Lexical Analysis

Dr. Nguyen Hua Phung

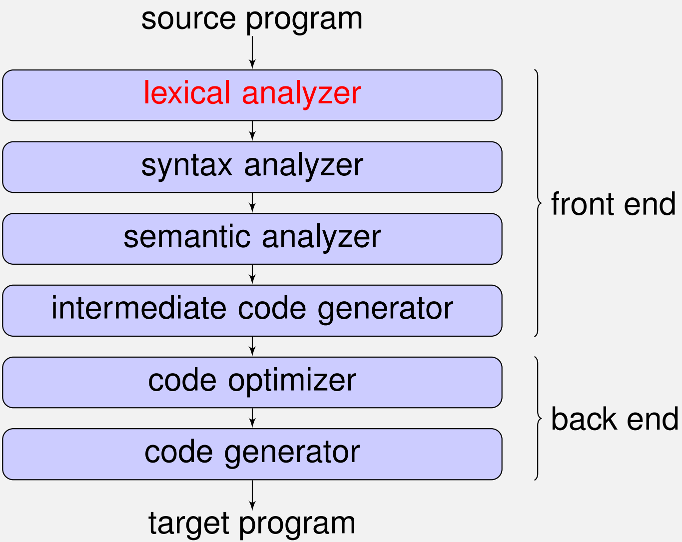
HCMC University of Technology, Viet Nam

08, 2016

Outline

1. Introduction
2. Roles
3. Implementation
4. Use ANTLR to generate Lexer

Compilation Phases



- Like a **word extractor**
  in ⇒ i n ⇒ in
- Like a **spell checker**
  I ogog to socholsochol
- Like a **classification**

  I am a student

  pronoun verb article noun

Lexical Analysis Roles

- 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

Example on Lexeme and Token

r	e	s	u	l	t	' '	=	' '	o	ldsum - value / 100;

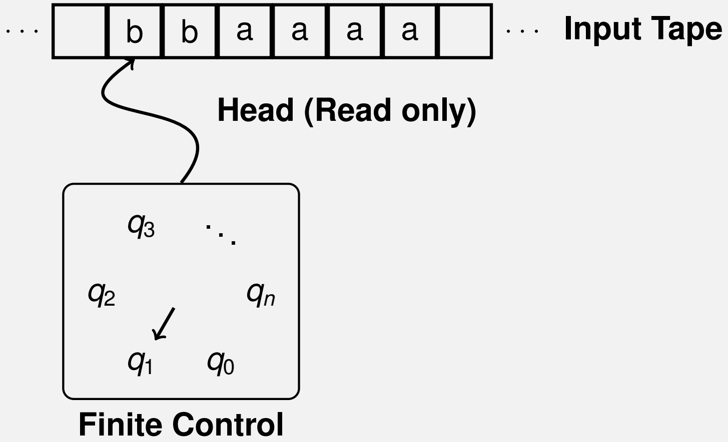
Lexemes	**Kind of Tokens**
result	IDENT
=	ASSIGN_OP
oldsum	IDENT
-	SUBSTRACT_OP
value	IDENT
/	DIV_OP
100	INT_LIT
;	SEMICOLON

How to build a lexical analyzer?

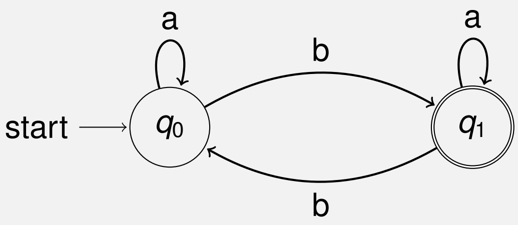
- How to build a lexical analysis for English?
  - 65000 words
  - Simply build a dictionary: {(I,pronoun);(We,pronoun);(am,verb);...}
  - Extract, search, compare
- But for a programming language?
  - How many words?
    - Identifiers: abc, cab, Abc, aBc, cAb, ...
    - Integers: 1, 10, 120, 20, 210,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

Rule Representations

- Finite Automata
  - Deterministic Finite Automata
  - Nondeterministic Finite Automata
- Regular Expressions


· · ·
b
b
a
a
a
a
· · ·
Input Tape
Head (Read only)
q_3
q_2
q_n
q_1
q_0
Finite Control

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

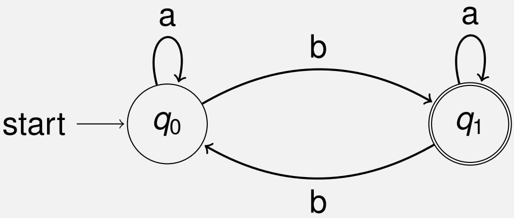
Deterministic Finite Automaton(DFA) is a 5-tuple
M =(K,Σ,δ,s,F) where

