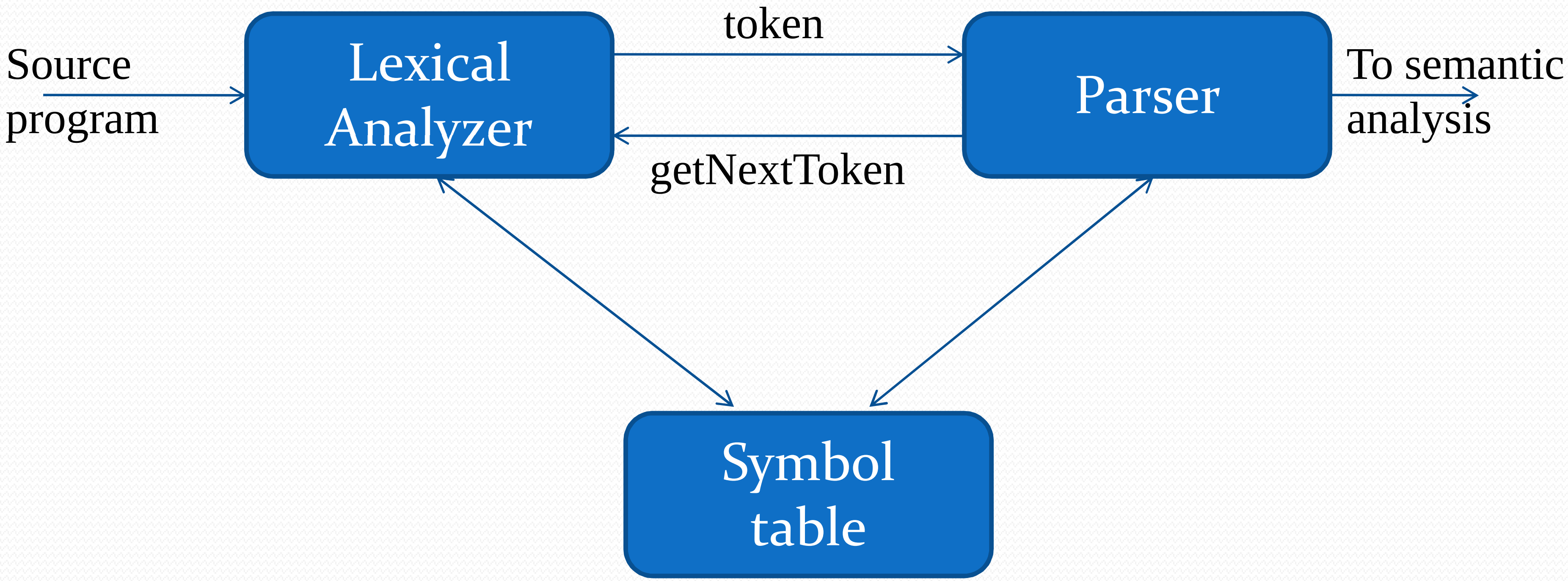
Lexical Analyzer

Introduction

The Role of Lexical Analyzer



Why to separate Lexical analysis and Parsing

1. Simplicity of design

   Removing white space by Lexical Analyzer pays way for easy implementation of parsing

2. Improving compiler efficiency

   Large amount of time is spend in reading the source program. Buffering techniques for reading input characters.

3. Enhancing compiler portability

   The representation of special symbols can be isolated in Lexical Analyzer

Tokens, Patterns and Lexemes

- A **token** is a pair of a token name and an optional token value
- A **pattern** is a description of the form that the lexemes of a token may take
- A **lexeme** is a sequence of characters in the source program that matches the pattern for a token

Example

Token	Informal description	Sample lexemes
if	Characters i, f	if
else	Characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	Letter followed by letter and digits	pi, score, D2
number	Any numeric constant	3.14159, 0, 6.02e23
literal	Anything but " sorrounded by "	"core dumped"

Attributes for tokens

- E = M * C ** 2
  - <id, pointer to symbol table entry for E>
  - <assign-op>
  - <id, pointer to symbol table entry for M>
  - <mult-op>
  - <id, pointer to symbol table entry for C>
  - <exp-op>
  - <number, integer value 2>

