Lexical Analysis (Scanning)

- Basic Concepts & Regular Expressions
  - What does a Lexical Analyzer do?
  - How does it Work?
  - Formalizing Token Definition & Recognition
- LEX - A Lexical Analyzer Generator

The Role of the Lexical Analyzer

- **Lexical Analyzer** (词法分析器) is the first phase of a compiler, it reads the source program character by character to produce tokens for syntax analysis.
- Normally a lexical analyzer doesn't return a list of tokens at one shot, it returns a token when the parser asks a token from it.
- Lexical analyzer = scanning + lexical analysis

Lexical Analysis

- Lexical analysis: Characters into Tokens
  - Characters → $\boxed{\text{Scanner}}$ → Tokens
    - ∗ but that's not the whole story
  - $\boxed{\text{i}}\ \boxed{:}\boxed{=}\ \boxed{\text{i}}\ \boxed{+}\ \boxed{1}$ → $\boxed{\text{Scanner}}$ → $\boxed{\text{i}}\boxed{:=}\boxed{\text{i}}\boxed{+}\boxed{1}$
    - ∗ also filter whitespace (e.g. blank, tab)
  - $\boxed{(}\boxed{*}\ \boxed{\text{x}}\boxed{\text{z}}\ \boxed{*}\boxed{)}$ → $\boxed{\text{Scanner}}$ →
    - ∗ also filter comments

Lexical Analysis (cont.)

- Reading the source program as a file of characters.
- Dividing the file up into tokens.
- Tokens（单词）: represents a unit of information in the source program.
- Examples: keywords（关键词/保留字）, identifiers（标识符）, arithmetic symbols, multicharacter symbols(>=, <>).
- **Emphasis:**
  - specification and recognition of tokens—Regular expression（正规/正则表达式）
  - how to recognize tokens--Finite automata（有限自动机）
  - design of a lexical analyzer—programming in C or LEX

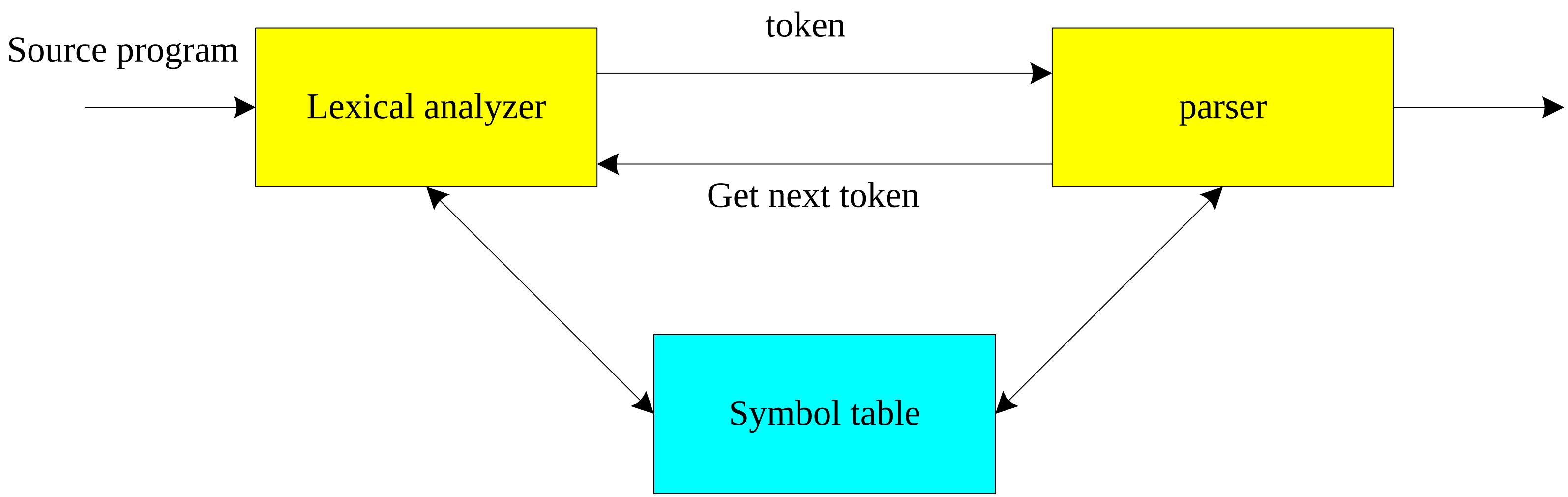


Fig.3.1. interaction of lexical analyzer with parser

What are responsibilities of each box ?

