Compiler Design

Samit Biswas

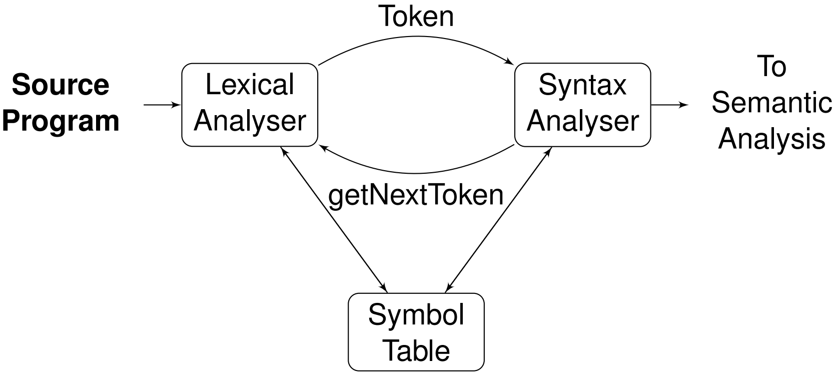
samit@cs.iiests.ac.in

Department of Computer Science and Technology,
Indian Institute of Engineering Science and Technology, Shibpur

July 21, 2018

Lexical Analyser
- Tokens, Patterns and Lexemes
- Attributes of Tokens
- Specification of Tokens

Lexical Analyser



Lexical Analyser

- Main task is to read the input characters and produce as output a sequence of tokens.
- Stripping from the source program comments and white space in the form of blank, tab and newline characters.
- Correlating error messages from the compiler with the same source program

Tokens, Patterns and Lexemes

- A token is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier.
- This set of strings is described by a rule called pattern associated with that token. The pattern is said to match each string in the set.
- A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyser as an instance of that token. These are smallest logical unit (words) of a program such as $A, B, 1.0, true, +, <=$ ....

Examples - Tokens, Patterns and Lexemes

Consider The Following C Statement:
printf (“Total = %d”, score) ;

- **printf** and **score** are lexemes matching the pattern for token id
- **“Total = %d”** is a lexeme matching literal.

Examples - Tokens, Patterns and Lexemes

Token	**Sample lexemes**	**Informal Description of Pattern**
if	if	characters i,f
else	else	characters e,l,s,e
comparison	$<, <=, ==, !=, >, >=$	$<$ *or* $>$ *or* $<=$ *or* $>=$ *or* $==$ *or*$!=$
id	*pi*, *score*, *D2*	Letters followed by letters and digit.
number	3.14159, 0, 6.02e23	any numeric constant
literal	“Total = %d”	Total = %d

Attributes of Tokens

- The token names and associated attribute values for the Fortran statement are written below as a sequence of pairs.
  $E = M * C * *2$

<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 >

Specification of Tokens

- Regular Expression.
- Deterministic FiniteAutomata.
- Non-Deterministic Finite Automata.
- Non-Deterministic Finite Automata with empty transitions.

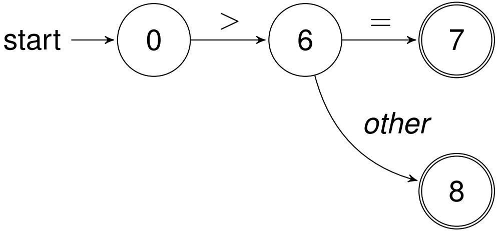
Recognitions of Tokens

Regular-Expression Pattern

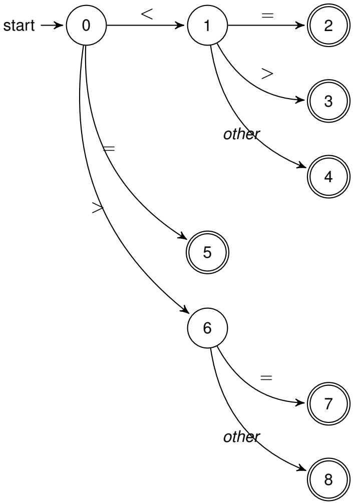
Regular Expression	Token	Attribute Value
WS	–	–
if	if	–
then	then	–
else	else	–
id	id	pointer to table entry
num	num	pointer to table entry
$<$	relop	LT
$<=$	relop	LE
$=$	relop	EQ
$<>$	relop	NE
$>$	relop	GT
$>=$	relop	GE

Construct a lexical analyser that will isolate the lexeme for the next token in the input buffer and produce as output a pair consisting of the appropriate token and attribute-value, using the given translation table.

