

Scanner

Outline

- Introduction
- How to construct a scanner
- Regular expressions describing tokens
- FA recognizing tokens
- Implementing a DFA
- Error Handling
- Buffering

Introduction

- A scanner, sometimes called a lexical analyzer
- A scanner :
 - gets a stream of characters (source program)
 - divides it into tokens
 - *Tokens* are units that are meaningful in the source language.
 - *Lexemes* are strings which match the patterns of tokens.

Examples of Tokens in C

Tokens	Lexemes
identifier	Age, grade, Temp, zone, q1
number	3.1416, -498127, 987.76412097
string	“A cat sat on a mat.”, “90183654”
open parentheses	(
close parentheses)
Semicolon	;
reserved word if	IF, if, If, iF

Scanning

- When a token is found:
 - It is passed to the next phase of compiler.
 - Sometimes values associated with the token, called attributes, need to be calculated.
 - Some tokens, together with their attributes, must be stored in the symbol/literal table.
 - it is necessary to check if the token is already in the table
- Examples of attributes
 - Attributes of a variable are name, address, type, etc.
 - An attribute of a numeric constant is its value.

How to construct a scanner

- Define tokens in the source language.
- Describe the patterns allowed for tokens.
- Write regular expressions describing the patterns.
- Construct an FA for each pattern. *finite automata*
- Combine all FA's which results in an NFA.
- Convert NFA into DFA
- Write a program simulating the DFA.

Regular Expression

- a character or symbol in the alphabet
- λ : an empty string
- ϕ : an empty set
- if r and s are regular expressions

- $r \mid s$

or

- rs

concatenation

- r^*

- (r)

Extension of regular expr.

- $[a-z]$
 - any character in a range from a to z
- $.$
 - any character
- r^+
 - one or more repetition
- $r?$
 - optional subexpression
- $\sim(a \mid b \mid c), [^abc]$
 - any single character NOT in the set

Examples of Patterns

- $(a \mid A)$ = the set $\{a, A\}$
- $[0-9]^+ = (0 \mid 1 \mid \dots \mid 9) (0 \mid 1 \mid \dots \mid 9)^*$
- $(0-9)? = (0 \mid 1 \mid \dots \mid 9 \mid \lambda)$
- $[A-Za-z] = (A \mid B \mid \dots \mid Z \mid a \mid b \mid \dots \mid z)$
- $A .$ = the string with A following by any one symbol
- $\sim[0-9] = [^0123456789] =$ any character which is not $0, 1, \dots, 9$

Describing Patterns of Tokens

- reservedIF = (IF| if| If| iF) = (I|i)(F|f)
- letter = [a-zA-Z]
- digit = [0-9]
- identifier = letter (letter|digit)*
- numeric = (+|-)? digit⁺ (.^{2,31} digit⁺)? (E (+|-)? digit⁺)? *not 12*

■ Comments

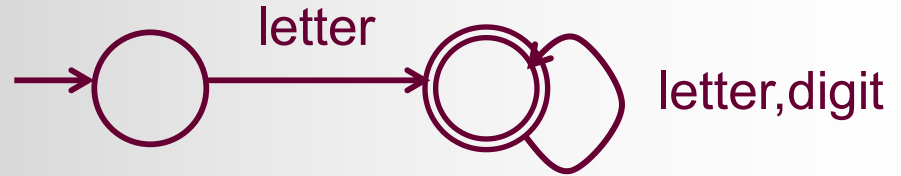
- { (~)* }
- /* ([^*]*[^\/*])* */
- ;(~newline)* newline

