

# Lexical Analysis

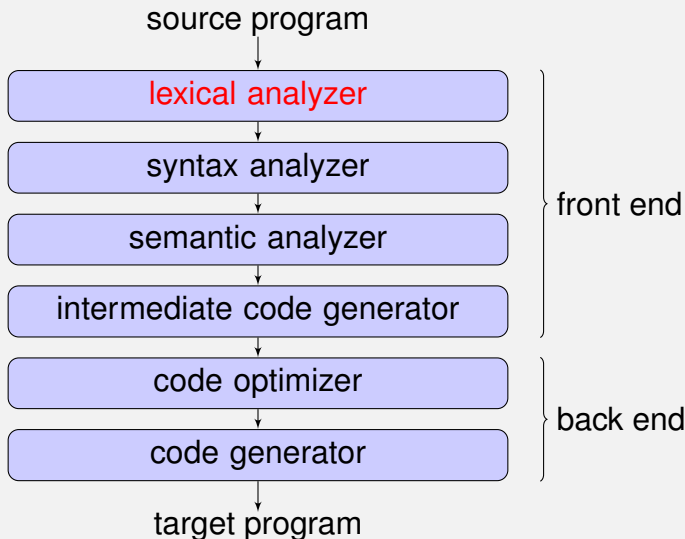
Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

08, 2016

- 1 Introduction
- 2 Roles
- 3 Implementation
- 4 Use ANTLR to generate Lexer
- 5 Regex and Parser Libraries in Scala

# Compilation Phases



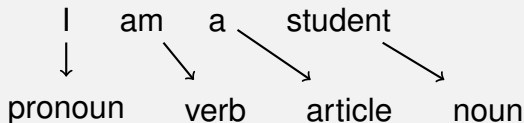
- Like a **word extractor**

in  $\Rightarrow$  i n  $\Rightarrow$  in

- Like a **spell checker**

I ogog to socholsochol

- Like a **classification**



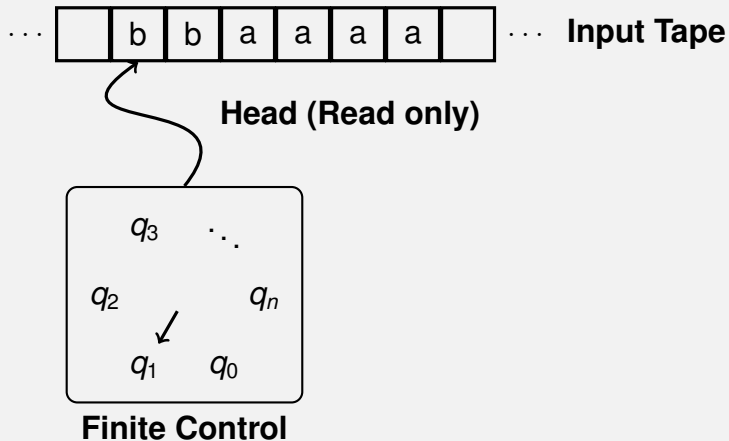
- Identify **lexemes**: substrings of the source program that belong to a grammar unit
- Return **tokens**: a lexical category of lexemes
- Ignore **spaces** such as blank, newline, tab
- Record the **position** of tokens that are used in next phases

*result = oldsum - value / 100;*

<b>Lexemes</b>	<b>Tokens</b>
<i>result</i>	IDENT
<i>=</i>	ASSIGN_OP
<i>oldsum</i>	IDENT
<i>-</i>	SUBSTRACT_OP
<i>value</i>	IDENT
<i>/</i>	DIV_OP
<i>100</i>	INT_LIT
<i>;</i>	SEMICOLON

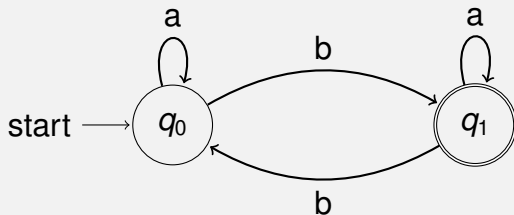
# How to build a lexical analyzer?

