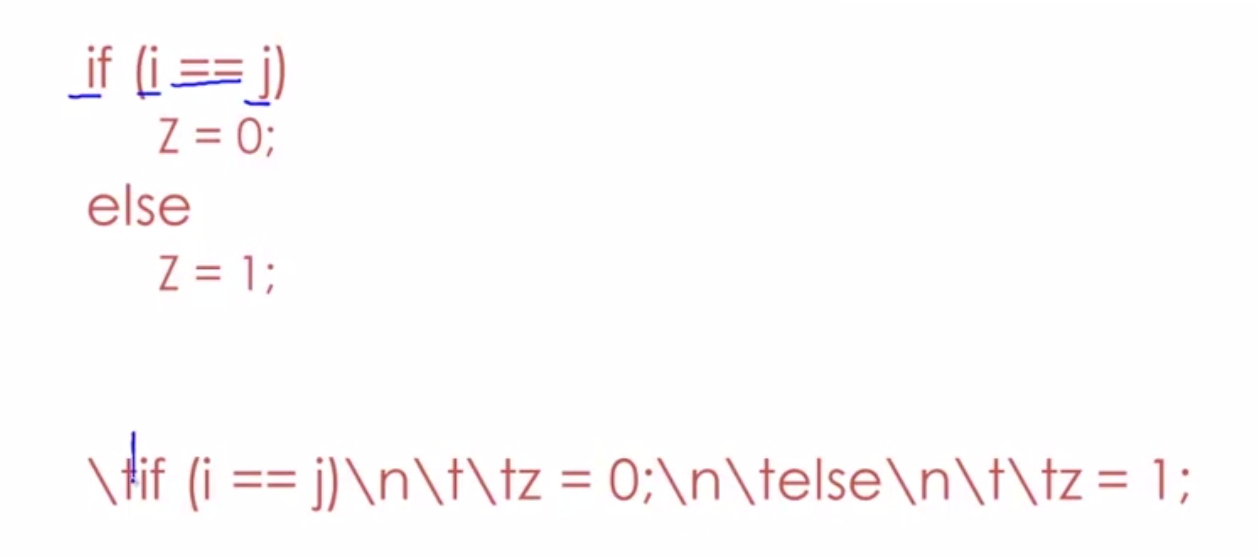
Lexical Analysis:



* Token Class

-In English: noun, verb, adjective

-In a programming language: Identifier, keywords, numbers, ’(’,’)’…

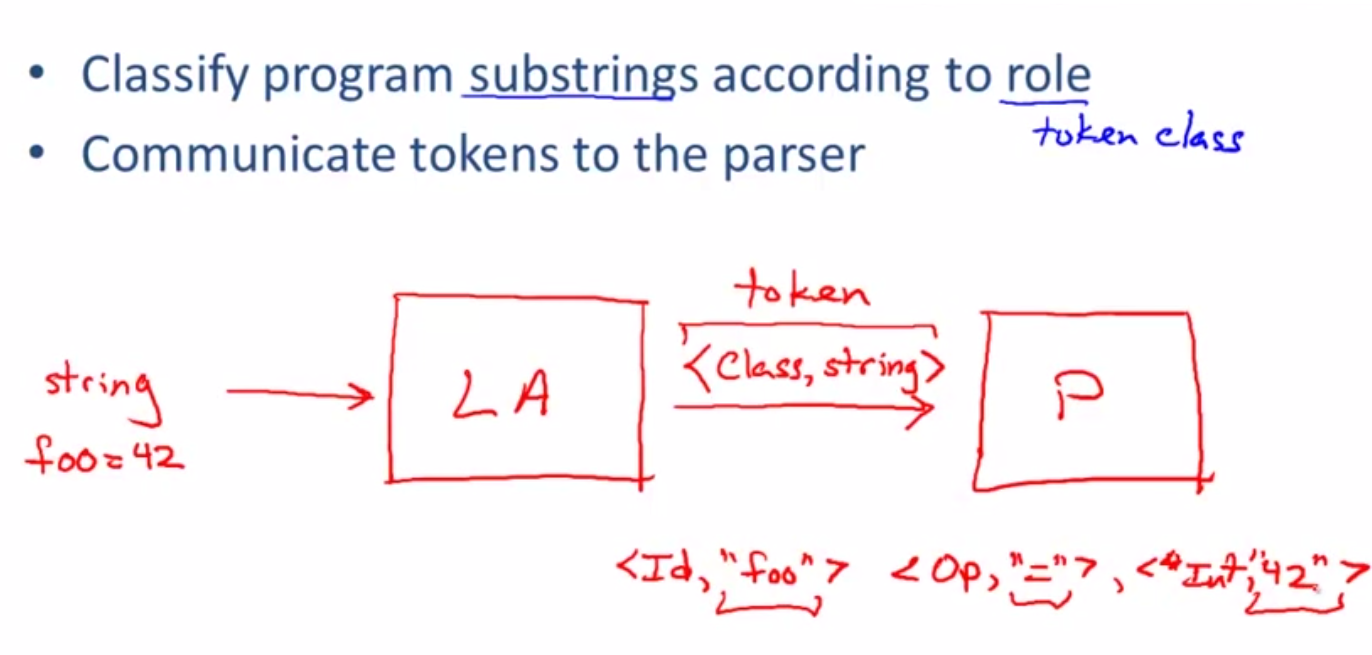
* Identifier:

- strings of letters or digits, starting with a letter,

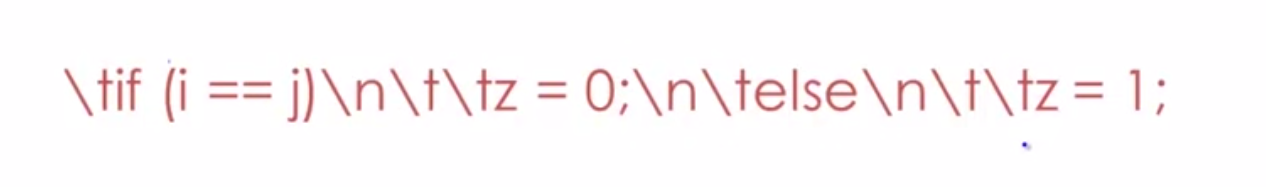
* Integer:

-a non-empty string of digits,

* Keyword: if, else
* Whitespace: tab, new line,



Example:



Whitespace: \t, ,\n,\t,\t

Keywords: if,else,

Identifier:i,j,z

Number:0,1

)

(

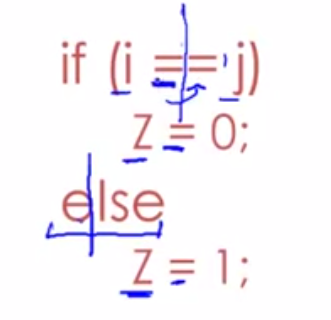
;

=

Operater:==,

1. The goal is to partition the string, this is implemented by reading left-to-right, recognizing one token at a time.
2. “Lookahead” may be required to decide where one token ends and the next token begins.

Lookahead ex:



Regular language:

Lexical structure = token classes,

We must say what set of strings is in a token classes,

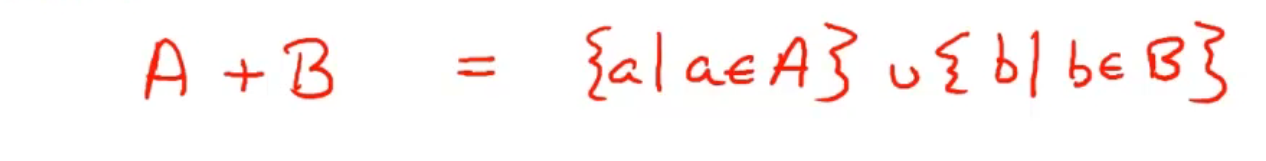
* Use regular languages,
* Single character