Issues in lexical analysis

- Separation of Lexical Analysis From Parsing Presents a Simpler Conceptual Model
  - From a Software Engineering Perspective Division Emphasizes
    - High Cohesion and Low Coupling
    - Implies Well Specified ⇒ Parallel Implementation
- Separation Increases Compiler Efficiency (I/O Techniques to Enhance Lexical Analysis)
- Separation Promotes Portability（可移植性）.
  - This is critical today, when platforms (OSs and Hardware) are numerous and varied!
  - Emergence of Platform Independence – Java

Lexical Analyzer in Perspective

- **LEXICAL ANALYZER**
  - Scan Input
  - Remove WS, NL, …
  - Identify Tokens
  - Create Symbol Table
  - Insert Tokens into ST
  - Generate Errors
  - Send Tokens to Parser

- **PARSER**
  - Perform Syntax Analysis
  - Actions Dictated by Token Order
  - Update Symbol Table Entries
  - Create Abstract Rep. of Source
  - Generate Errors
  - And More…. (We'll see later)

Introducing Basic Terminology

What are Major Terms for Lexical Analysis?

- **TOKEN**
  - A classification for a common set of strings
  - Examples Include <Identifier>, <number>, etc.
- **PATTERN**
  - The rules which characterize the set of strings for a token
  - Recall File and OS Wildcards ([A-Z]*. *)
- **LEXEME**
  - Actual sequence of characters that matches pattern and is classified by a token
  - Identifiers: x, count, name, etc…

Token, pattern, lexemes

- Since a token can represent more than one lexeme, additional information should be held for that specific lexeme. This additional information is called as the ***attribute*** (属性)of the token.
- For simplicity, a token may have a single attribute which holds the required information for that token.
  - For identifiers, this attribute a pointer to the symbol table, and the symbol table holds the actual attributes for that token.
- Token type and its attribute uniquely identifies a lexeme.
- *Regular expressions* （正规表达式）are widely used to specify patterns.

Token, pattern, lexemes

- Token is a logical unit in the scanner.
- A lexeme is a instance of token.

Token	Sample Lexemes	Informal Description of Pattern
const	const	const
if	if	if
relation	<, <=, =, < >, >, >=	< or <= or = or < > or >= or >
id	pi, count, D2	letter followed by letters and digits
num	3.1416, 0, 6.02E23	any numeric constant
literal	"core dumped"	any characters between " and " except "

Classifies Pattern

Actual values are critical. Info is :

1. Stored in symbol table
2. Returned to parser

Fig.3.2 Examples of tokens.

Attributes for tokens

- An attribute of the token : any value associated to a token
- The lexical analyzer collects information about tokens into their associated attributes.
- The tokens influence parsing decisions; the attributes influence the translation of tokens.
- Usually a token has only a single attribute—a pointer to the symbol table entry in which the information about the token is kept.
- Some attributes: (cf. example 3.1)
  - <id,attr> where attr is pointer to the symbol table
  - <assgop,_> no attribute is needed (if there is only one assignment operator)
  - <num,val> where val is the actual value of the number.

Scanning process

- tokens : the logical units which the scanner generates.
- Tokens fall into several categories:
  - Reserved words: IF, THEN
  - Identifiers: ID
  - operation symbols: PLUS, MINUS
  - Numbers: NUM
  - Character and string: STR
  - Others: {, }, (,), ;, ", ', /, *

Scanning process (cont.)