Disambiguating Rules

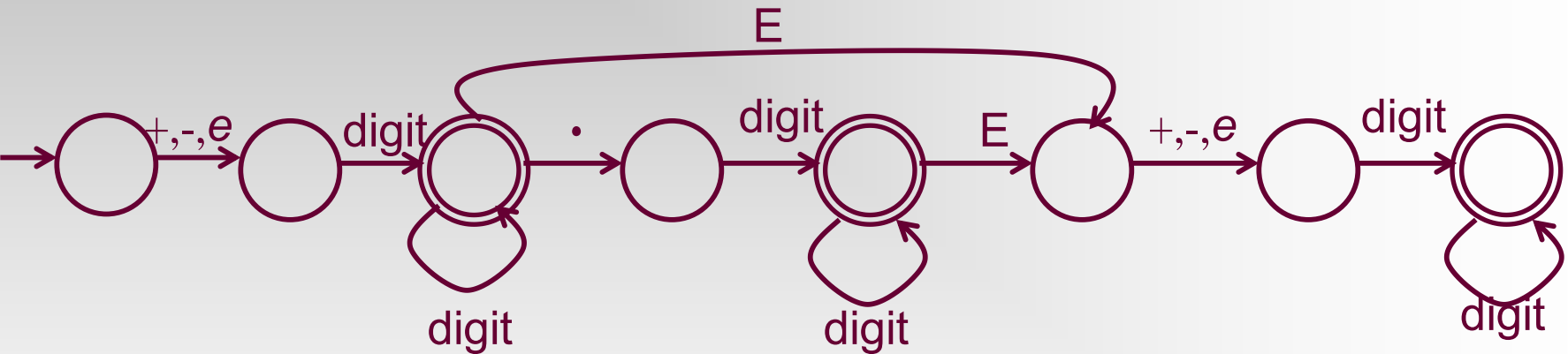
- *IF* is an identifier or a reserved word?
 - A reserved word cannot be used as identifier.
 - A keyword can also be identifier.
- `<=` is `<` and `=` or `<=`?
 - Principle of longest substring
 - When a string can be either a single token or a sequence of tokens, single-token interpretation is preferred.

