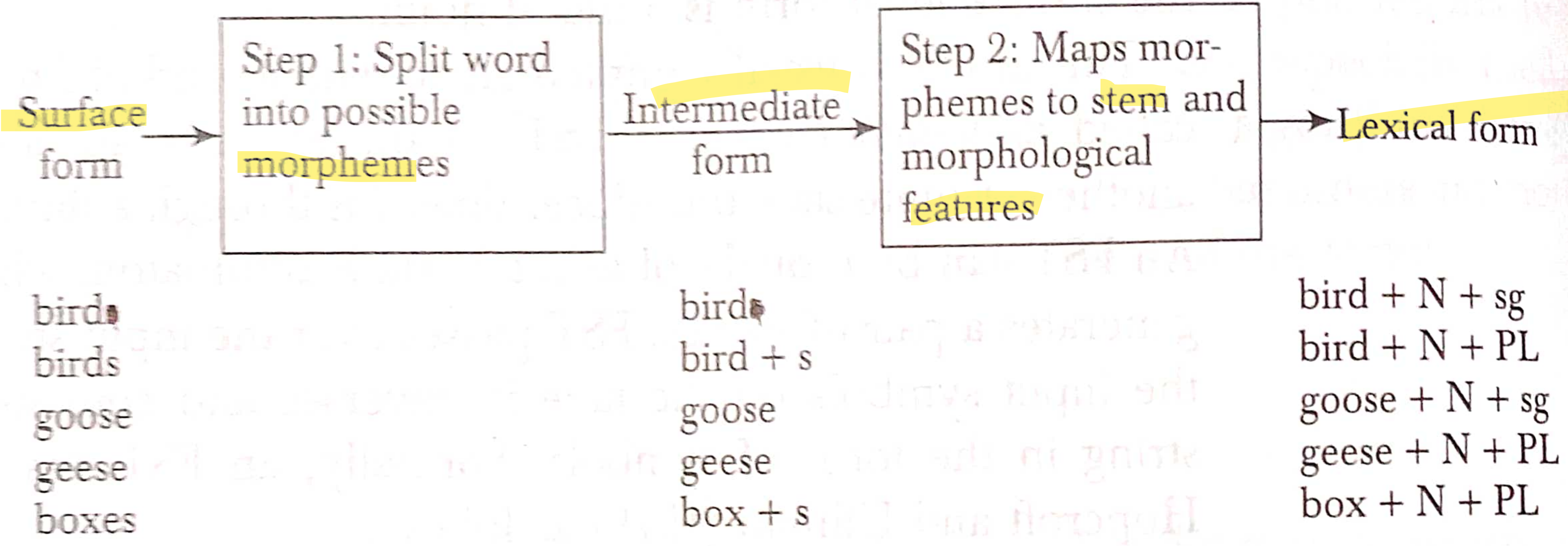


Figure 3.7 Two-step morphological parser

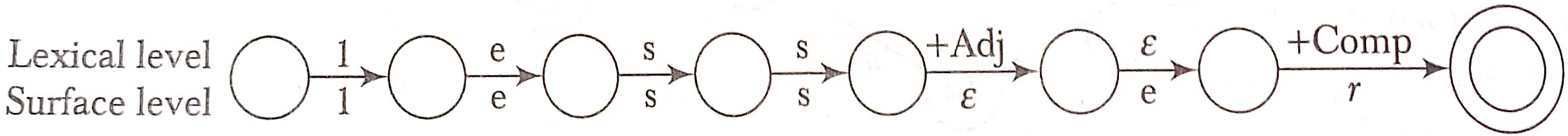


Figure 3.8 A simple FST

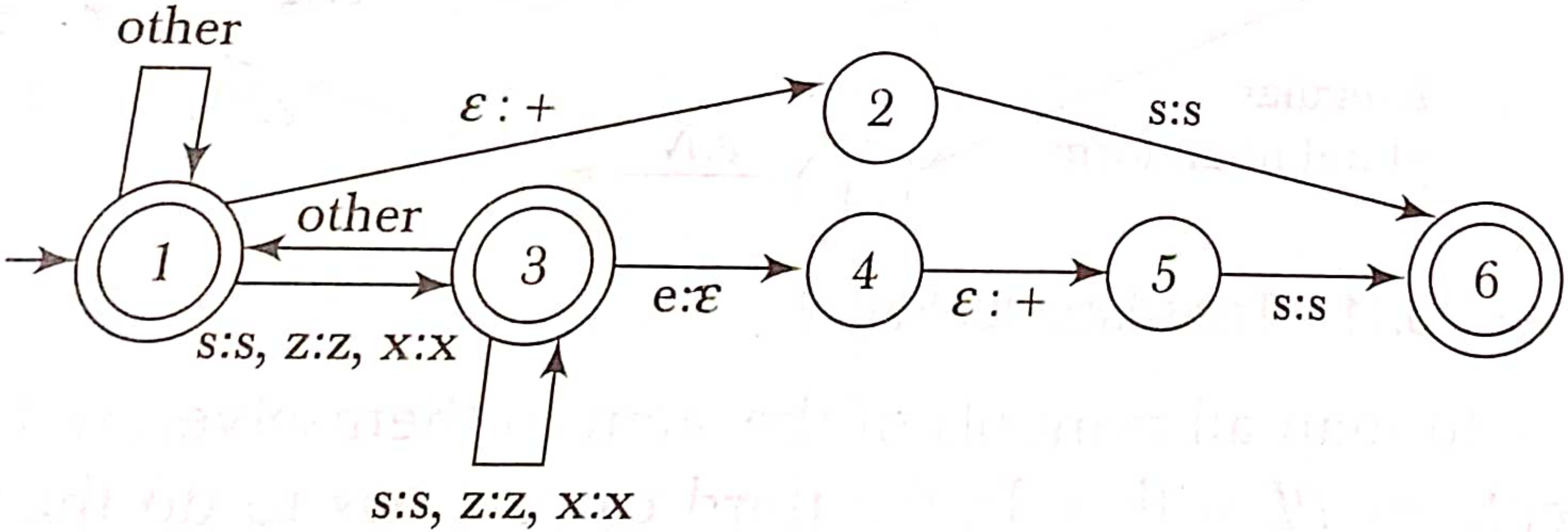


Figure 3.9 A simplified FST, mapping English nouns to the intermediate form

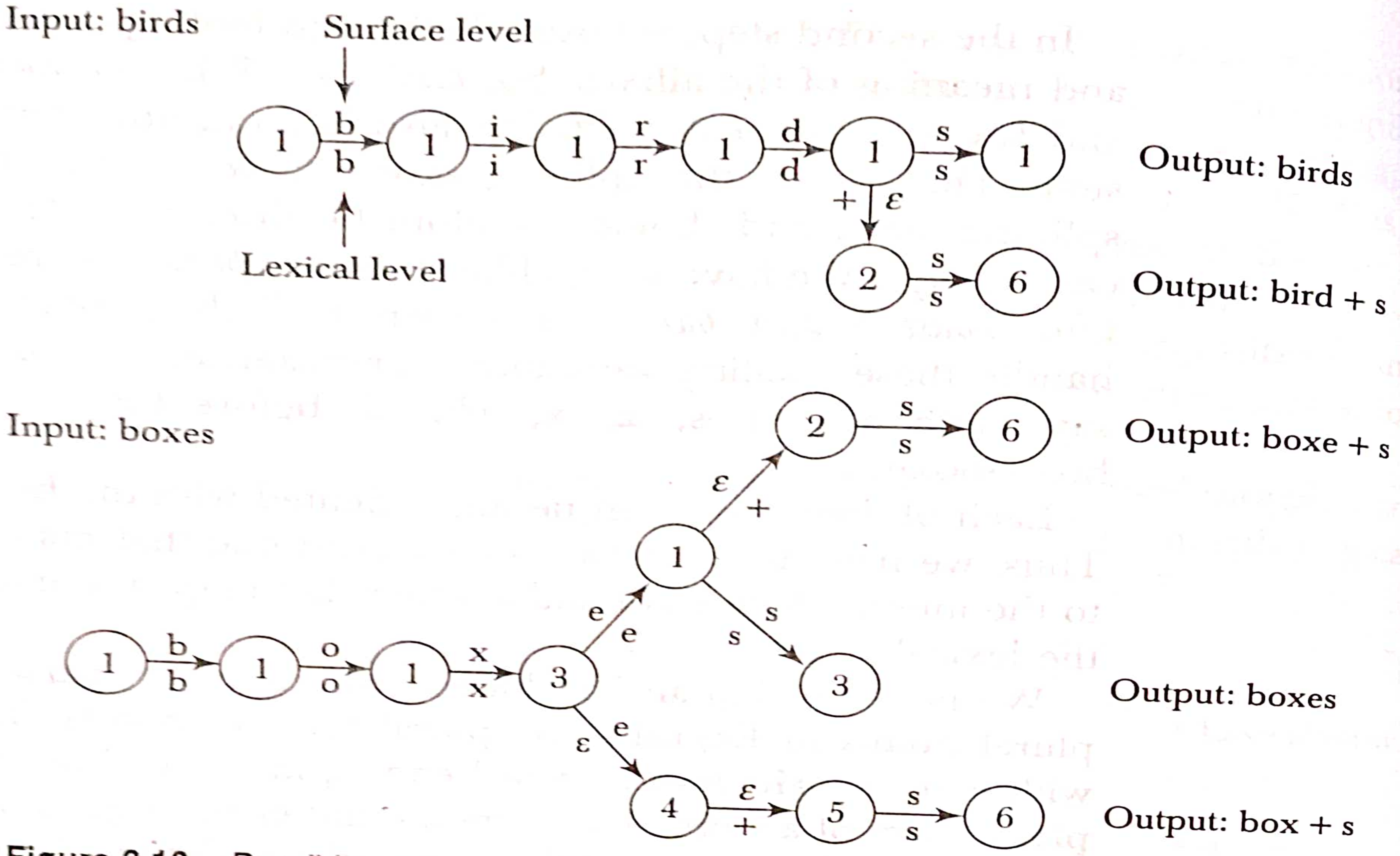


Figure 3.10 Possible sequences of states

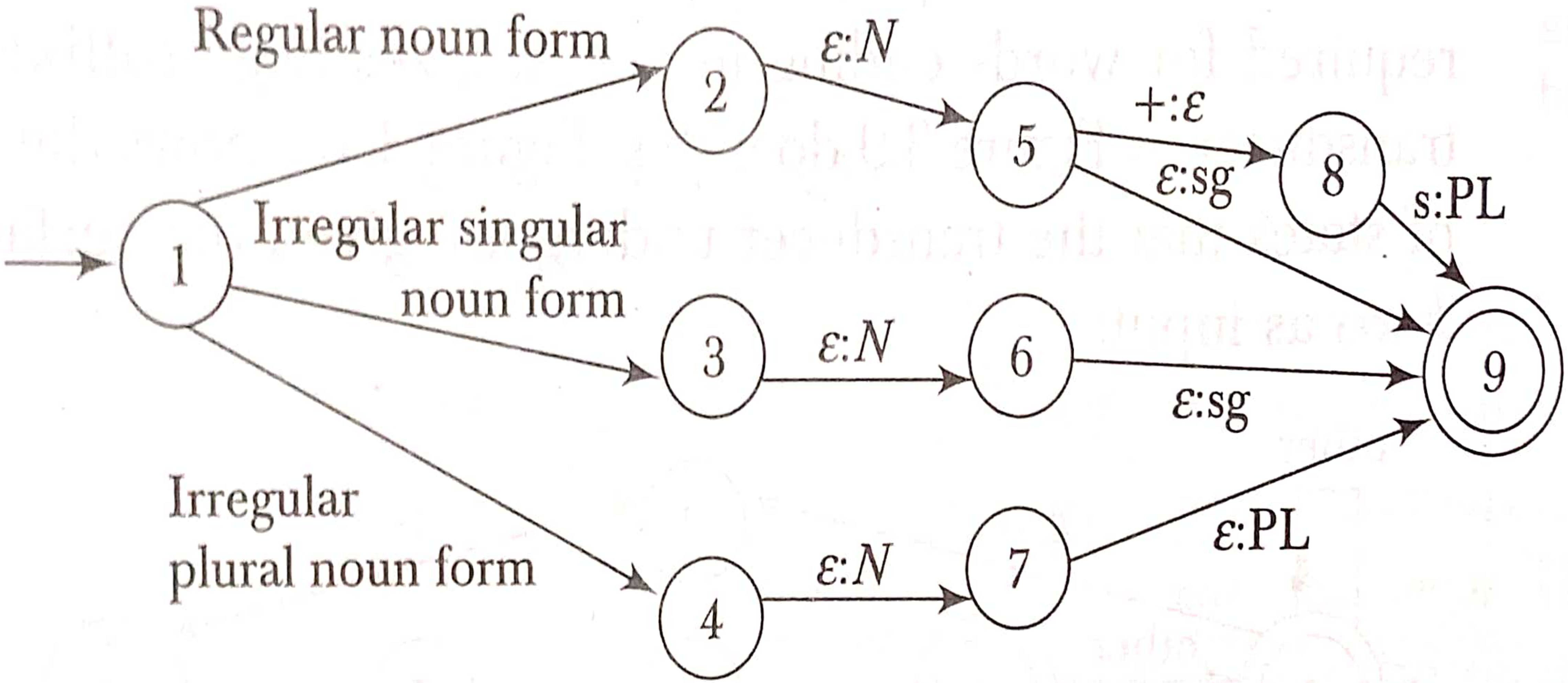


Figure 3.11 Transducer for Step 2

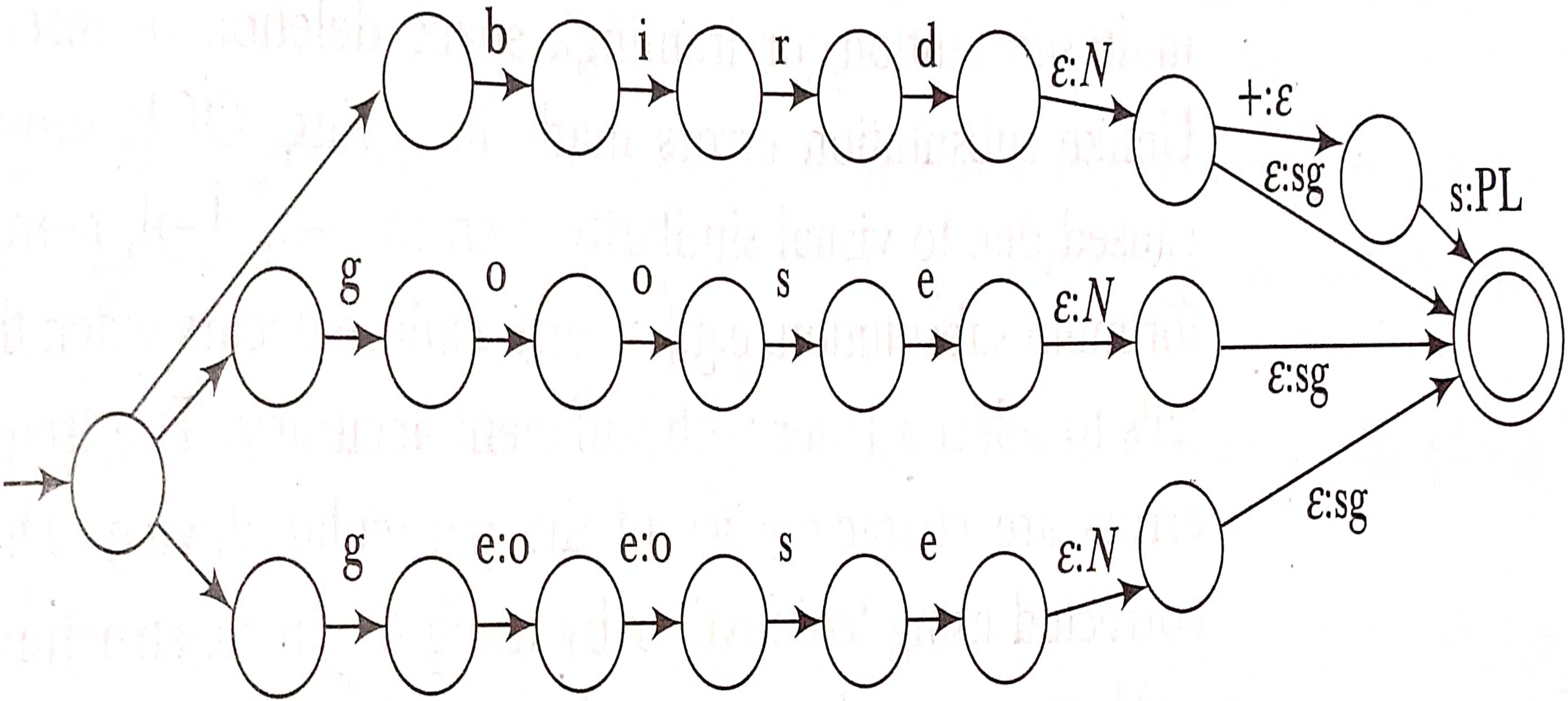


Figure 3.12 A transducer mapping nouns to their stem and morphological features

Spelling Error Detection And Correction

Detection vs. Correction

- There are two distinct tasks:
  - **error detection** = simply find the misspelled words
  - **error correction** = correct the misspelled words