‘c’ = {“c”},

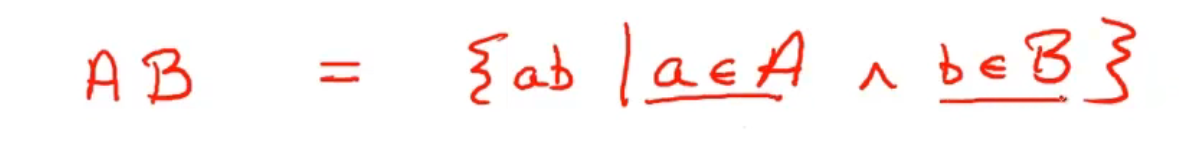
* Epsilon

E ={“”}

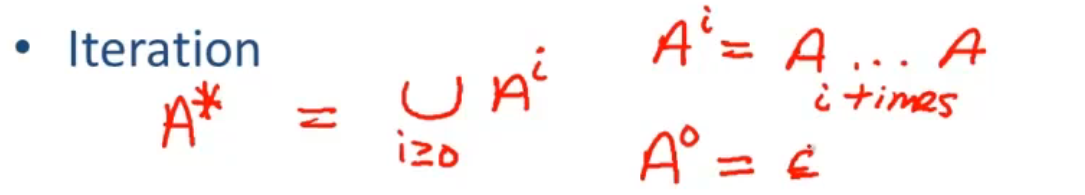
* Union

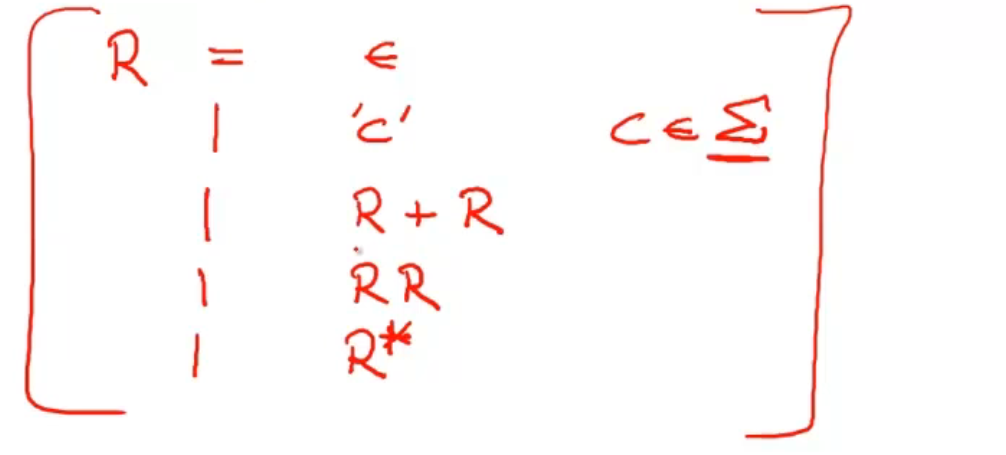


* Concatenation

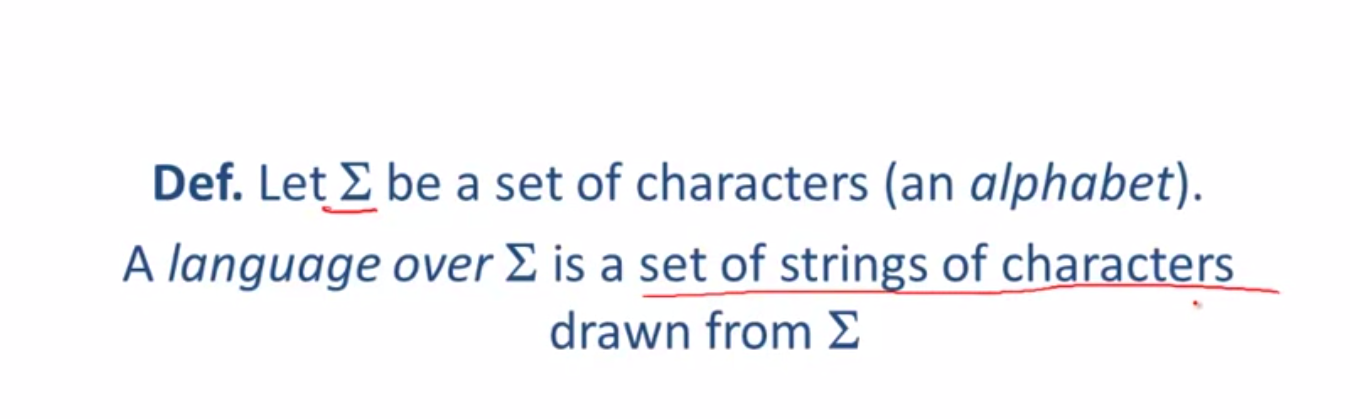


* Iteration





