Compiler Design

Fatemeh Deldar

Isfahan University of Technology

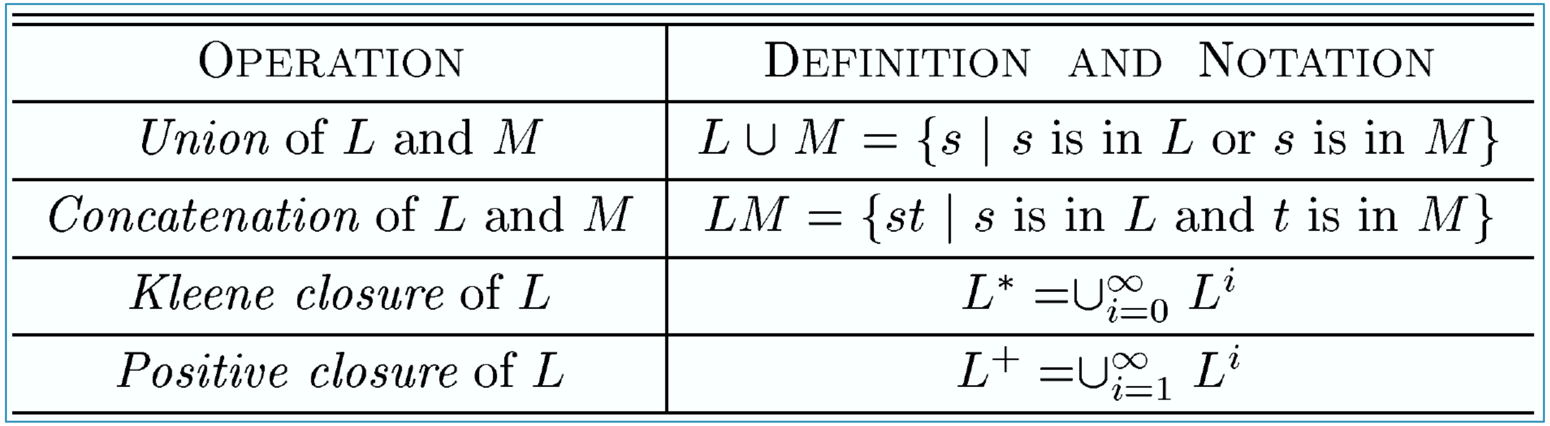
1403-1404

Specification of Tokens

- **Regular expressions** are an important notation for specifying lexeme patterns
- **Strings and Languages**
  - An **alphabet** is any finite set of symbols
    - Letters, digits, and punctuation
  - A **string** over an alphabet is a finite sequence of symbols drawn from that alphabet
    - The terms "sentence" and "word" are often used as synonyms for "string"
  - A **language** is any countable set of strings over some fixed alphabet

Operations on Languages

- In lexical analysis, the most important operations on languages are **union**, **concatenation**, and **closure**



OPERATION	DEFINITION AND NOTATION
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

- **Example**
  - Let L be the set of letters {A, B, …, Z, a, b, …, z} and let D be the set of digits {0, 1, …, 9}
    - $L \cup D$, LD, L^4, L^*, $L(L \cup D)^*$, D^+