FA Recognizing Tokens

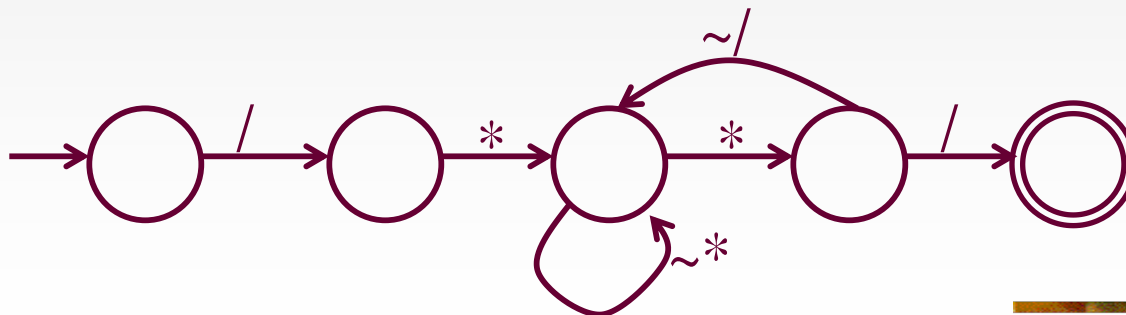
■ Identifier



■ Numeric

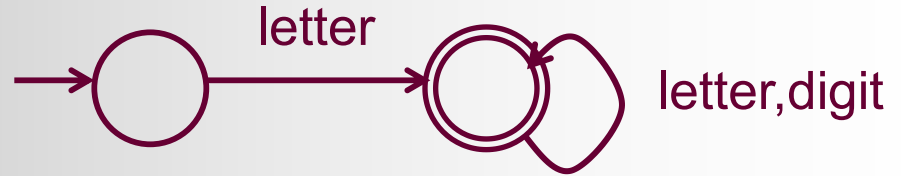


■ Comment



Combining FA's

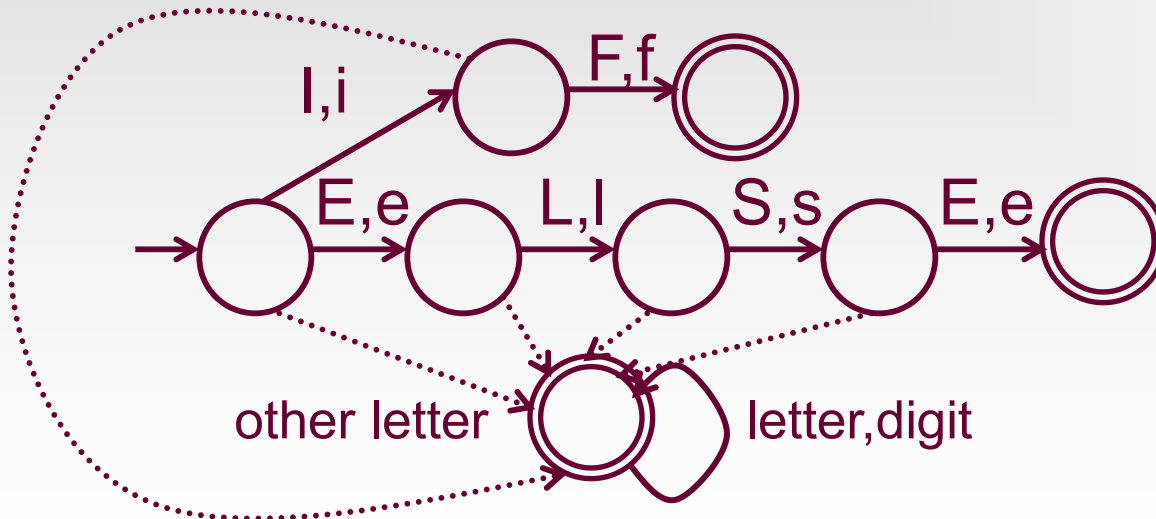
Identifiers



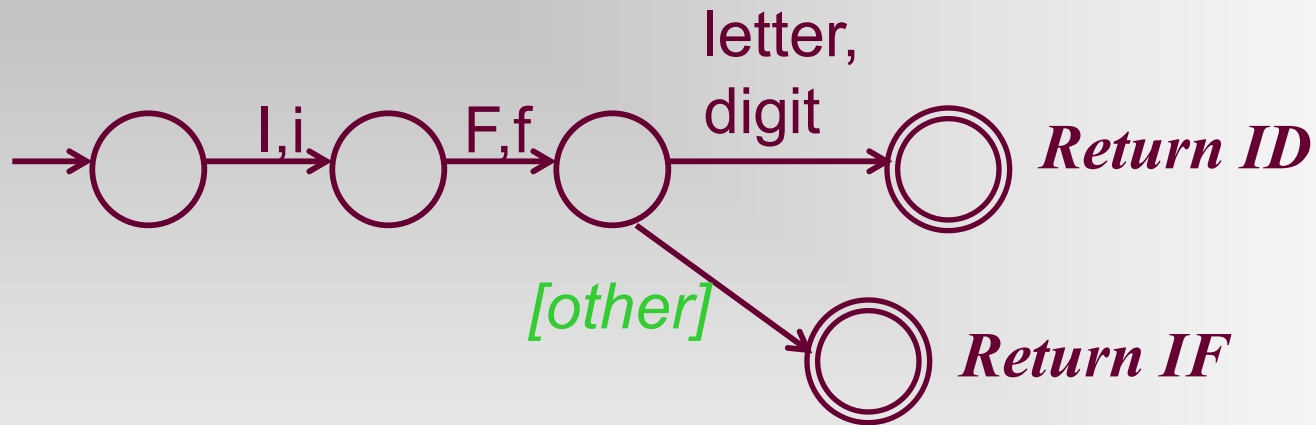
Reserved words



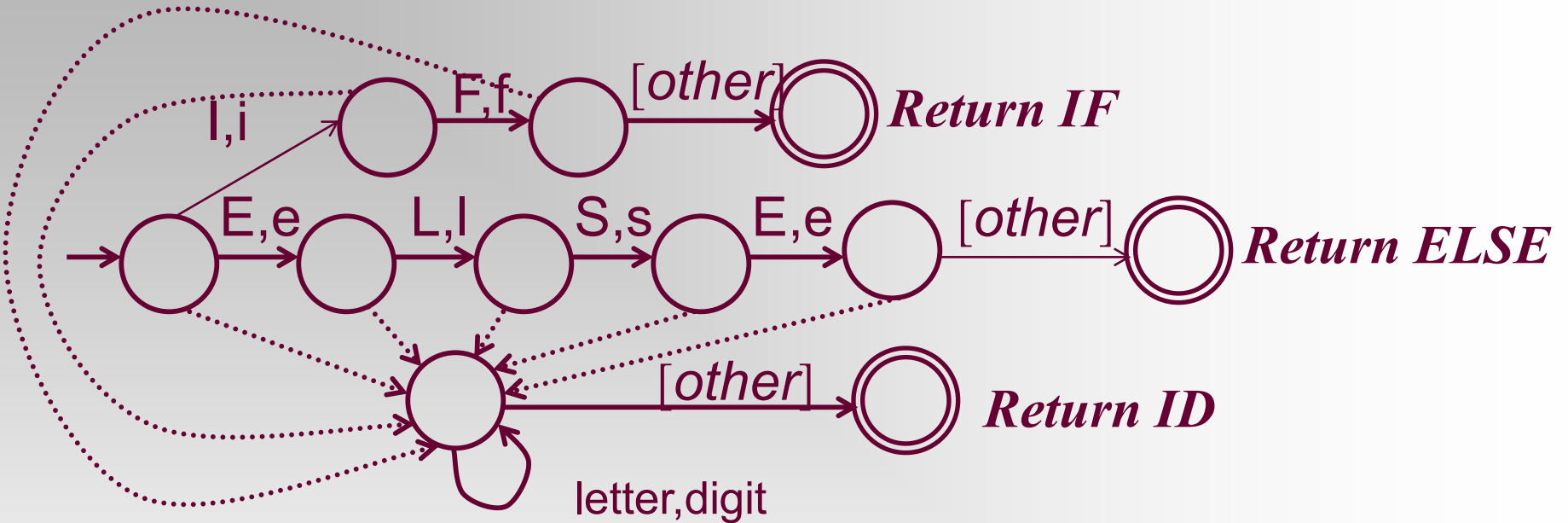