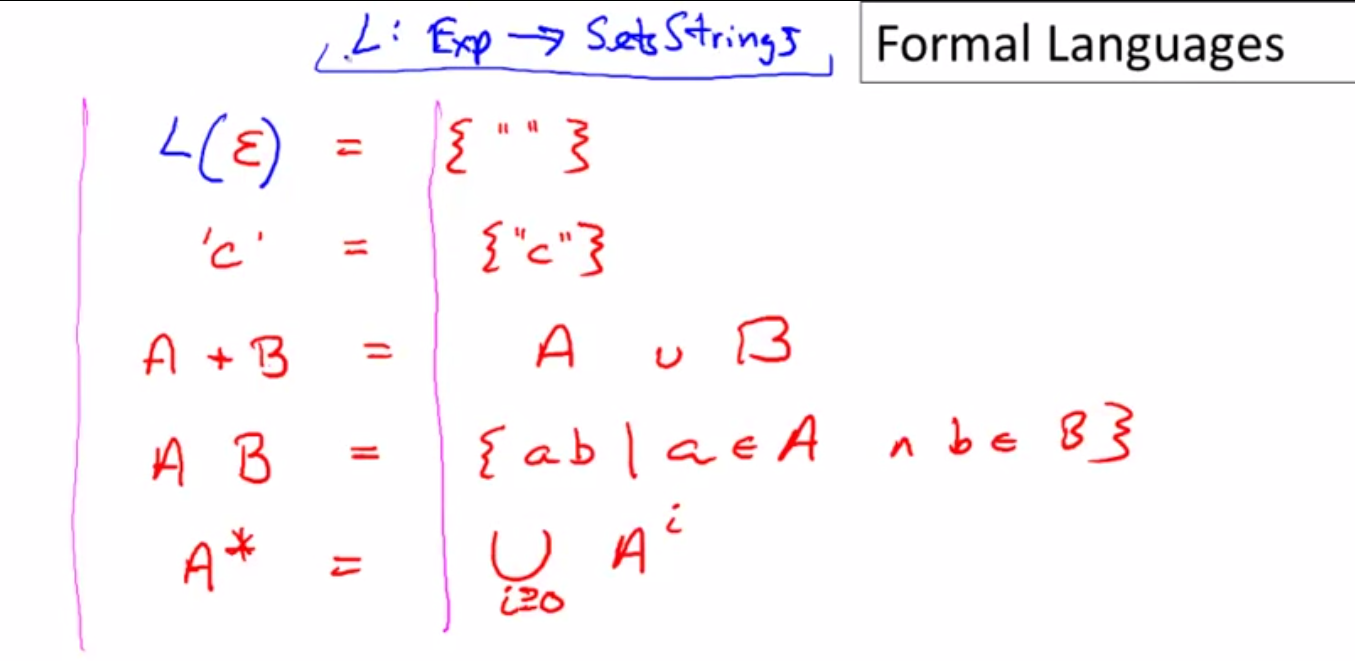
Formal Language:



Meaning function L maps syntax to semantics

L(e) = M

Reg exp set of strings



Meaning is many to one;

-never one to many

Keywords: ”if”,”else”,”then”,”or”…

‘if’+’else’+’then’+…… union all keywords;