- Lexeme (string value): the strings of characters represented by a token.
- Tokens $\longleftrightarrow$ one lexeme or many lexemes
- An attribute of the token : any value associated to a token
- Tokens $\longleftrightarrow$ many attributes
- The scanner needs to compute at least as many attributes of a token .

Scanning process (cont.)

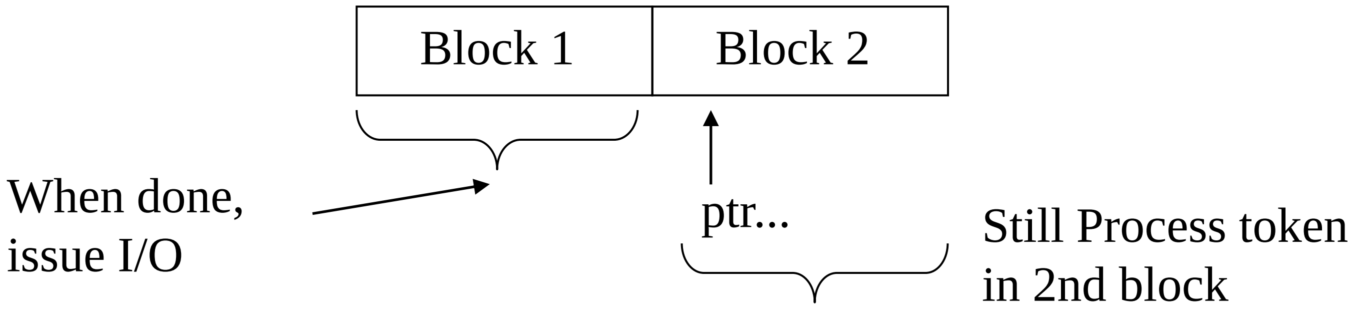
- A token record :

```
Typedef  struct
  {  TokenType  tokenval;
       char  *stringval;
            int numval;
              } TokenRecord
```

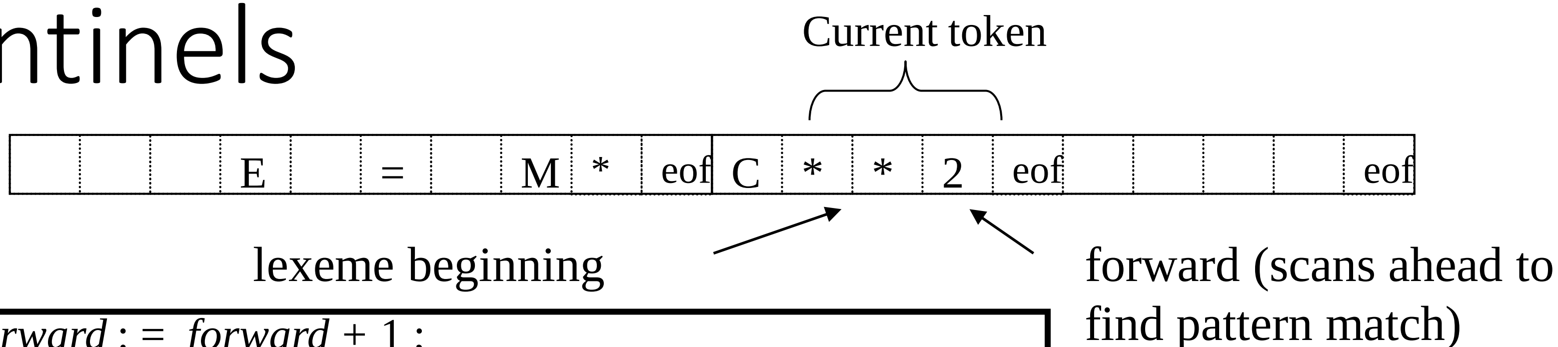
- A more common arrangement: the scanner return the token value only and place the other attributes in variables (such as in LEX and YACC).
- The string of input characters is kept in a buffer or provided by the system input facilities.