Combined



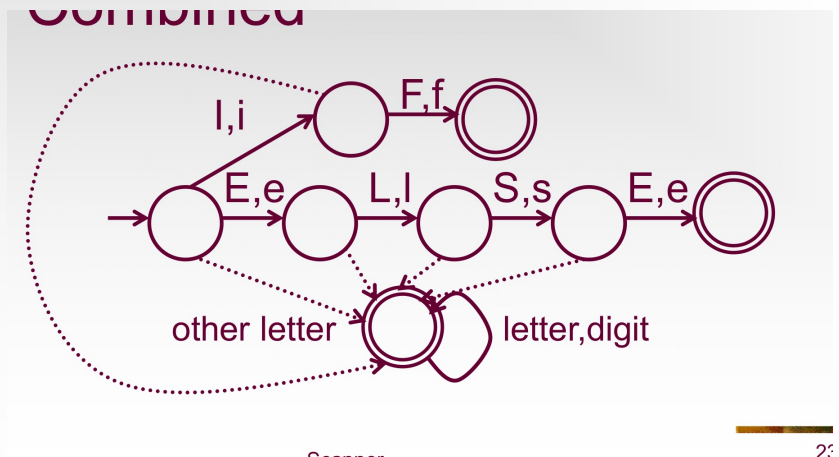
Lookahead



Implementing DFA



- nested-if
- transition table



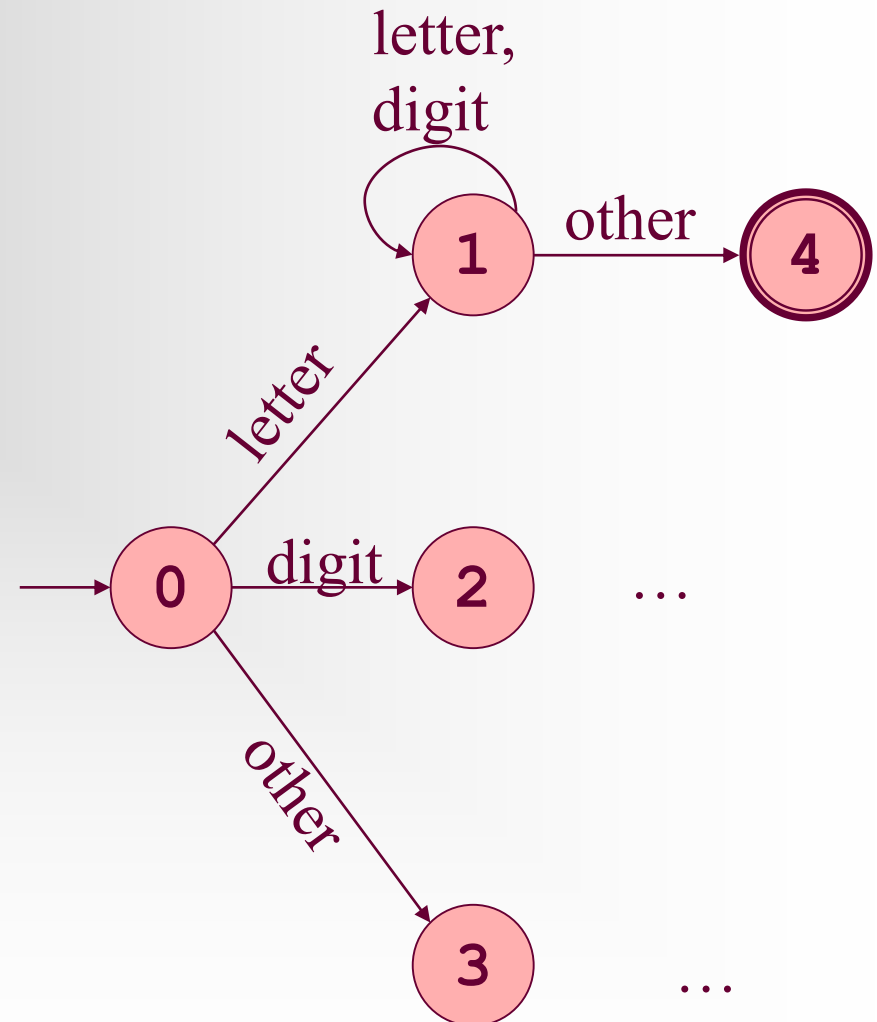
Scanner

23

Nested IF



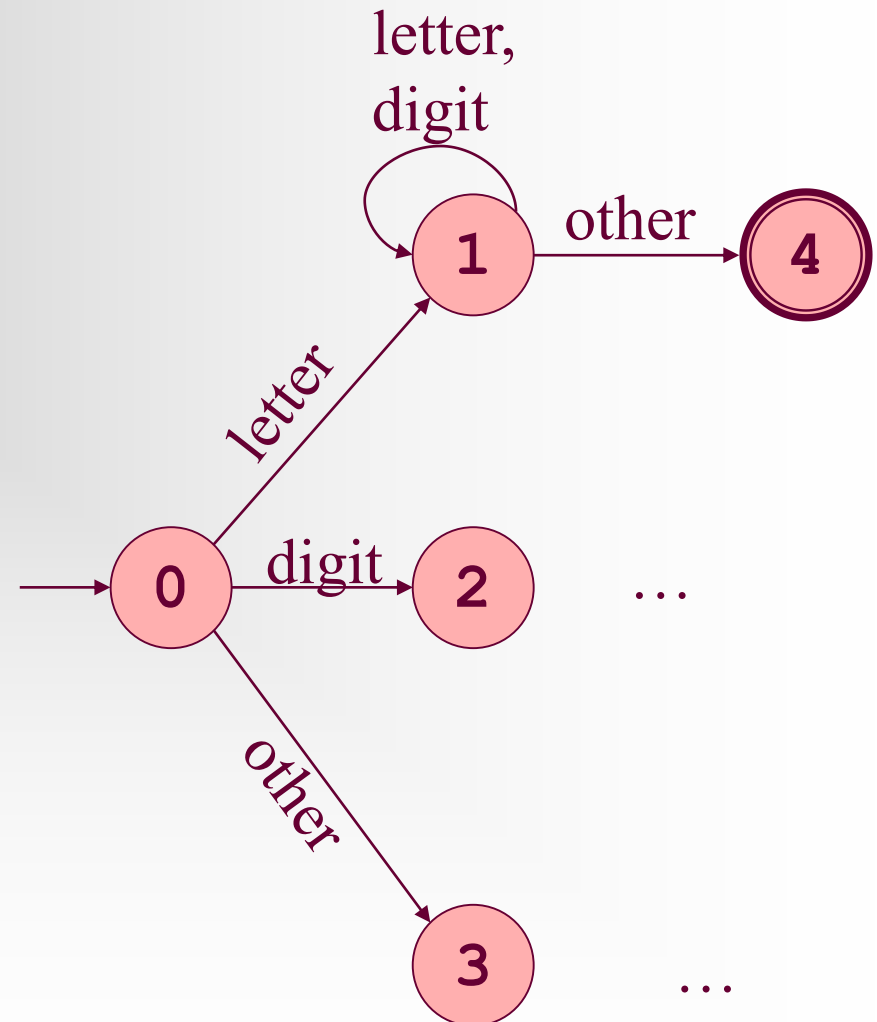
```
switch (state)
{
  case 0:
    {
      if isletter(nxt)
        state=1;
      elseif isdigit(nxt)
        state=2;
      else state=3;
      break;
    }
  case 1:
    {
      if isletVdig(nxt)
        state=1;
      else state=4;
      break;
    }
  ...
}
```



Transition table



St \ ch	0	1	2	3	...
letter	1	1	
digit	2	1	
...	3	4			
..					