Integers: a non-empty string of digits,

Digit= ‘0’+’1’+……

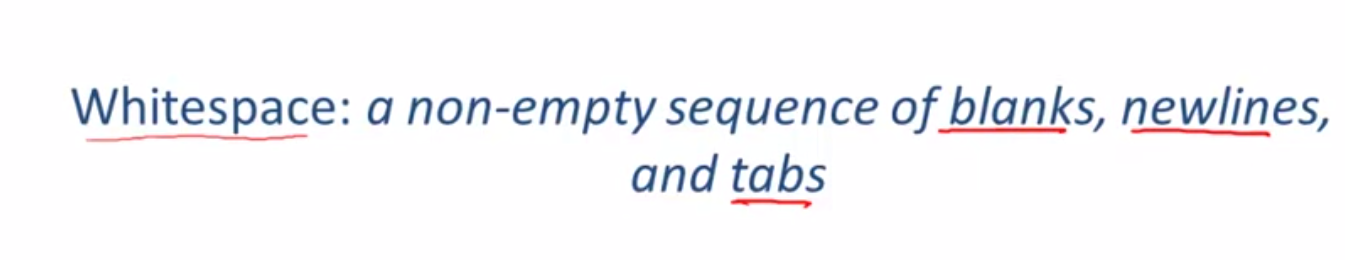
Digit Digit\*, or Digit+

Identifier: string of letters or digits, starting with a letter

Letter= ‘a’+’b’+…..

Or [a-zA-Z]