Lexical errors

- Some errors are out of power of lexical analyzer to recognize:
  - fi (a == f(x)) ...
- However it may be able to recognize errors like:
  - d = 2r
- Such errors are recognized when no pattern for tokens matches a character sequence

Error recovery

- Panic mode: successive characters are ignored until we reach to a well formed token
- Delete one character from the remaining input
- Insert a missing character into the remaining input
- Replace a character by another character
- Transpose two adjacent characters

Specification of Tokens

Specification of tokens

- Alphabet or Character Class
  - Σ is a finite set of symbols (characters)
  - {0,1} is a binary alphabet
- String or Sentence or word
  - A *string* *s* is a finite sequence of symbols from Σ
    - |*s*| denotes the length of string *s*
    - ε denotes the empty string, thus |ε| = 0
    - banana → |banana|=6
- Language
  - A *language* is a specific set of strings over some fixed alphabet Σ
  - Σ={0,1}
  - L={0,1,00,11,01,10,000,001,010,011,...}

Specification of tokens Cont…

- **Prefix of s**
  - A string obtained by removing 0 or more trailing symbols of s
  - b, ba, ban, bana, banan, banana
- **Suffix of s**
  - A string formed by deleting 0 or more leading symbols of s
  - a, na, ana, nana …
- **Substring of s**
  - A string obtained by removing the suffix and prefix from s
  - ana, nan etc
- **Proper prefix and Proper Suffix**
  - Any prefix or suffix other than the string itself
  - b, ba, a, nana …
- **Subsequence of s**
  - Any string formed by deleting zero or more not necessarily contiguous symbols from s.
  - baaa, ann…

Language Operations

- *Union*

$$L \cup M = \{s \mid s \in L \text{ or } s \in M\}$$

- *Concatenation*

$$LM = \{xy \mid x \in L \text{ and } y \in M\}$$

- *Kleene closure*

$$L^* = \cup_{i=0,\ldots,\infty} L^i$$

- *Positive closure*

$$L^+ = \cup_{i=1,\ldots,\infty} L^i$$

Regular Expressions

<u>Rules for Regular Expression</u>

- ε is a regular expression, $L(\varepsilon) = \{\varepsilon\}$
- If a is a symbol in Σ then a is a regular expression, $L(a) = \{a\}$
- $(r) \mid (s)$ is a regular expression denoting the language $L(r) \cup L(s)$
- $(r)(s)$ is a regular expression denoting the language $L(r)L(s)$
- $(r)^*$ is a regular expression denoting $(L(r))^*$
- (r) is a regular expression denoting $L(r)$

Ex : Identifier → letter (letter |digit) *

Language for Regular Expression → Regular Set

Precedence

* (Closure) has the higher precedence

. (Concatenation) has the next higher precedence

| (Union) has the least precedence

Remove unnecessary parentheses

(a)|((b)*c) →a | b * c

Σ={a,b}

RE a|b {a,b}

(a/b)(a/b) {aa,ab,ba,bb}

(a/b)* ? a/a*b ?

If 2 r.e ***r*** and ***s*** denote the same language then ***r*** and ***s*** are said to be **equivalent** ie. ***r=s*** ex. ***a/b = b/a***

Regular definitions

- Regular definitions introduce a naming convention:

  $d_1 \to r_1$

  $d_2 \to r_2$

  …

  $d_n \to r_n$

  where each r_i is a regular expression over

  $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$
- Any d_j in r_i can be textually substituted in r_i to obtain an equivalent set of definitions

Regular definitions Cont…

- Example:

$$\textbf{letter} \rightarrow \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z}$$
$$\textbf{digit} \rightarrow \texttt{0} \mid \texttt{1} \mid \ldots \mid \texttt{9}$$
$$\textbf{id} \rightarrow \textbf{letter} \; (\, \textbf{letter} \mid \textbf{digit} \,)^*$$

- Regular definitions are not recursive:

$$\textbf{digits} \rightarrow \textbf{digit digits} \mid \textbf{digit} \qquad \textit{wrong!}$$

Notational Shorthand

- One or more instances: (r)+
- Zero of one instances: r?
- Character classes: [abc]

$$r^{+} = rr^{*}$$
$$r? = r \mid \varepsilon$$
$$[\mathbf{a}\text{-}\mathbf{z}] = \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}$$

Notational Shorthand

- letter_ -> [A-Za-z_]
- digit -> [0-9]
- id -> letter_(letter|digit)*

- Examples:
  digit $\rightarrow$ [**0**-**9**]
  num $\rightarrow$ **digit**$^{+}$ (**.** **digit**$^{+}$)? (**E** (**+** | **-**)? **digit**$^{+}$)?