- How to build a lexical analysis for English?
  - 65000 words
  - Simply build a dictionary:  
{(I,pronoun);(We,pronoun);(am,verb);...}
  - Extract, search, compare
- But for a programming language?
  - How many words?
    - Identifiers: abc, cab, Abc, aBc, cAb, ...
    - Integers: 1, 10, 120, 20, 210, ...
    - ...
  - Too many words to build a dictionary, so how?





# State Diagram



Input: abaabb

Current state	Read	New State
$q_0$	a	$q_0$
$q_0$	b	$q_1$
$q_1$	a	$q_1$
$q_1$	a	$q_1$
$q_1$	b	$q_0$
$q_0$	b	$q_1$

## Definition

Deterministic Finite Automaton(DFA) is a 5-tuple  $M = (K, \Sigma, \delta, s, F)$  where

- $K$  = a finite set of state
- $\Sigma$  = alphabet
- $s \in K$  = the initial state
- $F \subseteq K$  = the set of final states
- $\delta$  = a transition function from  $K \times \Sigma$  to  $K$

# Example

$M = (K, \Sigma, \delta, s, F)$

where  $K = \{q_0, q_1\}$

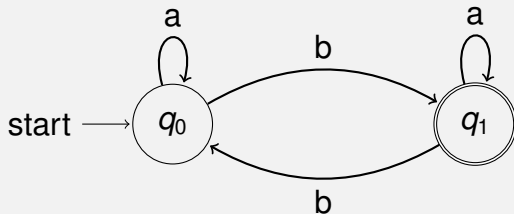
$\Sigma = \{a, b\}$

$s = q_0$

$F = \{q_1\}$

and  $\delta$

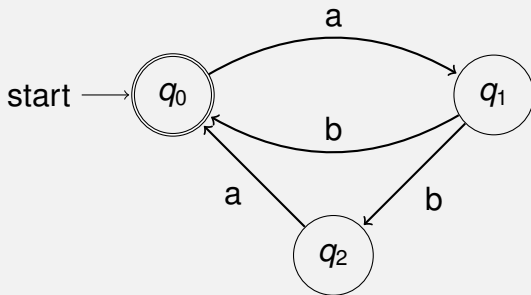
$K$	$\Sigma$	$\delta(K, \Sigma)$
$q_0$	a	$q_0$
$q_0$	b	$q_1$
$q_1$	a	$q_1$
$q_1$	b	$q_0$



- Permit several possible “next states” for a given combination of current state and input symbol
- Accept the empty string  $\epsilon$  in state diagram
- Help simplifying the description of automata
- Every NFA is equivalent to a DFA

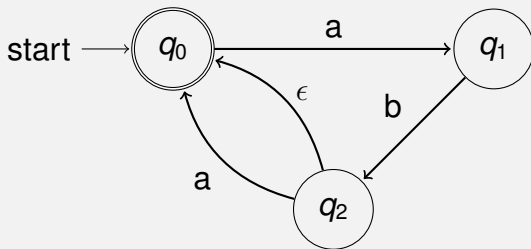
# Example

Language  $L = (\{ab\} \cup \{aba\})^*$



# Example

Language  $L = (\{ab\} \cup \{aba\})^*$



- Describe regular sets of strings
- Symbols other than ( ) | \* stand for themselves
- Use  $\epsilon$  for an empty string
- Concatenation  $\alpha \beta$  = First part matches  $\alpha$ , second part  $\beta$
- Union  $\alpha \mid \beta$  = Match  $\alpha$  or  $\beta$
- Kleene star  $\alpha^*$  = 0 or more matches of  $\alpha$
- Use ( ) for grouping

**(i|I)(f|F)**

Keyword **if** of language Pascal

- if
- IF
- If
- iF

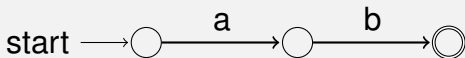
**E(0|1|2|3|4|5|6|7|8|9)\***

An E followed by a (possibly empty) sequence of digits

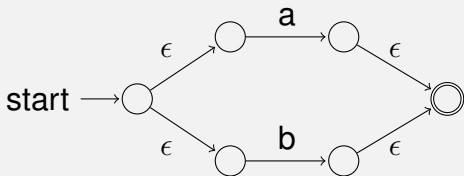
- E123
- E9
- E



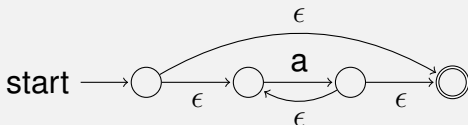
# Regular Expression and Finite Automata