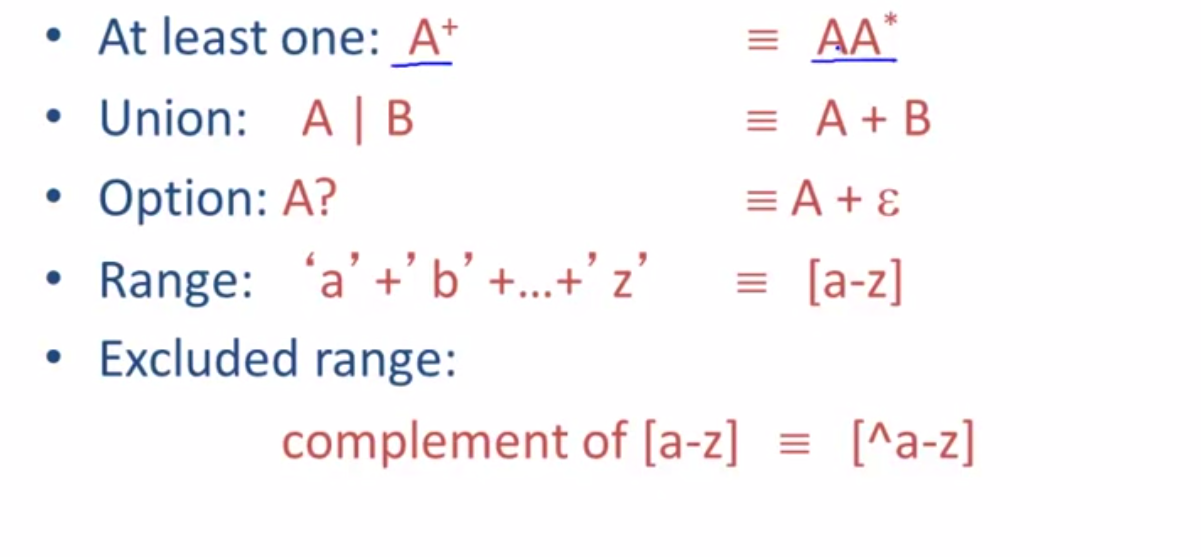
Identifier= letter (letter+digit)\*

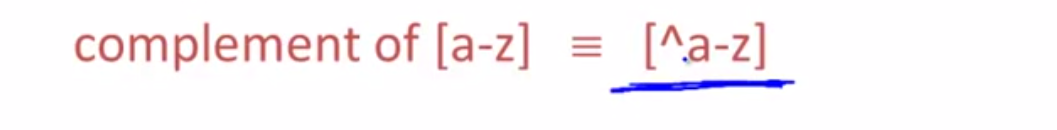


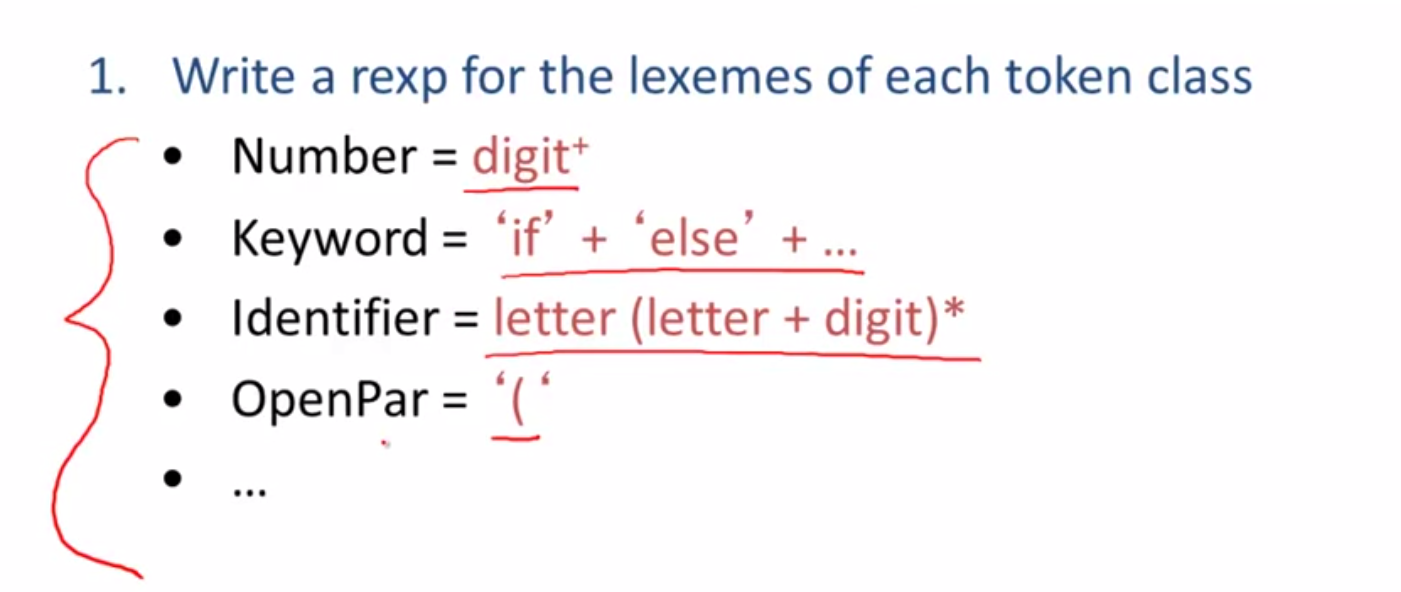
( ‘ ’+ ’\n’+ ‘\t’)+

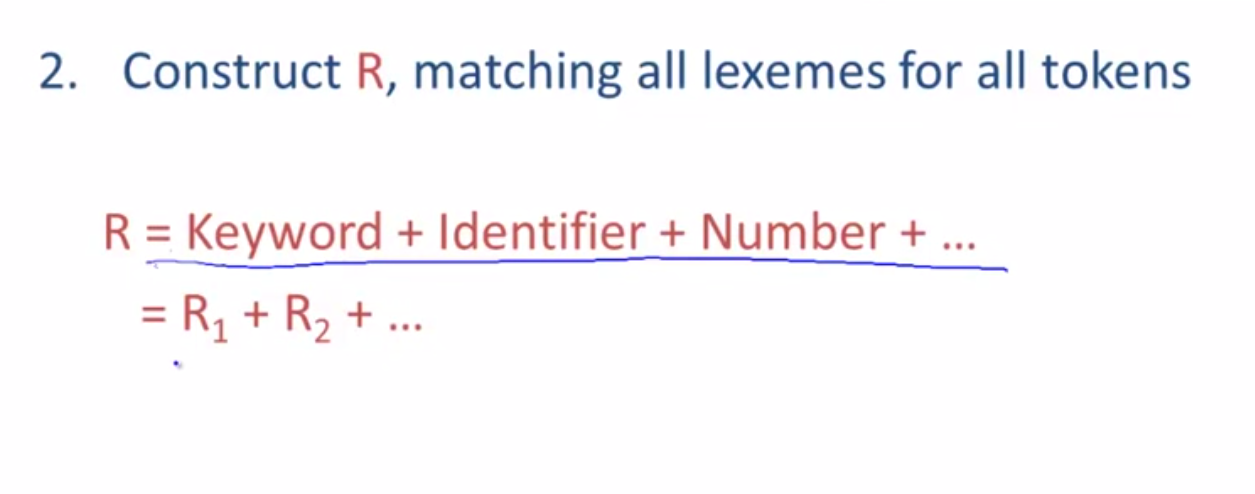
Regular languages are a language specification