Input buffering

- Character-at-a-time I/O (getc, ungetc)
- Block / Buffered I/O
  - Utilize Block of memory
  - Stage data from source to buffer block at a time
  - Maintain two blocks - Why (Recall OS)?
    - Asynchronous I/O - for 1 block
    - While Lexical Analysis on 2nd block



Algorithm: Buffered I/O with Sentinels




```
forward : =  forward + 1 ;
if  forward ↑ = eof  then  begin
    if  forward  at end of first half  then  begin
        reload second half ;  ←  Block I/O
        forward  : =  forward  + 1
    end
else if  forward  at end of second half  then  begin
        reload first half ; ←  Block I/O
        move  forward  to beginning of first half
    end
else  / * eof within buffer signifying end of input  * /
        terminate lexical analysis
end        2nd eof ⇒ no more input !
```

Algorithm performs I/O's. We can still have get & ungetchar Now these work on real memory buffers !

Fig.3.6 Lookahead code with sentinels.

Character-at-a-time I/O

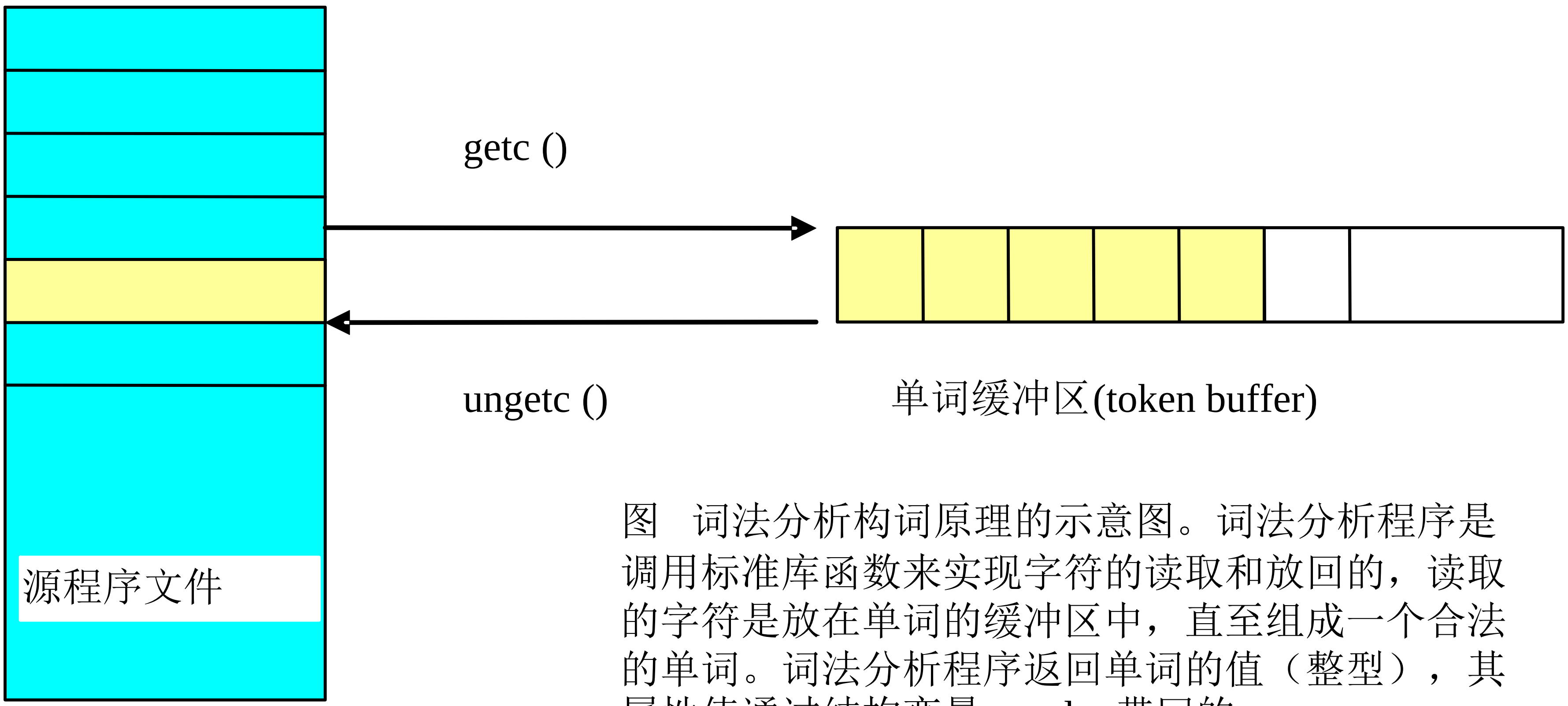


图 词法分析构词原理的示意图。词法分析程序是调用标准库函数来实现字符的读取和放回的，读取的字符是放在单词的缓冲区中，直至组成一个合法的单词。词法分析程序返回单词的值（整型），其属性值通过结构变量yyvalue带回的。

Specification of tokens

Language Concepts :

A language, L, is simply any set of strings over a fixed alphabet.

Alphabet	Languages
{0,1}	{0,10,100,1000,100000…}
	{0,1,00,11,000,111,…}
{a,b,c}	{abc,aabbcc,aaabbbccc,…}
{A, … ,Z}	{TEE,FORE,BALL,…}
	{FOR,WHILE,GOTO,…}
{A,…,Z,a,…,z,0,…9,	{ All legal PASCAL progs}
+,-,…,<,>,…}	{ All grammatically correct English sentences }

Special Languages: ∅ - EMPTY LANGUAGE

{∈} - contains ∈ string only

Terminology of Languages

- **Alphabet** : a finite set of symbols (ASCII characters)
- **String** :
  - Finite sequence of symbols on an alphabet
  - Sentence and word are also used in terms of string
  - ε is the empty string
  - $|s|$ is the length of string s.

Terminology of Languages