$ab$



$a \mid b$



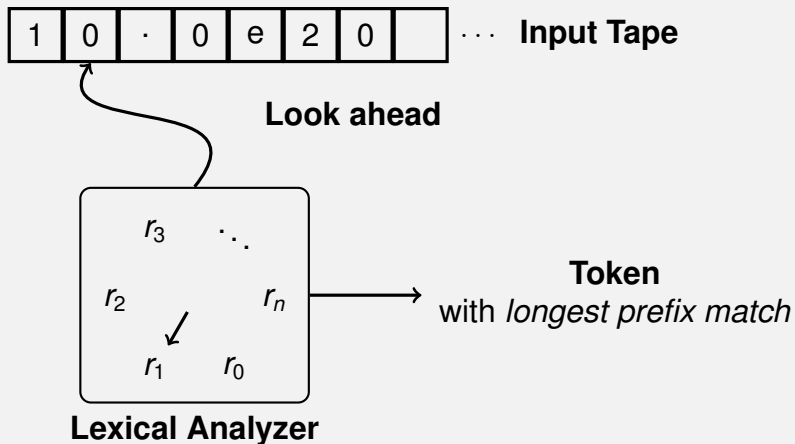
$a^*$

- $\alpha^+ =$  one or more (i.e.  $\alpha\alpha^*$ )
- $\alpha? =$  0 or 1 (i.e.  $(\alpha|\epsilon)$ )
- $[xyz] = x|y|z$
- $[x-y] =$  all characters from x to y, e.g.  $[0-9] =$  all ASCII digits
- $[\^x-y] =$  all characters other than  $[x-y]$
- $.$  matches any character

- Integer:
- Hexadecimal number:
- Fixed-point number:
- Floating point number:
- String:

- ANother Tool for Language Recognition
- Terence Parr, Professor of CS at the Uni. San Francisco
- powerful parser/lexer generator

```
/**  
 * Filename: Hello.g4  
 */  
lexer grammar Hello;  
  
// match any digits  
INT: [0-9]+;  
  
// Hexadecimal number  
HEX: 0[Xx][0-9A-Fa-f]+;  
  
// match lower-case identifiers  
ID : [a-z]+ ;  
  
// skip spaces, tabs, newlines  
WS : [ \t\r\n]+ -> skip ;
```



Library	import scala.util.matching.Regex
Construction	new Regex(String)
	new Regex("[0-9]+")
	"[0-9]+".r
Method	findFirstIn(String):Option[Match]
	findFirstMatchIn(String):Option[String]
	findPrefixOf(String):Option[String]
	findPrefixMatchOf(String):Option[String]
	findAllIn(String):MatchIterator
	...

```
import scala.util.matching.Regex
val pat = new Regex("[0-9]+")
val pattern = "[a-z][a-z]*".r
val str = "123 abc 456"
pat.findFirstIn(str)
pattern.findFirstIn(str)
```



Library	scala.util.parsing.combinator.Parsers.Parser	
Construction	new Parser[T] new Parser[Token] new Parser[Any]	
Method	~	p1 ~ p2: must match p1 followed by p2
		p1   p2: must match either p1 or p2, with preference given to p1
	?	p1.? : may match p1 or not
	*	p1.*: matches any number of repetitions of p1
	^^	p1 ^^ f: combine for function application
	^^^	p1 ^^^ T: changes a successful result into the specified value
	...	...

- A lexical analyzer is a pattern matcher that isolates small-scale parts of a program
- Regular expressions are built based on Finite Automata
- How to write a lexical analyzer (lexer) in Scala

- [1] ANTLR, <http://antlr.org>, 19 08 2016.