Recognition of tokens

- Starting point is the language grammar to understand the tokens:

stmt -> **if** expr **then** stmt

| **if** expr **then** stmt **else** stmt

| ε

expr -> term **relop** term

| term

term -> **id**

| **number**

Recognition of tokens (cont.)

- The next step is to formalize the patterns:

  digit -> [0-9]

  digits -> digit+

  number -> digit(.digits)? (E[+-]? Digit)?

  letter -> [A-Za-z_]

  id -> letter (letter|digit)*

  if -> if

  then -> then

  else -> else

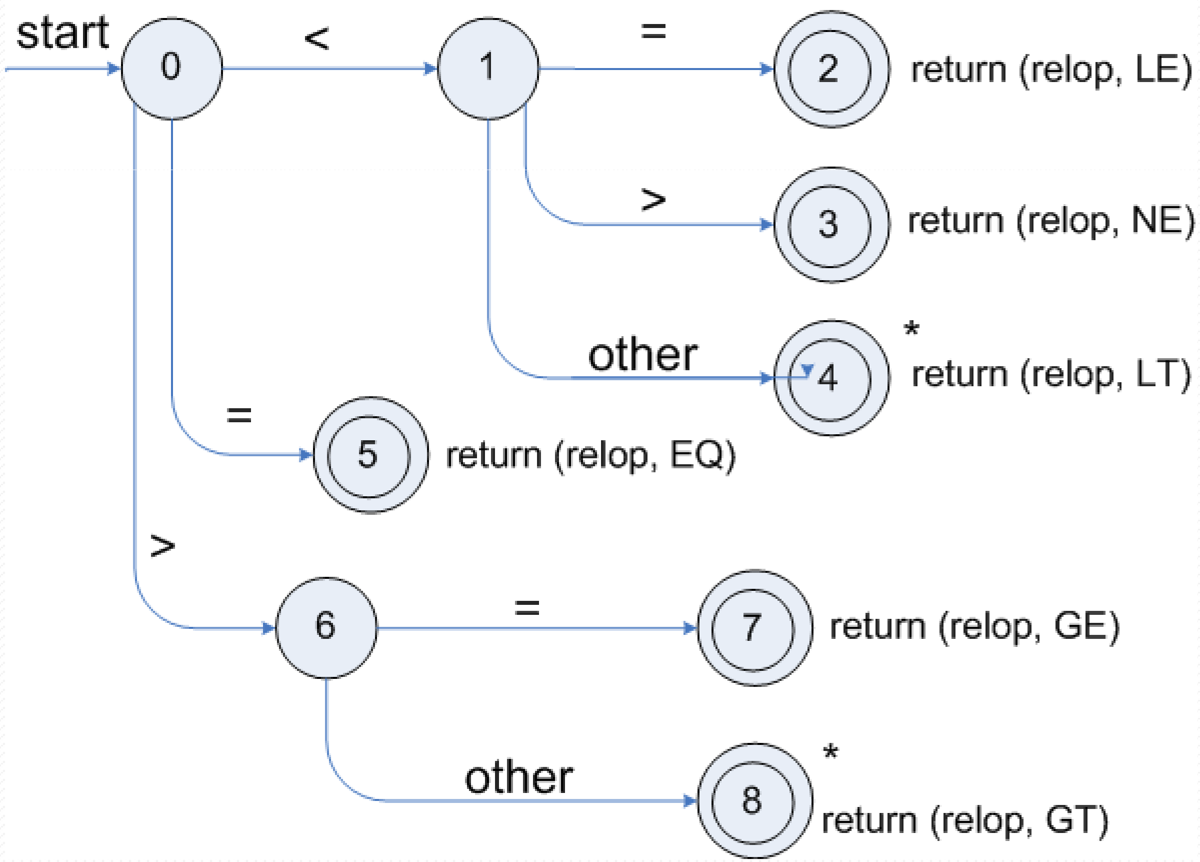
  Relop -> < | > | <= | >= | = | <>

- We also need to handle whitespaces:

  ws -> (blank | tab | newline)+

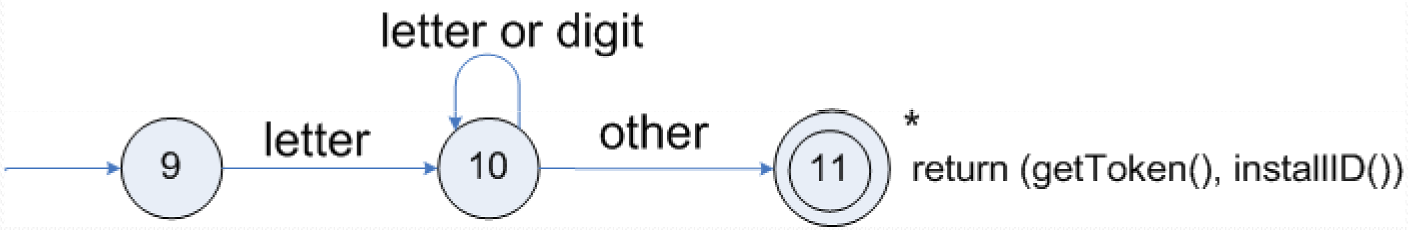
Transition diagrams

- Transition diagram for relop



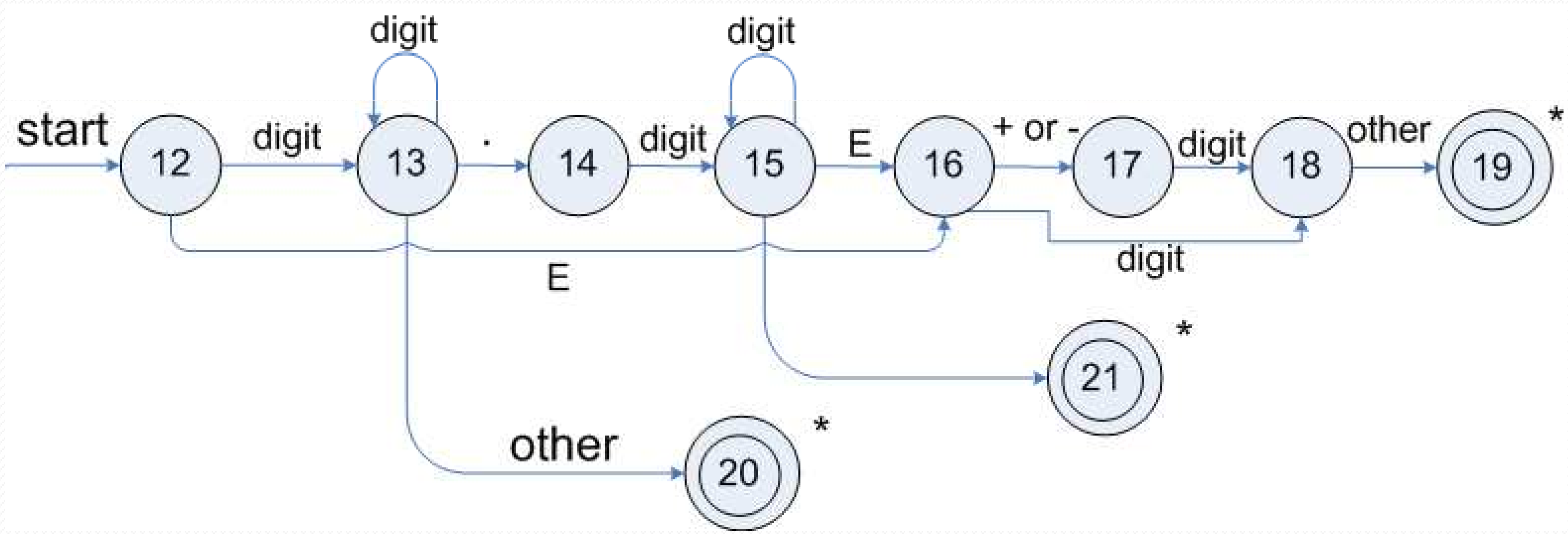
Transition diagrams (cont...)

- Transition diagram for reserved words and identifiers



Transition diagrams (cont.)

- Transition diagram for unsigned numbers



Transition diagrams (cont.)

- Transition diagram for whitespace