EXAMPLES AND OTHER CONCEPTS:

Suppose: S is the string banana

Prefix : ban, banana

Suffix : ana, banana

Substring : nan, ban, ana, banana

Subsequence: bnan, nn

Proper prefix, subfix, or substring *cannot* be all of S

Terminology of Languages (cont.)

- **Language**: sets of strings over some fixed alphabet
  - $\emptyset$ the empty set is a language.
  - $\{\varepsilon\}$ the set containing empty string is a language
  - The set of all possible identifiers is a language.
- **Operators on Strings**:
  - *Concatenation*: xy represents the concatenation of strings x and y. $s\,\varepsilon = s$ $\varepsilon\, s = s$
  - $s^n = s\,s\,s \,..\, s$ (n times) $s^0 = \varepsilon$

Operations on Languages

OPERATION	DEFINITION
union of L and M written $L \cup M$	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
concatenation of L and M written LM	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L written L*	$L^* = \bigcup_{i=0}^{\infty} L^i$ L* denotes "zero or more concatenations of " L
positive closure of L written L^+	$L^+ = \bigcup_{i=1}^{\infty} L^i$ L^+ denotes "one or more concatenations of " L

Fig.3.8 Definitions of operations on languages.

Operations on Languages (Ex3.2)

$L = \{A, B, C, D\}$ $D = \{1, 2, 3\}$

$L \cup D = \{A, B, C, D, 1, 2, 3\}$

$LD = \{A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3\}$

$L^2 = \{AA, AB, AC, AD, BA, BB, BC, BD, CA, \ldots DD\}$

$L^4 = L^2 L^2 =$?? {is the set of all four-letter strings}

$L^* = \{$ All possible strings of L plus $\in \}$

$L^+ = L^* - \in$

$L(L \cup D) =$?? {is the set of strings beginning with a letter followed by a letter or digit}

$L(L \cup D)^* =$?? {is the set of all strings of letters and digits beginning with a letter}

Language & Regular Expressions

- A Regular Expression is a Set of Rules / Techniques for Constructing Sequences of Symbols (Strings) From an Alphabet.