Regular Expressions

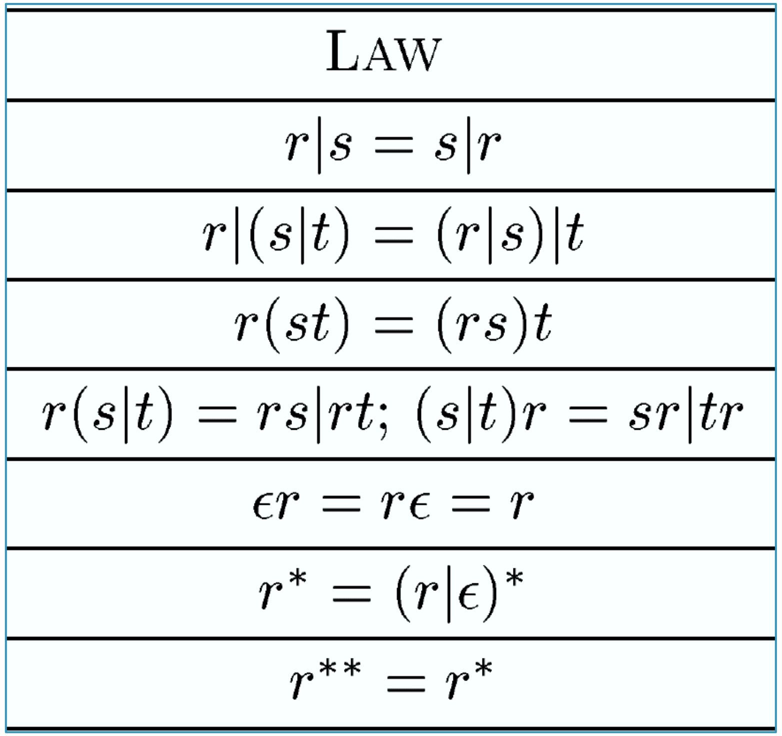
- **Basis**
  - ϵ is a regular expression, and $L(\epsilon)$ is $\{\epsilon\}$
  - If a is a symbol in $\sum$, then $\boldsymbol{a}$ is a regular expression, and $L(\boldsymbol{a}) = \{a\}$
- **Induction**
  1. $(r)|(s)$ is a regular expression denoting the language $L(r) \cup L(s)$
  2. $(r)(s)$ is a regular expression denoting the language $L(r)L(s)$
  3. $(r)^*$ is a regular expression denoting $(L(r))^*$
  4. (r) is a regular expression denoting $L(r)$

Regular Expressions

- **Precedences**
  - The unary operator * has highest precedence
  - Concatenation has second highest precedence
  - | has lowest precedence
- **Example**
  - $\sum = \{a, b\}$
  - $(a|b)(a|b) \rightarrow \{aa, ab, ba, bb\}$
- A language that can be defined by a regular expression is called a regular set

Regular Expressions

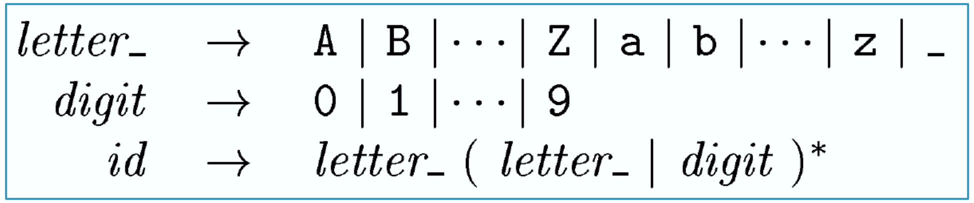
- Algebraic laws for regular expressions



LAW
$r\|s = s\|r$
$r\|(s\|t) = (r\|s)\|t$
$r(st) = (rs)t$
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$
$\epsilon r = r\epsilon = r$
$r^* = (r\|\epsilon)^*$
$r^{**} = r^*$

