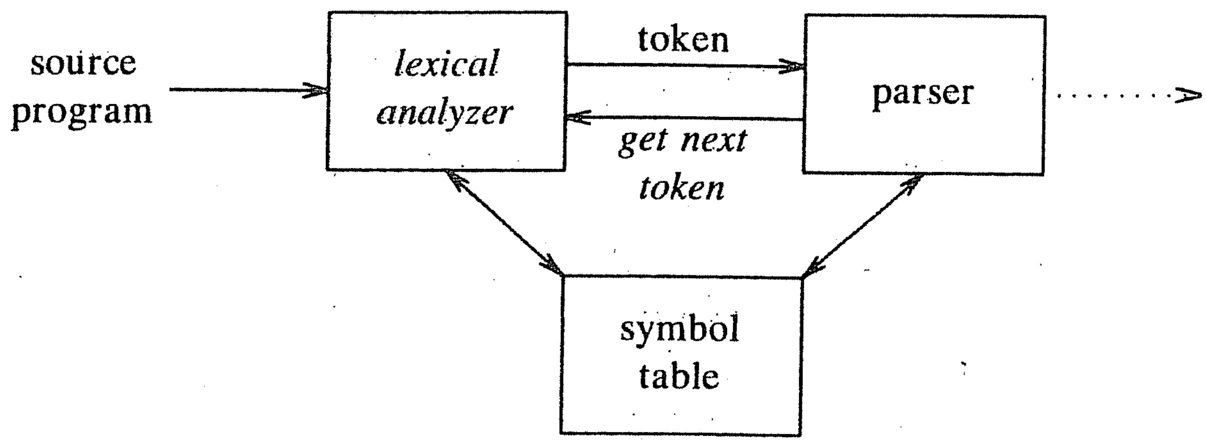

Fig. 3.1. Interaction of lexical analyzer with parser.

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 letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Figure 3.2: Examples of tokens

TERM	DEFINITION
prefix of s	A string obtained by removing zero or more trailing symbols of string s; e.g., **ban** is a prefix of **banana**.
suffix of s	A string formed by deleting zero or more of the leading symbols of s; e.g., **nana** is a suffix of **banana**.
substring of s	A string obtained by deleting a prefix and a suffix from s; e.g., **nan** is a substring of **banana**. Every prefix and every suffix of s is a substring of s, but not every substring of s is a prefix or a suffix of s. For every string s, both s and ϵ are prefixes, suffixes, and substrings of s.
proper prefix, suffix, or substring of s	Any nonempty string x that is, respectively, a prefix, suffix, or substring of s such that $s \neq x$.
subsequence of s	Any string formed by deleting zero or more not necessarily contiguous symbols from s; e.g., **baaa** is a subsequence of **banana**.

Terms for parts of a string.

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

Figure 3.6: Definitions of operations on languages

AXIOM	DESCRIPTION
$r \mid s = s \mid r$	$\mid$ is commutative
$r \mid (s \mid t) = (r \mid s) \mid t$	$\mid$ is associative
$(rs)t = r(st)$	concatenation is associative
$r(s \mid t) = rs \mid rt$ $(s \mid t)r = sr \mid tr$	concatenation distributes over $\mid$
$\epsilon r = r$ $r\epsilon = r$	ϵ is the identity element for concatenation
$r^* = (r \mid \epsilon)^*$	relation between * and ϵ
$r^{**} = r^*$	* is idempotent

Fig. 3.7. Algebraic properties of regular expressions.

<table>
<tr><th>EXPRESSION</th><th>MATCHES</th><th>EXAMPLE</th></tr>
<tr><td>c</td><td>the one non-operator character c</td><td>a</td></tr>
<tr><td>$\backslash c$</td><td>character c literally</td><td>*</td></tr>
<tr><td>"s"</td><td>string s literally</td><td>"**"</td></tr>
<tr><td>.</td><td>any character but newline</td><td>a.*b</td></tr>
<tr><td>^</td><td>beginning of a line</td><td>^abc</td></tr>
<tr><td>$</td><td>end of a line</td><td>abc$</td></tr>
<tr><td>[s]</td><td>any one of the characters in string s</td><td>[abc]</td></tr>
<tr><td>[^s]</td><td>any one character not in string s</td><td>[^abc]</td></tr>
<tr><td>$r*$</td><td>zero or more strings matching r</td><td>a*</td></tr>
<tr><td>$r+$</td><td>one or more strings matching r</td><td>a+</td></tr>
<tr><td>$r?$</td><td>zero or one r</td><td>a?</td></tr>
<tr><td>$r\{m,n\}$</td><td>between m and n occurrences of r</td><td>a[1,5]</td></tr>
<tr><td>r_1r_2</td><td>an r_1 followed by an r_2</td><td>ab</td></tr>
<tr><td>$r_1 \mid r_2$</td><td>an r_1 or an r_2</td><td>a|b</td></tr>
<tr><td>(r)</td><td>same as r</td><td>(a|b)</td></tr>
<tr><td>r_1/r_2</td><td>r_1 when followed by r_2</td><td>abc/123</td></tr>
</table>