Transition Diagram for $>=$

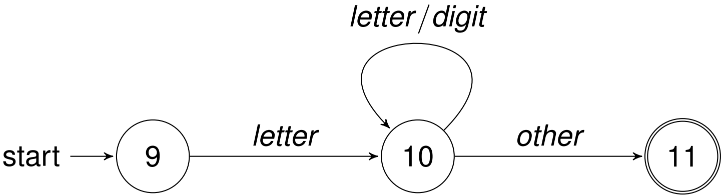


Transition Diagrams for Relational Operators



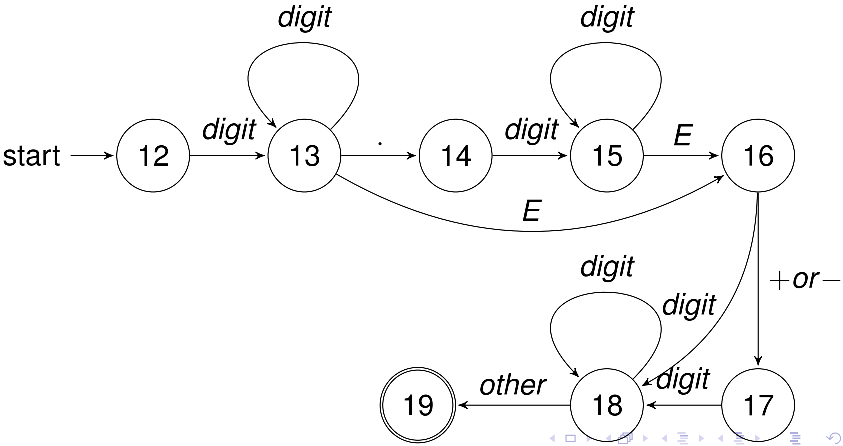
Recognitions of Tokens

Transition Diagrams for Identifiers or Keywords



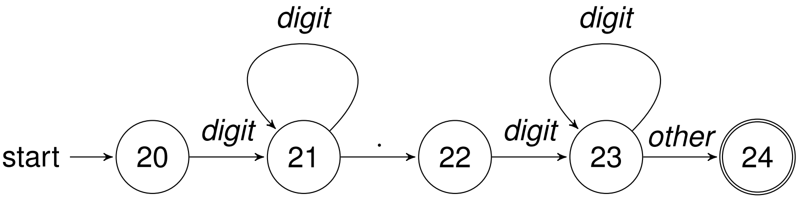
Recognitions of Tokens

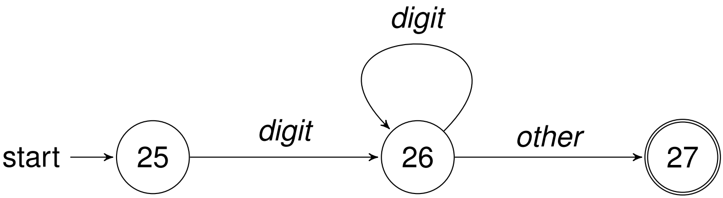
Transition Diagram for Numbers



Recognitions of Tokens

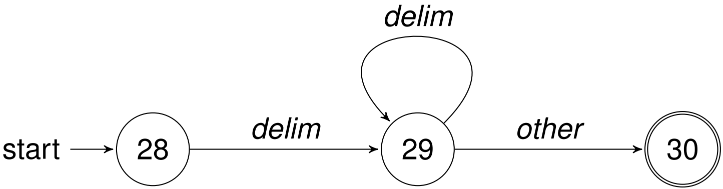
Transition Diagram for Numbers





Recognitions of Tokens

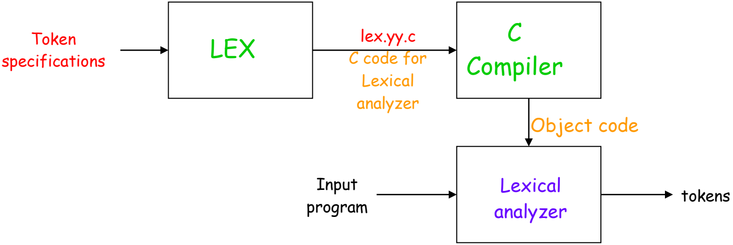
Transition Diagrams for White spaces



Implementing a Transition Diagram

```
token nexttoken()
{ while (1) {
    switch (state) {
    case 0: c = nextchar();
       if (c==blank || c==tab || c==newline) {
         state = 0;
         lexeme_beginning++;
       }
       else if (c==‘<’) state = 1;
       else if (c==‘=’) state = 5;
       else if (c==‘>’) state = 6;
       else state = fail();
       break;
     case 1:
       ...
     case 9: c = nextchar();
       if (isletter(c)) state = 10;
       else state = fail();
       break;
     case 10: c = nextchar();
       if (isletter(c)) state = 10;
       else if (isdigit(c)) state = 10;
       else state = 11;
       break;
     ...
```

Lexical Analyser Generators — Lex



References

- Alfred V. Aho, Ravi Sethi, Jeffrey D Ullman, “Compilers Principles Techniques and Tools”, Pearson Education.

Thank You