- Let Σ be an Alphabet, r a Regular Expression. Then L(r) is the Language, that is Characterized by the Rules of r.

Regular Expressions

- We use regular expressions to describe tokens of a programming language.
- A regular expression is built up of simpler regular expressions using a set of defining rules.
- Each regular expression r denotes a language $L(r)$.
- A language $L(r)$ denoted by a regular expression r is called as a regular set.

Regular Expressions (cont.)

- a regular expression *r*: is defined by the set of strings that it matches.
- ***L(r)***: language generated by the regular expression.
- **Character set**: be set of the ASCII characters or some subset of it
- ∑: legal symbols, called alphabet.
- A regular repression symbols indicates patterns, metacharacters (metasymbols).
- Backslash and quotes(escape character): turns off the special meaning of a metacharacter.

Rules for Specifying Regular Expressions

fix alphabet Σ

- $\in$ is a regular expression denoting $\{\in\}$
- If a is in Σ, a is a regular expression that denotes {a}
- Let r and s be regular expressions with languages L(r) and L(s). Then
  - $(r) \mid (s)$ is a regular expression $\Rightarrow L(r) \cup L(s)$
  - $(r)(s)$ is a regular expression $\Rightarrow L(r)\, L(s)$
  - $(r)^*$ is a regular expression $\Rightarrow (L(r))^*$
  - (r) is a regular expression $\Rightarrow L(r)$

All are Left-Associative. Parentheses are dropped as allowed by precedence rules.

Rules for Specifying Regular Expressions (cont.)

Regular expressions over alphabet Σ

Reg. Expr	Language it denotes
ε	$\{\varepsilon\}$
$a \in \Sigma$	$\{a\}$
$(r_1) \mid (r_2)$	$L(r_1) \cup L(r_2)$
$(r_1)\ (r_2)$	$L(r_1)\ L(r_2)$
$(r)^*$	$(L(r))^*$
(r)	$L(r)$

- $(r)^+ = (r)(r)^*$ $\quad L(r)(L(r))^*$
- $(r)? = (r) \mid \varepsilon$ $\quad L(r) \cup \{\varepsilon\}$

Regular Expressions (Ex.)

- Ex:
  - $\Sigma = \{0,1\}$
  - $0 \mid 1 \Rightarrow \{0,1\}$
  - $(0 \mid 1)(0 \mid 1) \Rightarrow \{00,01,10,11\}$
  - $0^* \Rightarrow \{\varepsilon ,0,00,000,0000,....\}$
  - $(0 \mid 1)^* \Rightarrow$ all strings with 0 and 1, including the empty string
  -

Regular Expressions (Ex.)

$L = \{A, B, C, D\}$ $D = \{1, 2, 3\}$

$$L = A \mid B \mid C \mid D$$

$$L^2 = (A \mid B \mid C \mid D)(A \mid B \mid C \mid D)$$

$$L^* = (A \mid B \mid C \mid D)^*$$

$$L(L \cup D) = (A \mid B \mid C \mid D)((A \mid B \mid C \mid D) \mid (1 \mid 2 \mid 3))$$

AXIOM	DESCRIPTION
r | s = s | r	| is commutative
r | (s | t) = (r | s) | t	| is associative
(r s) t = r (s t)	concatenation is associative
r (s | t) = r s | r t (s | t) r = s r | t r	concatenation distributes over |
∈r = r r∈ = r	∈ is the identity element for concatenation
r* = (r | ∈)*	relation between * and ∈
r** = r*	* is idempotent

Fig.3.9 Algebraic Properties of Regular Expressions (P.96)

Regular Expressions (Exercise)

- $\Sigma = \{A,\ldots,Z, a,\ldots,z\}$

All Strings that start with "tab" or end with "bat":

tab[A,...,Z,a,...,z]* | [A,...,Z,a,....,z]* bat

- $\Sigma = \{A,\ldots,Z\}$

All Strings in Which Digits 1,2,3 exist in ascending numerical order:

[A,...,Z]* 1 [A,...,Z]* 2 [A,...,Z]* 3 [A,...,Z]*

Regular Definitions

- To write regular expression for some languages can be difficult, because their regular expressions can be quite complex. In those cases, we may use *regular definitions*.
- We can give names to regular expressions, and we can use these names as symbols to define other regular expressions.