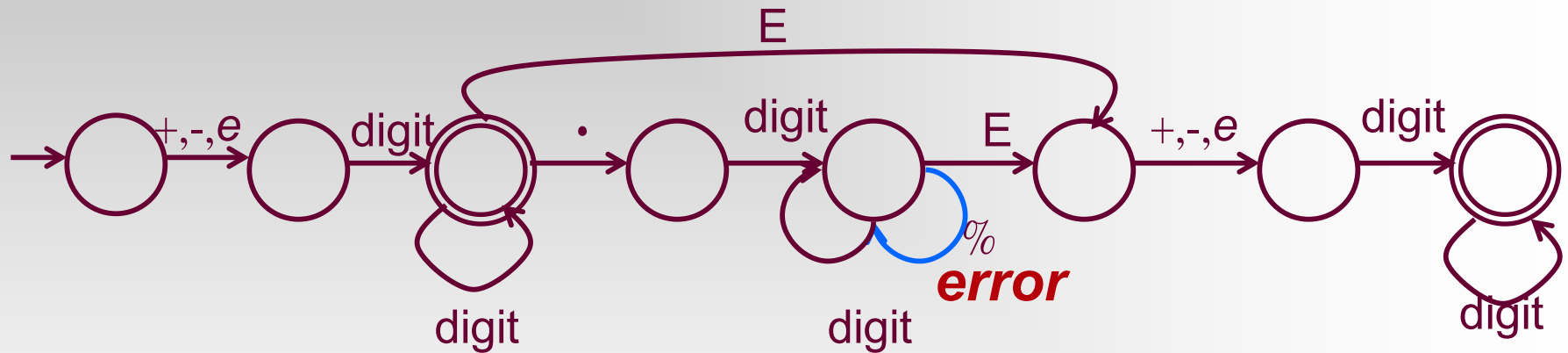
Simulating a DFA

```
initialize current_state=start  
while (not final(current_state))  
{   next_state=dfa(current_state, next)  
    current_state=next_state;  
}
```

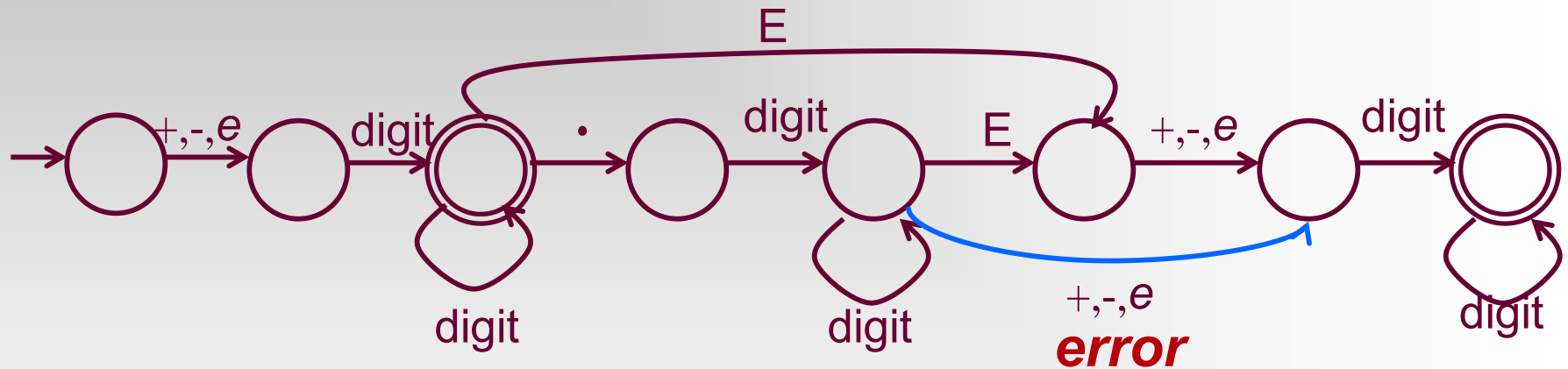
Error Handling

- Delete an extraneous character
- Insert a missing character
- Replace an incorrect character by a correct character
- Transposing two adjacent characters