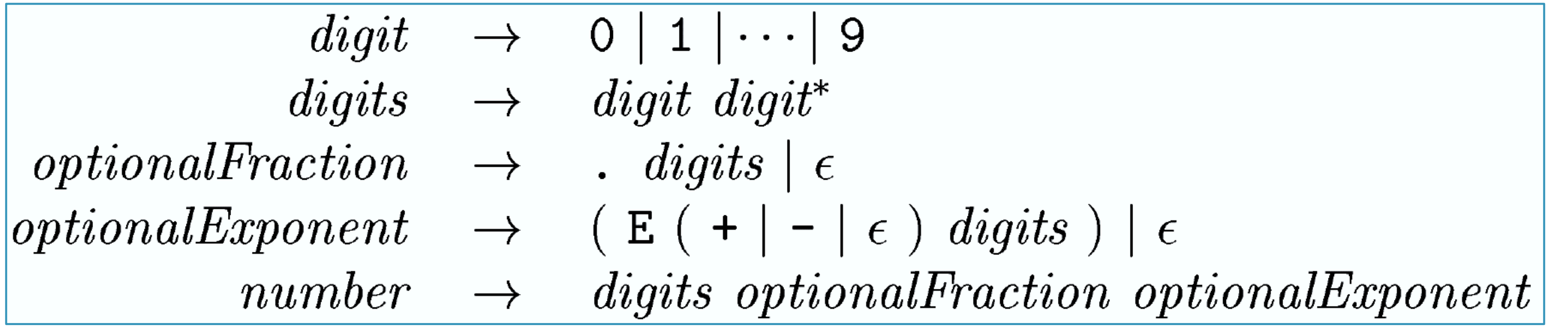
Regular Expressions

- **Example**
  - A regular definition for the language of C identifiers



$$
\begin{aligned}
\mathit{letter\_} &\rightarrow \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \\
\mathit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\mathit{id} &\rightarrow \mathit{letter\_} \; (\; \mathit{letter\_} \mid \mathit{digit} \;)^*
\end{aligned}
$$

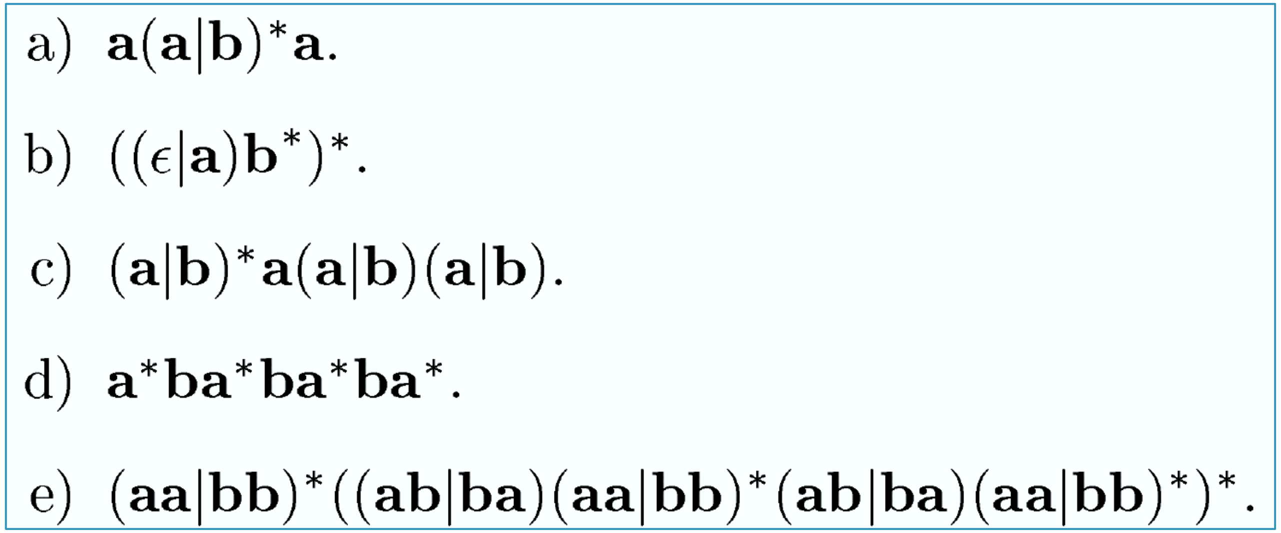
  - A regular definition for unsigned numbers (such as 5280, 0.034, 6.36E4, or 1.89E-4)



$$
\begin{aligned}
\mathit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\mathit{digits} &\rightarrow \mathit{digit} \; \mathit{digit}^* \\
\mathit{optionalFraction} &\rightarrow \texttt{.} \; \mathit{digits} \mid \epsilon \\
\mathit{optionalExponent} &\rightarrow (\; \texttt{E} \; (\; \texttt{+} \mid \texttt{-} \mid \epsilon \;) \; \mathit{digits} \;) \mid \epsilon \\
\mathit{number} &\rightarrow \mathit{digits} \; \mathit{optionalFraction} \; \mathit{optionalExponent}
\end{aligned}
$$

Regular Expressions

- **Example**
  - Describe the languages denoted by the following regular expressions:



a) $\mathbf{a}(\mathbf{a}|\mathbf{b})^*\mathbf{a}$.