- e.g., It might be easy to tell that ater is a misspelled word, but what is the correct word? water? later? after?
- So, what causes errors?

- Types of errors
  - **insertion** = a letter is added to a word
  - **deletion** = a letter is deleted from a word
  - **substitution** = a letter is put in place of another one
  - **transposition** = two adjacent letters are switched
- Note that the first two alter the length of the word, whereas the second two maintain the same length.
- General properties
  - **single-error misspellings** = only one instance of an error
  - **multi-error misspellings** = multiple instances of errors (harder to identify)

Spelling Error Detection And Correction Techniques

- **Isolated- Error Detection And Correction**
  - Correcting words without taking Context into consideration
  - **Limitations:**
    - Lexicon occupy a lot of space
    - Impossible to list all the correct words of the language
    - Fails to find real-word errors(ex: theses-> these)
    - Larger the lexicon, error gets undetected

- **Context-dependent Error Detection And Correction**
  - Correcting words by taking Context into consideration

Spelling Correction Techniques

1. Minimum edit distance
2. Similarity key techniques
3. N-gram based technique
4. Neural nets
5. Rule based Techniques

Minimum edit distance

- In order to rank possible spelling corrections more robustly, we can calculate the **minimum edit distance** = minimum number of operations it would take to convert one word into another.
- For example, we can take the following five steps to convert junk to haiku:
  - 1. junk -> juk (deletion)
  - 2. juk -> huk (substitution)
  - 3. huk -> hku (transposition)
  - 4. hku -> hiku (insertion)
  - 5. hiku -> haiku (insertion)
- But is this the minimal number of steps needed?

Similarity key techniques

- Problem: How can we find a list of possible corrections?
- Solution: Store words in different boxes in a way that puts the similar words together.
- Example:
  - 1. Start by storing words by their first letter (first letter effect),
    - e.g., punc starts with the code P.
  - 2. Then assign numbers to each letter
    - b, f, p, v →1
    - c, g, j, k, q, s, x, z →2
    - d, t → 3
    - l → 4
    - m, n →5
    - r →6
    - e.g., punc -> P052
  - 3. Then throw out all zeros and repeated letters,
    - e.g., P052 -> P52.
  - 4. Look for real words within the same box,
    - e.g., punk is also in the P52 box.

Rule-based methods

- One can generate correct spellings by writing rules:
  - Common misspelling rewritten as correct word:
    - e.g., hte -> the
  - Rules
    - based on inflections:
      - e.g., V+C+ing -> V+CC+ing (where V = vowel and C = consonant)
    - based on other common spelling errors (such as keyboard effects or common transpositions):
      - e.g., Cie -> Cei

Minimum Edit Distance

Agenda

- 01 What is Minimum Edit Distance?
- 02 Example of Minimum Edit Distance
- 03 Minimum Edit Distance using Dynamic Programming

Minimum Edit Distance Problems

1. Tutor → tumour
2. paecflu → peaceful

Tutor □ tumour : Solution

	#	t	u	m	o	u	r
#	0	1	2	3	4	5	6
t	1	0	1	2	3	4	5
u	2	1	0	1	2	3	4
t	3	2	1	1	2	3	4
o	4	3	2	2	1	2	3
r	5	4	3	3	2	2	2