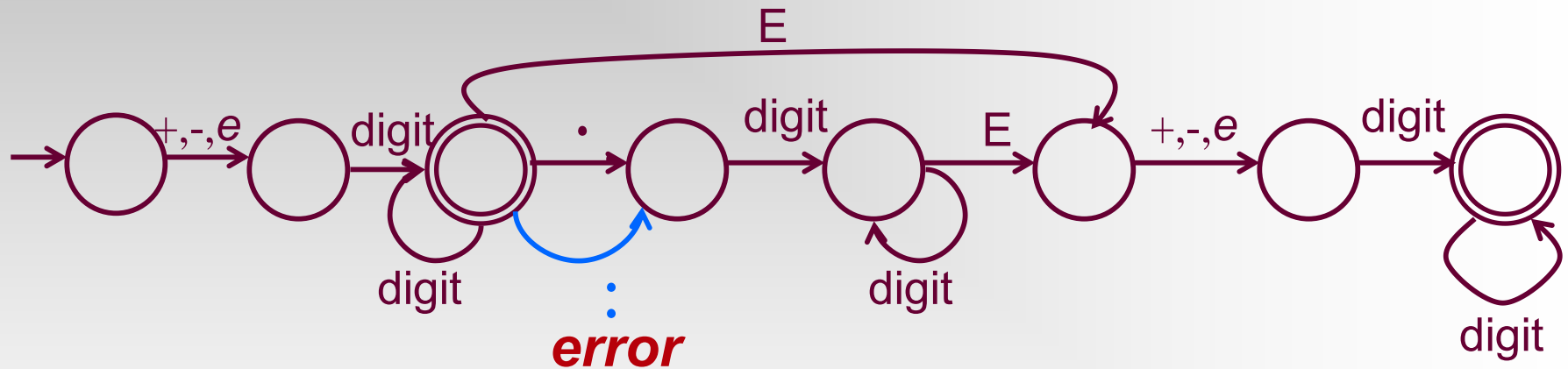
Delete an extraneous character



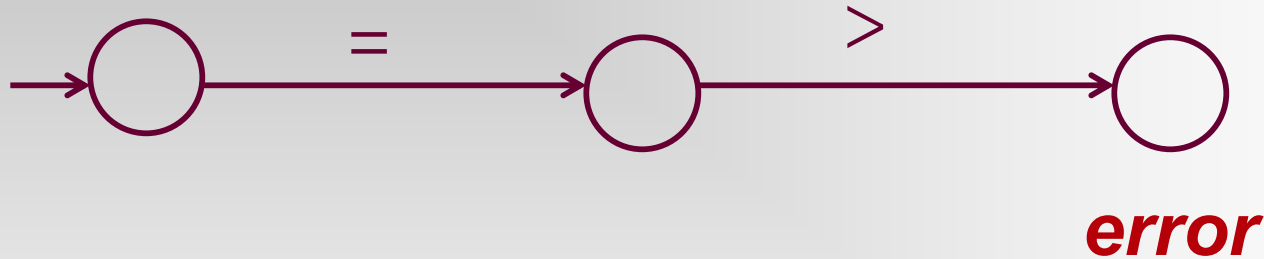
Insert a missing character



Replace an incorrect character



Transpose adjacent characters

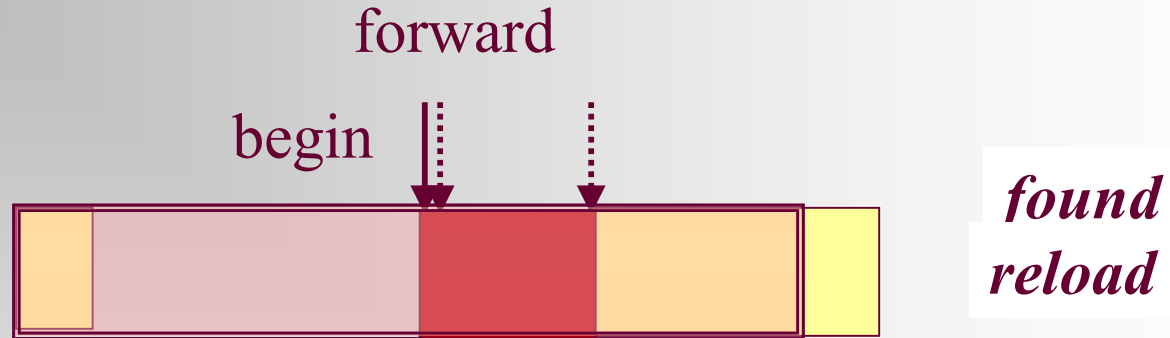


Correct token: >=

Buffering

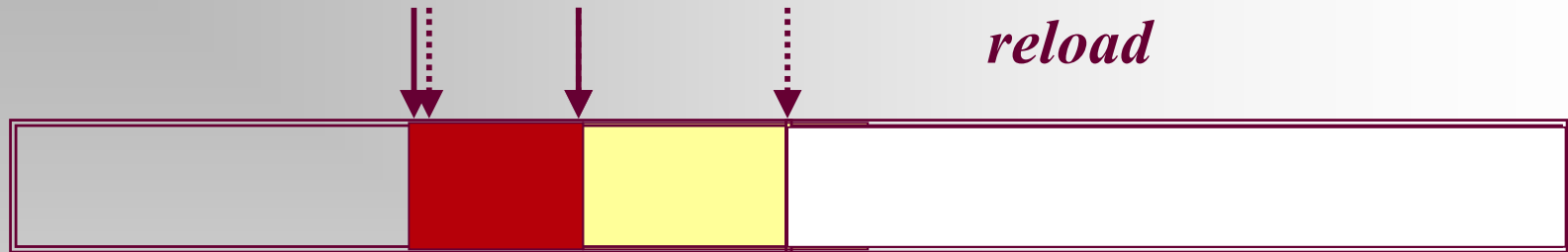
- Single Buffer
- Buffer Pair
- Sentinel

Single Buffer



The first part of the token will be lost if it is not stored somewhere else !

Buffer Pairs