b) $((\epsilon|\mathbf{a})\mathbf{b}^*)^*$.

c) $(\mathbf{a}|\mathbf{b})^*\mathbf{a}(\mathbf{a}|\mathbf{b})(\mathbf{a}|\mathbf{b})$.

d) $\mathbf{a}^*\mathbf{b}\mathbf{a}^*\mathbf{b}\mathbf{a}^*\mathbf{b}\mathbf{a}^*$.

e) $(\mathbf{aa}|\mathbf{bb})^*((\mathbf{ab}|\mathbf{ba})(\mathbf{aa}|\mathbf{bb})^*(\mathbf{ab}|\mathbf{ba})(\mathbf{aa}|\mathbf{bb})^*)^*$.

Recognition of Tokens

- **Example**
  - Patterns for tokens

$$\begin{array}{rcl}
\mathit{digit} & \to & [\texttt{0-9}] \\
\mathit{digits} & \to & \mathit{digit}^{+} \\
\mathit{number} & \to & \mathit{digits}\ (\texttt{.}\ \ \mathit{digits})?\ \ (\ \texttt{E}\ [\texttt{+-}]?\ \ \mathit{digits}\ )? \\
\mathit{letter} & \to & [\texttt{A-Za-z}] \\
\mathit{id} & \to & \mathit{letter}\ (\ \mathit{letter} \mid \mathit{digit}\ )^{*} \\
\mathit{if} & \to & \texttt{if} \\
\mathit{then} & \to & \texttt{then} \\
\mathit{else} & \to & \texttt{else} \\
\mathit{relop} & \to & \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{=} \mid \texttt{<>}
\end{array}$$

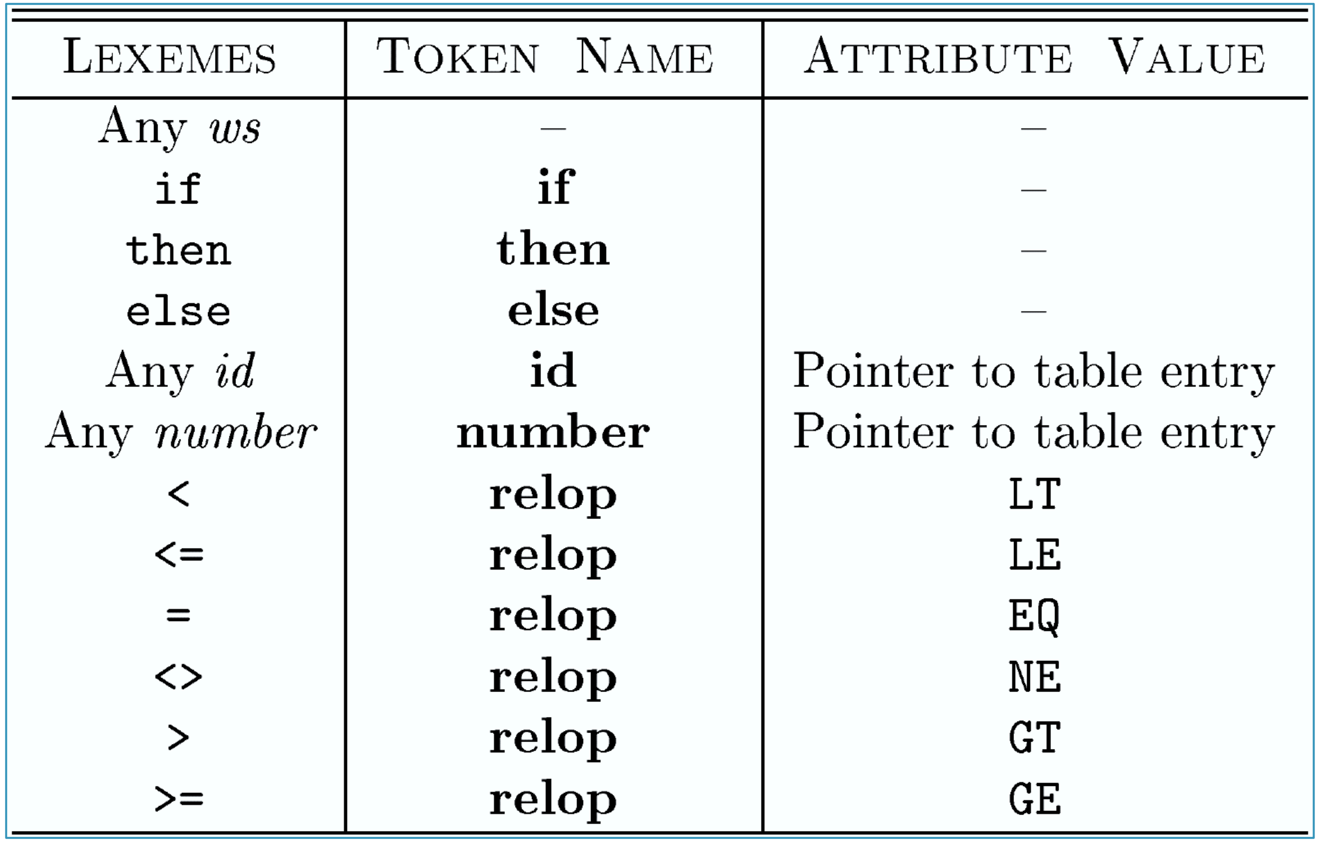
operator ? means:
zero or one occurrence