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

Example

M =(K,Σ,δ,s,F)
where K = $\{q_0, q_1\}$ Σ = {a,b} s=q_0 F=$\{q_1\}$
and δ

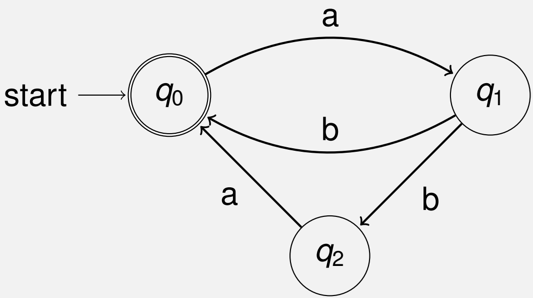
K	Σ	$\delta(K, \Sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0



Nondeterministic Finite Automata

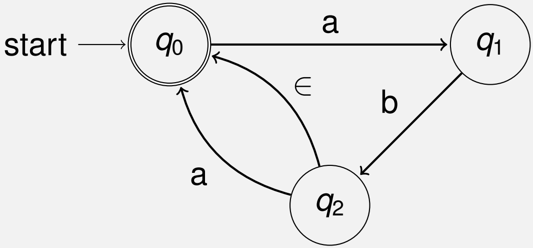
- Permit several possible "next states" for a given combination of current state and input symbol
- Accept the empty string $\in$ in state diagram
- Help simplifying the description of automata
- Every NFA is equivalent to a DFA

Language L = ({ab} ∪ {aba})*



Example

Language L = ({ab} ∪ {aba})*



Regular Expression (regex)

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

Example

RE		Language
0	=>	{ 0 }
01	=>	{ 01 }
0 | 1	=>	{0,1}
0(0|1)	=>	{00,01}
(0|1)(0|1)	=>	{00,01,10,11}
0*	=>	{∈,0,00,000,0000,...}
(0|1)*	=>	{∈,0,1,00,01,10,11,000,001,...}

Example

(i|I)(f|F)

Keyword **if** of language Pascal

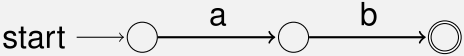
- if
- IF
- If
- iF

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

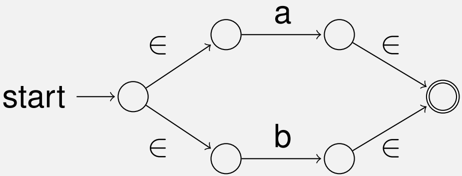
An E followed by a (possibly empty) sequence of digits

- E123
- E9
- E

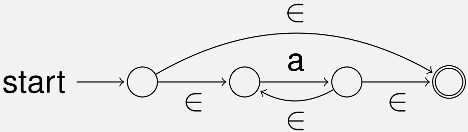
Regular Expression and Finite Automata



ab



a | b



a*

Convenience Notation

- α+ = one or more (i.e. $\alpha\alpha*$)
- α? = 0 or 1 (i.e. $(\alpha| \in)$)
- [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

Example

(0|1|2|3|4) => [0-4]

(a|g|h|m) => [aghm]

(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)* => [0-9]+

(E|e)(+|-|∈)(0|1|2|3|4|5|6|7|8|9)+ => [Ee][+-]?[0-9]+

ANTLR [1]

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

Lexer

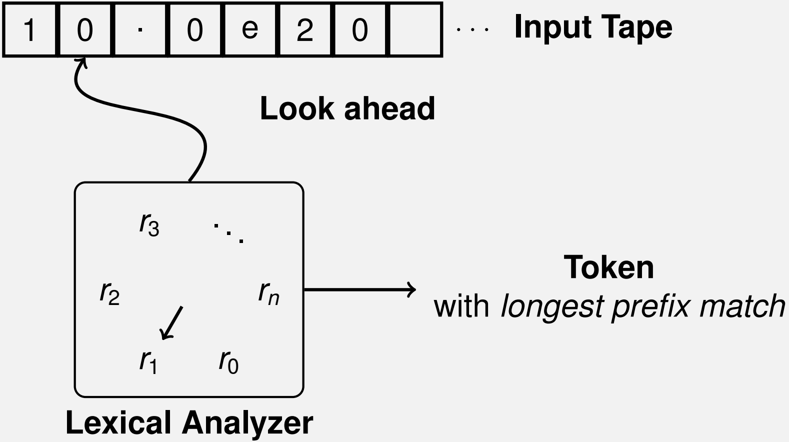
```
/**
 * Filename: Hello.g4
 */
lexer grammar Hello;

// 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 ;
```


1 0 . 0 e 2 0
· · ·
Input Tape
Look ahead
r_3
r_2
r_n
r_1
r_0
Lexical Analyzer
Token
with *longest prefix match*

Summary

- A lexical analyzer is a pattern matcher that isolates small-scale parts of a program
- Lexical rules are represented by Regular expressions or Finite Automata.
- How to write a lexical analyzer (lexer) in ANTLR

References I

[1] ANTLR, http:antlr.org, 19 08 2016.