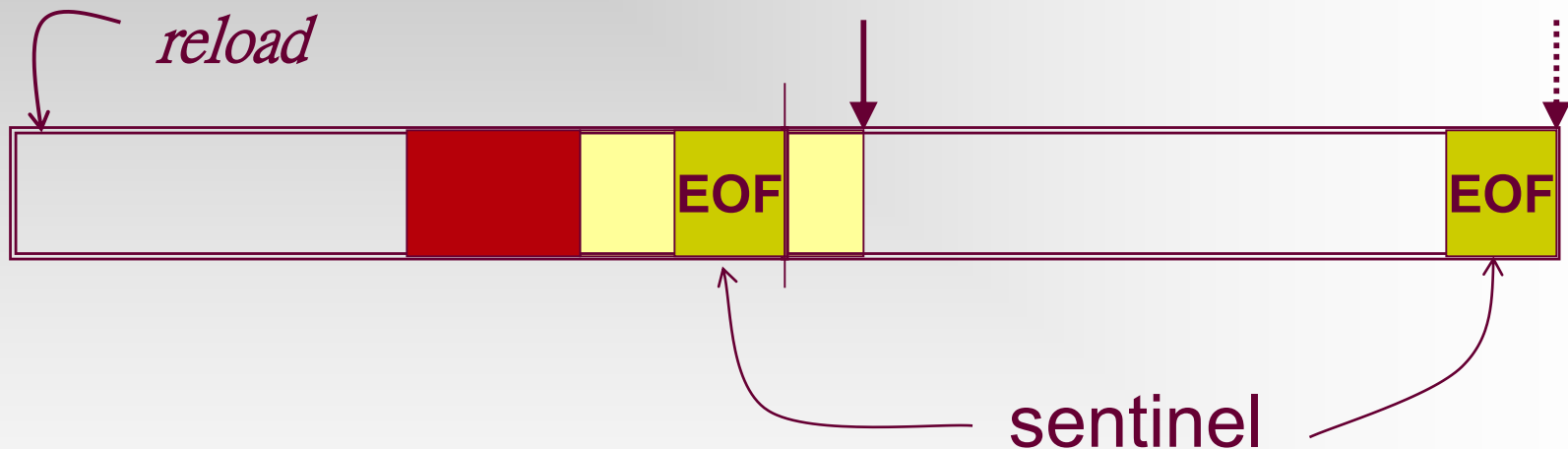
A buffer is reloaded when forward pointer reaches the end of the other buffer.

Similar for the second half of the buffer.

Check twice for the end of buffer if the pointer is not at the end of the first buffer!

Sentinel

For the buffer pair, it must be checked twice for each move of the forward pointer if the pointer is at the end of a buffer.



Using *sentinel*, it must be checked only once for most of the moves of the forward pointer.