- Assign the lexical analyzer the job of stripping out white-space

$$\mathit{ws} \to (\ \textbf{blank} \mid \textbf{tab} \mid \textbf{newline}\ )^{+}$$

Recognition of Tokens

- **Example**
  - Tokens, their patterns, and attribute values



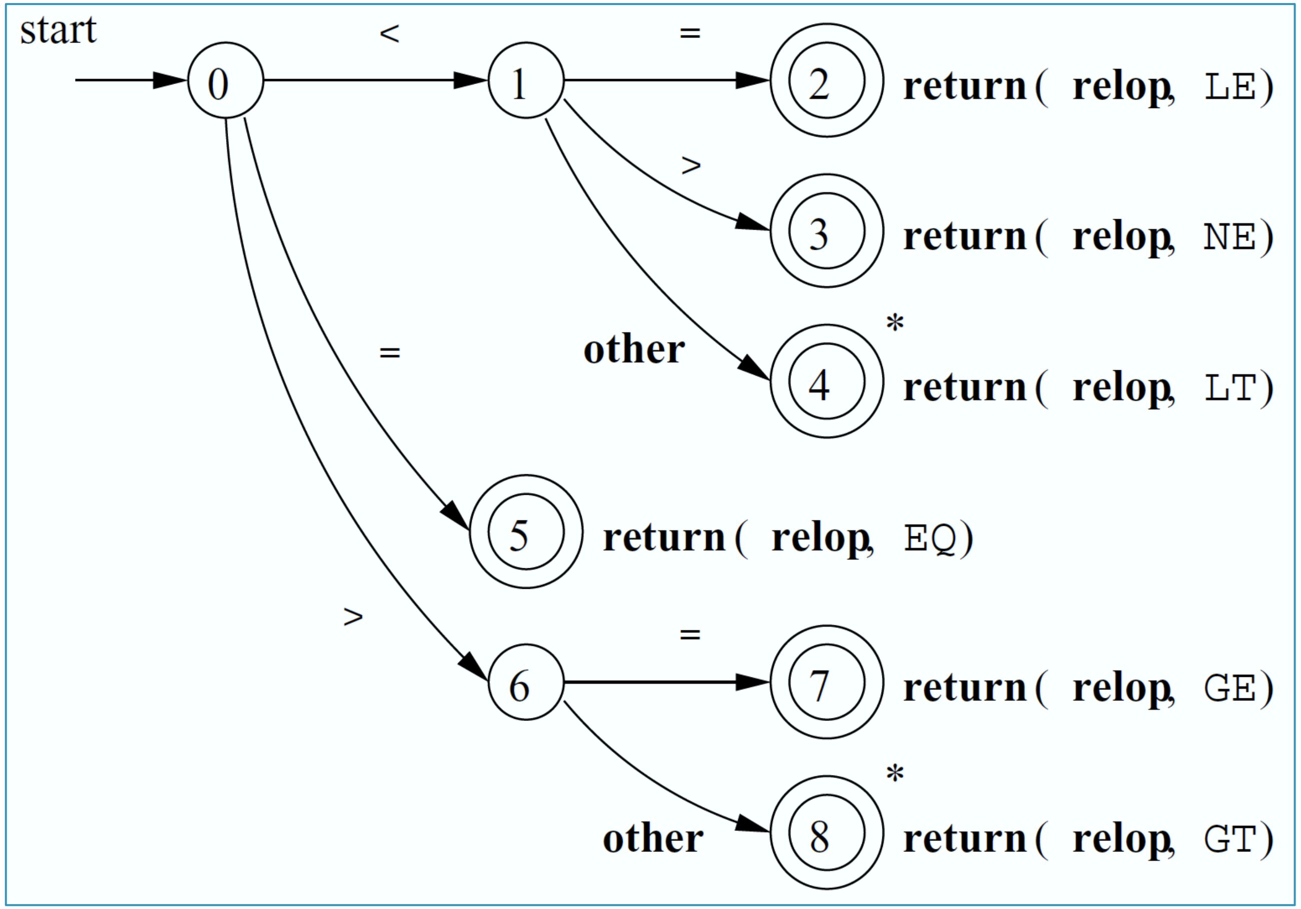
LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**number**	Pointer to table entry
`<`	**relop**	`LT`
`<=`	**relop**	`LE`
`=`	**relop**	`EQ`
`<>`	**relop**	`NE`
`>`	**relop**	`GT`
`>=`	**relop**	`GE`