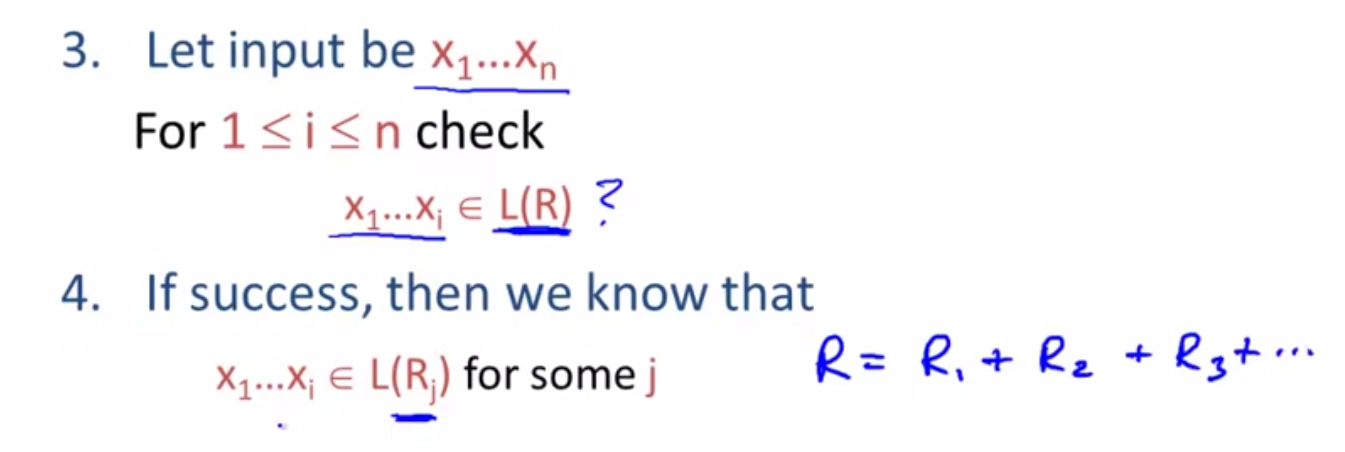
* We still need an implementation

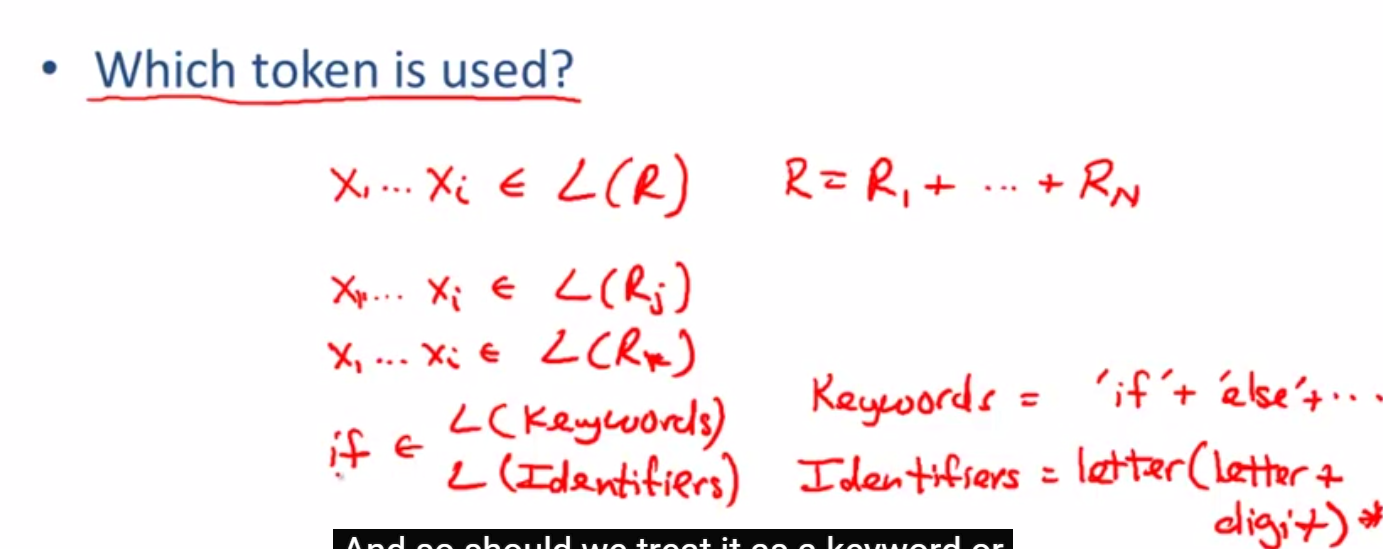






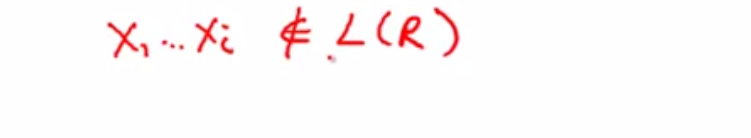






Choose the one listed first,