Figure 3.8: Lex regular expressions

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

Figure 3.12: Tokens, their patterns, and attribute values

```
%{
    /* definitions of manifest constants
    LT, LE, EQ, NE, GT, GE,
    IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* regular definitions */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?

%%

{ws}        {/* no action and no return */}
if          {return(IF);}
then        {return(THEN);}
else        {return(ELSE);}
{id}        {yylval = install_id(); return(ID);}
{number}    {yylval = install_num(); return(NUMBER);}
"<"         {yylval = LT; return(RELOP);}
"<="        {yylval = LE; return(RELOP);}
"="         {yylval = EQ; return(RELOP);}
"<>"        {yylval = NE; return(RELOP);}
">"         {yylval = GT; return(RELOP);}
">="        {yylval = GE; return(RELOP);}

%%

install_id() {
    /* procedure to install the lexeme, whose
    first character is pointed to by yytext and
    whose length is yyleng, into the symbol table
    and return a pointer thereto */
}

install_num() {
    /* similar procedure to install a lexeme that
    is a number */
}
```

Fig. 3.23. Lex program for the tokens of Fig. 3.12.

Operation	Description
ϵ-*closure*(s)	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-*closure*(T)	Set of NFA states reachable from some NFA state s in T on ϵ-transitions alone.
move(T, a)	Set of NFA states to which there is a transition on input symbol a from some NFA state s in T.

Fig. 3.31. Operations on NFA states.

```
initially, ε-closure(s0) is the only state in Dstates, and it is unmarked;
while ( there is an unmarked state T in Dstates ) {
    mark T;
    for ( each input symbol a ) {
        U = ε-closure(move(T, a));
        if ( U is not in Dstates )
            add U as an unmarked state to Dstates;
        Dtran[T, a] = U;
    }
}
```

Figure 3.32: The subset construction

push all states of T onto *stack*;
initialize ϵ-*closure*(T) to T;
while (*stack* is not empty) {
    pop t, the top element, off *stack*;
    for (each state u with an edge from t to u labeled ϵ)
        if (u is not in ϵ-*closure*(T)) {
            add u to ϵ-*closure*(T);
            push u onto *stack*;
        }
}

Figure 3.33: Computing ϵ-*closure*(T)

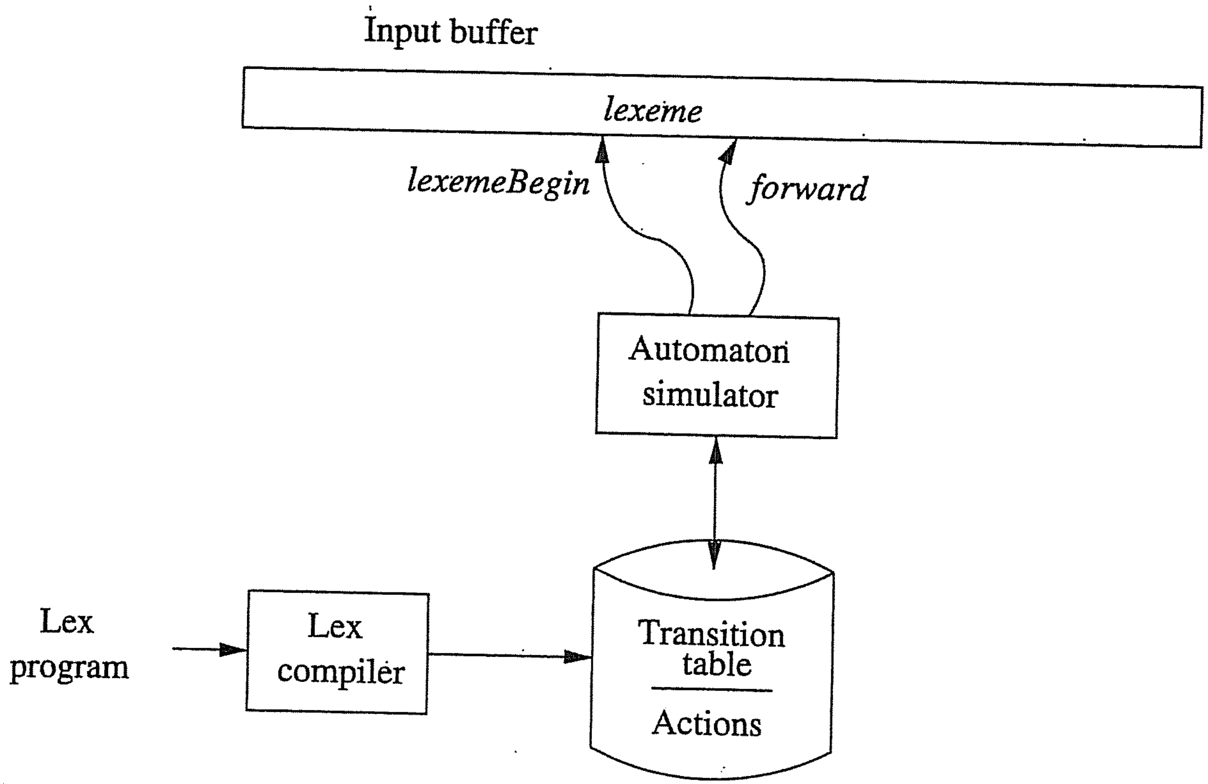

Figure 3.49: A Lex program is turned into a transition table and actions, which are used by a finite-automaton simulator