Transition Diagrams

- Transition diagrams have a collection of **nodes** or circles, called **states**
- **Edges** are directed from one state of the transition diagram to another
  - **Accepting or final states** indicate that a lexeme has been found
  - **If it is necessary to retract the forward pointer one position, a * is placed near that accepting state**
  - One state is designated the start state, or **initial state**

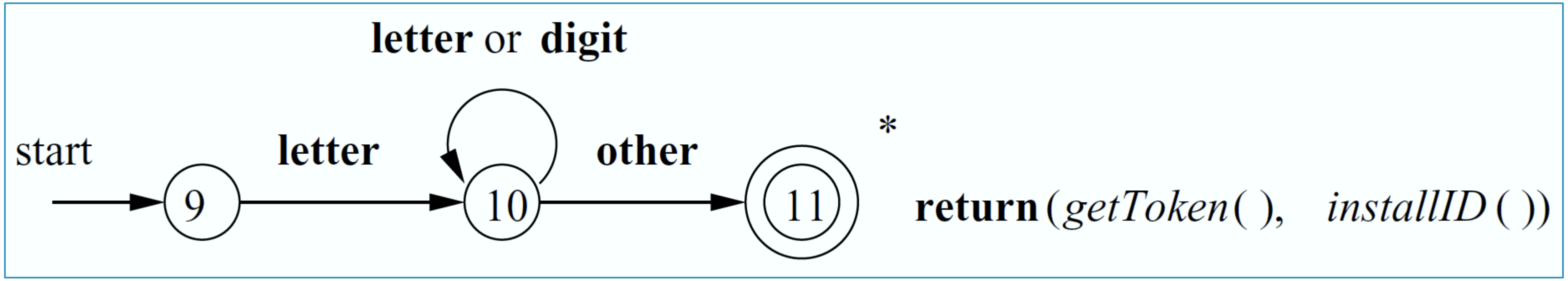
Transition Diagrams

- **Example:** A transition diagram that recognizes the lexemes matching the token relop



Transition Diagrams

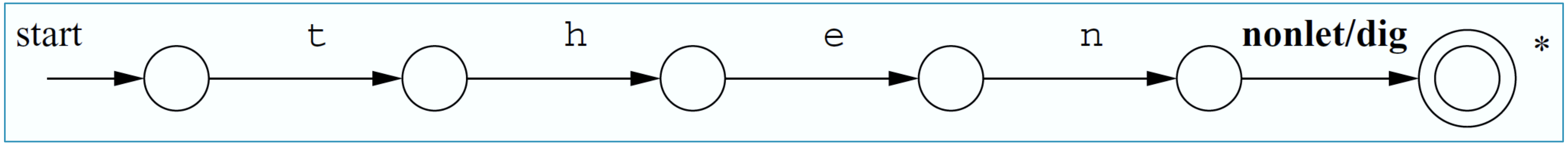
- **Example:** Recognition of Reserved Words and Identifiers