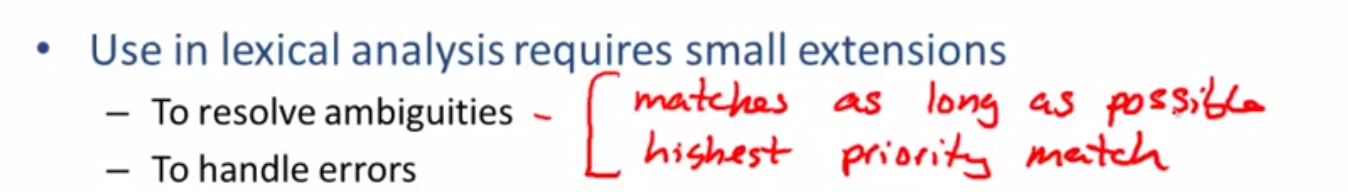
What if no rules matches?



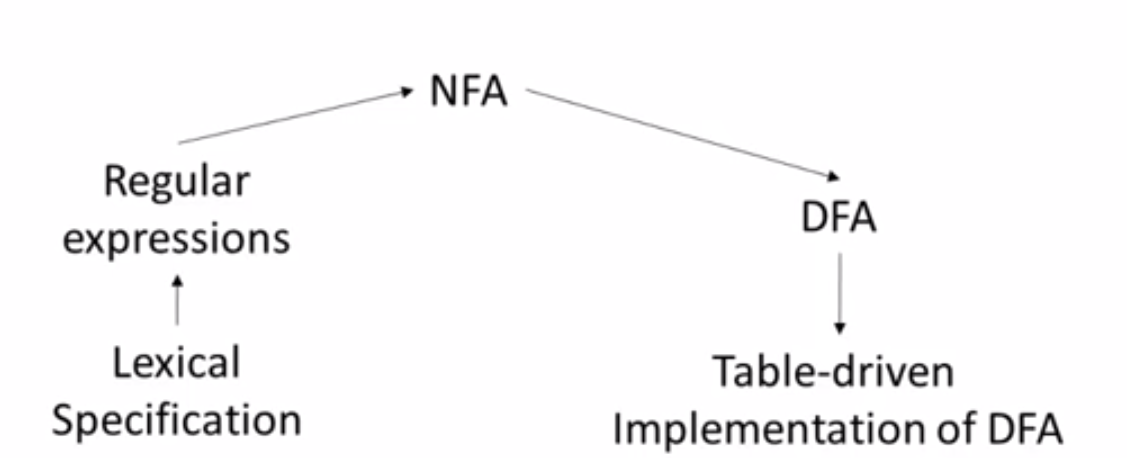


Another regular expression,

Put it last priority,



Regular expression to NFA



NFA to DFA