- A ***regular definition*** is a sequence of the definitions of the form:

$d_1 \rightarrow r_1$ where d_i is a distinct name and

$d_2 \rightarrow r_2$ r_i is a regular expression over symbols in

. $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$

$d_n \rightarrow r_n$

basic symbols previously defined names

Regular Definitions (cont.)

Regular Definitions: Associate names with Regular Expressions

Example 3.4 : PASCAL IDs

letter $\rightarrow$ A | B | C | … | Z | a | b | … | z

digit $\rightarrow$ 0 | 1 | 2 | … | 9

id $\rightarrow$ letter (letter | digit)*

Shorthand Notation:

- ✓ "+" : one or more $r^* = r^+ \mid \in$ & $r^+ = r\, r^*$
- ✓ "?" : zero or one $r? = r \mid \in$
- ✓ [range] : set range of characters (replaces "|")

[A-Z] = A | B | C | … | Z

Example Using Shorthand : PASCAL IDs

id $\rightarrow$ [A-Za-z][A-Za-z0-9]*

Exercise in class

1. Given: $\sum$={ a,b,c}, describing the set of all strings that contain at most one b with regular expression.

2. Example 3.5:

   Given: letter $\rightarrow$ A | B | … | Z | a | b | … | z

   digit $\rightarrow$ 0 | 1 | … | 9

   Describing identifiers and numbers in Pascal or C with regular expression.

Exercise in class (cont.)

- EX: $\sum = \{ a,b,c \}$

  the set of all strings that contain at most one b.

  $(a|c)^*(b|\varepsilon)(a|c)^*$ $\qquad$ $(a|c)^* | (a|c)^*b(a|c)^*$

  the same language may be generated by many different regular expressions.

- Ex: Identifiers in Pascal or C

  letter → A | B | … | Z | a | b | … | z

  digit → 0 | 1 | … | 9

  id → letter (letter | digit) *

  If we try to write the regular expression representing identifiers without using regular definitions, that regular expression will be complex.

  $(A|\ldots|Z|a|\ldots|z)\ (\ (A|\ldots|Z|a|\ldots|z)\ |\ (0|\ldots|9)\ )^*$

Exercise in class (cont.)

Ex: Unsigned numbers in Pascal or C

$$\text{digit} \rightarrow 0 \mid 1 \mid \ldots \mid 9$$

$$\text{digits} \rightarrow \text{digit}^{+}$$

$$\text{opt-fraction} \rightarrow (\ .\ \text{digits}\ )\ ?$$

$$\text{opt-exponent} \rightarrow (\ \text{E}\ (+\mid-)\ ?\ \ \text{digits}\ )\ ?$$

$$\text{unsigned-num} \rightarrow \text{digits opt-fraction opt-exponent}$$

Review

- Lexical analyzer (scanner) 词法分析器
- Syntax analyzer (parser) 语法分析器
- Token 单词
- Attribute 属性
- Regular expression 正规/正则表达式
- Regular definition 正规定义
- Transition Diagram (TD) 转换图
- Finite Automata (FA) 有限自动机
- Nondeterministic Finite Automata (NFA) 不确定的有限自动机
- Deterministic Finite Automata (DFA) 确定的有限自动机

Review

- Reviewing Finite Automata Concepts
  - Non-Deterministic and Deterministic FA
  - Conversion Process
    - Regular Expressions to NFA
    - NFA to DFA
    - Regular Expressions to DFA
- Relating NFAs/DFAs /Conversion to Lexical Analysis
- Concluding Remarks /Looking Ahead