Transition Diagrams

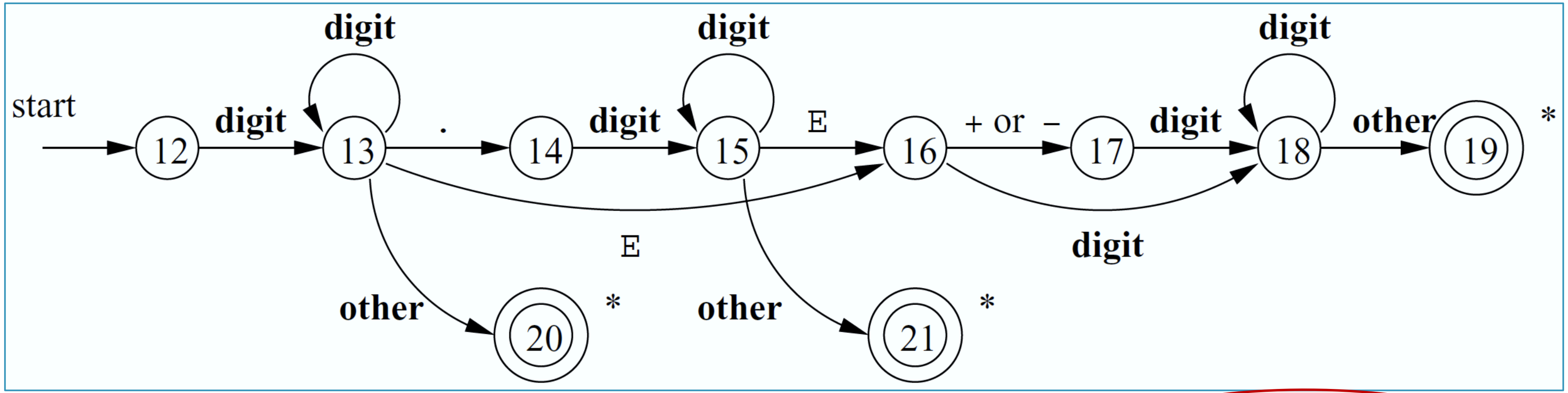
- There are two ways that we can handle reserved words that look like identifiers:

  1. ***Install the reserved words in the symbol table initially***
     - When an identifier is found, a call to **installID** places it in the symbol table if it is not already there
  2. ***Create separate transition diagrams for each keyword***
     - In this method we must prioritize the tokens so that the reserved-word tokens are recognized in preference to id

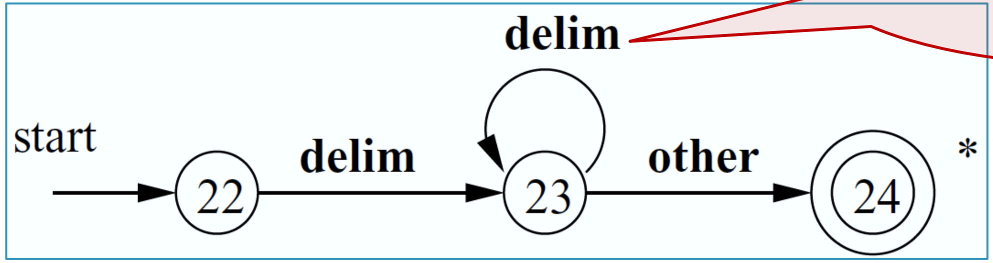


Transition Diagrams

- **Example:** The transition diagram for token number



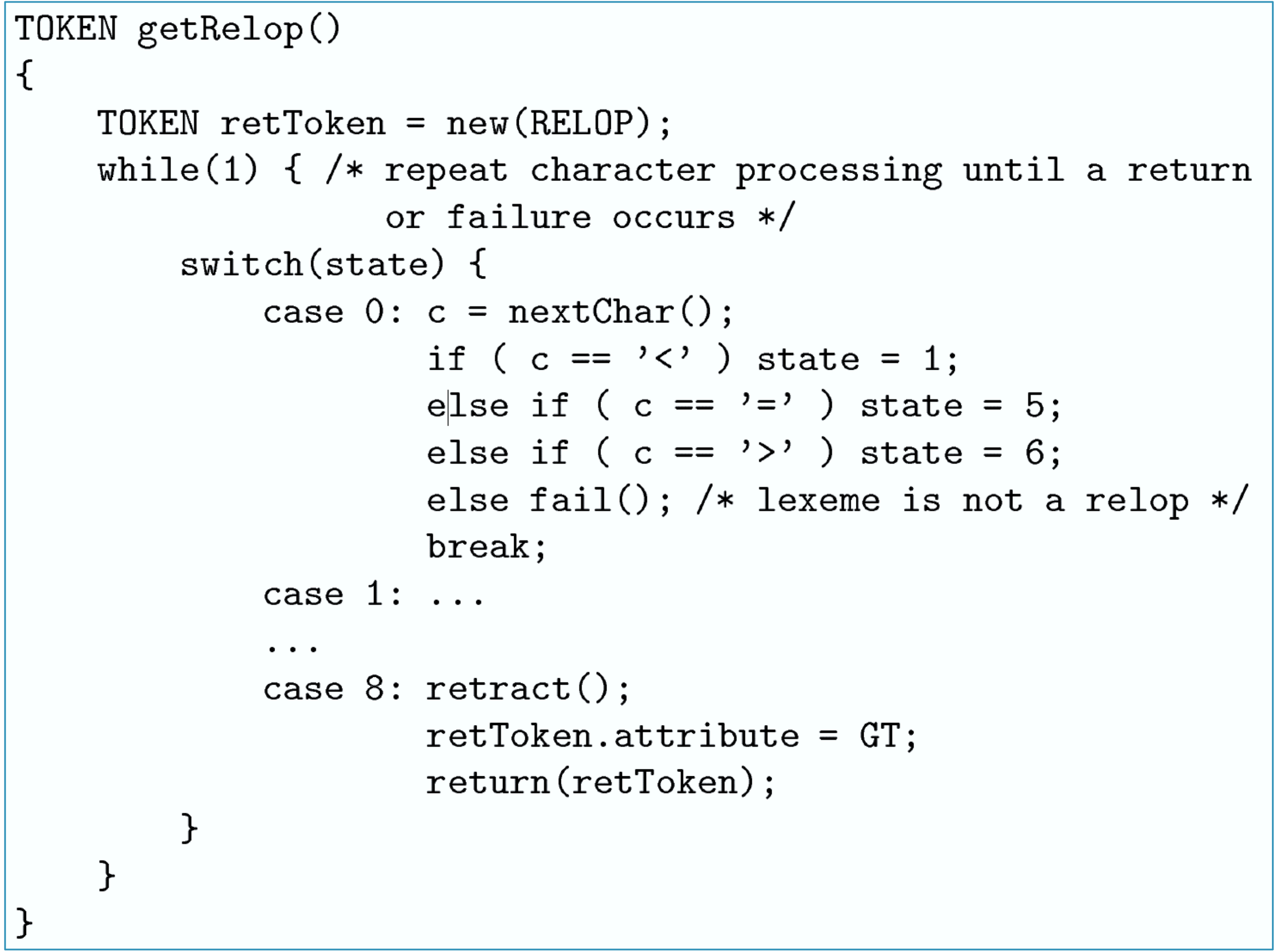
- **Example:** The transition diagram for whitespace



"delim" means any whitespace characters

Transition Diagrams

- **Sketch of implementation of relop transition diagram**
- **Architecture of a Transition-Diagram-Based Lexical Analyzer**
    - Combining all the transition diagrams into one (Combining states 0, 9, 12, and 22 into one start state)



```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                  or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == ’<’ ) state = 1;
                    else if ( c == ’=’ ) state = 5;
                    else if ( c == ’>’ ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```