Compilers (CS31003)

Lecture 03

Lexical Analysis

- Token: const/if/relation
- Pattern: const/if/< or <= or >= or …
- Lexeme: const/if/<,<=,>=,…

A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

printf(“Total = %d\n”,score);

printf and score <- lexemes

Matching the pattern for token id, and

“Total = %d\n” is a lexeme matching literals.

Lexical Analysis

- Attribute values:

  F = m * a

- Sentinel:

Strings and Languages

- For a word $w = xy$ with $x,y \in \Sigma^*$ we call x a *prefix* and y a *suffix* of w.
- Word y is a subword of word w, if $w = xyz$ for words $x,z \in \Sigma^*$.
- Prefixes, suffixes, and, in general, subwords of w are called *proper*, if they are different from w.

Operation	Definition and Notation
Union of L and M	LUM={s|s is in L or s is in M}
Concatenation of L and M	LM={st|s is in L and t is in M}
Kleene closure of L	$L^*=\cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+=\cup_{i=1}^{\infty} L^i$

Generated scanners always search for longest prefixes of the remaining input that lead into a final state.

Example: int-constants

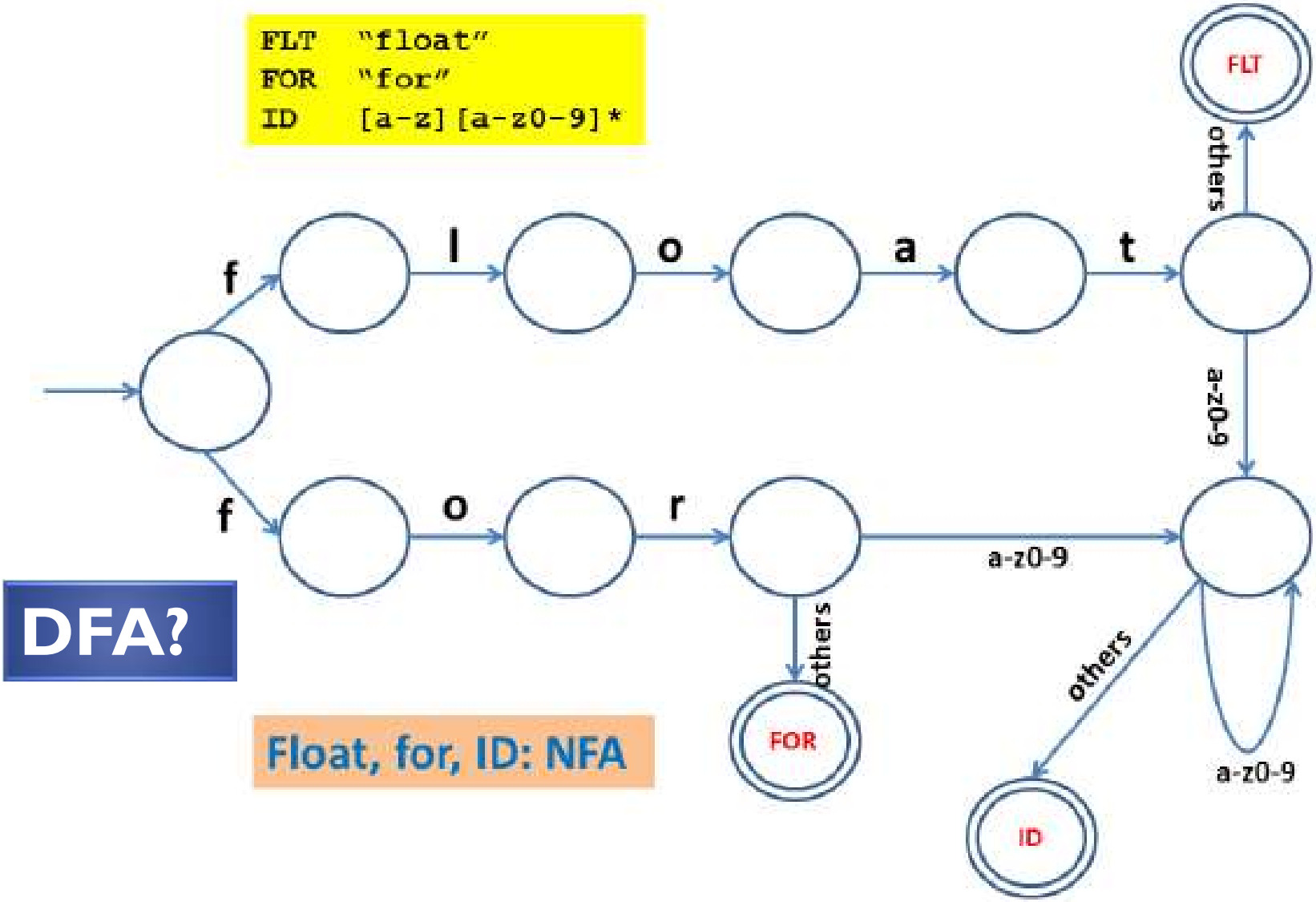
$$(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$$

Example: Character class

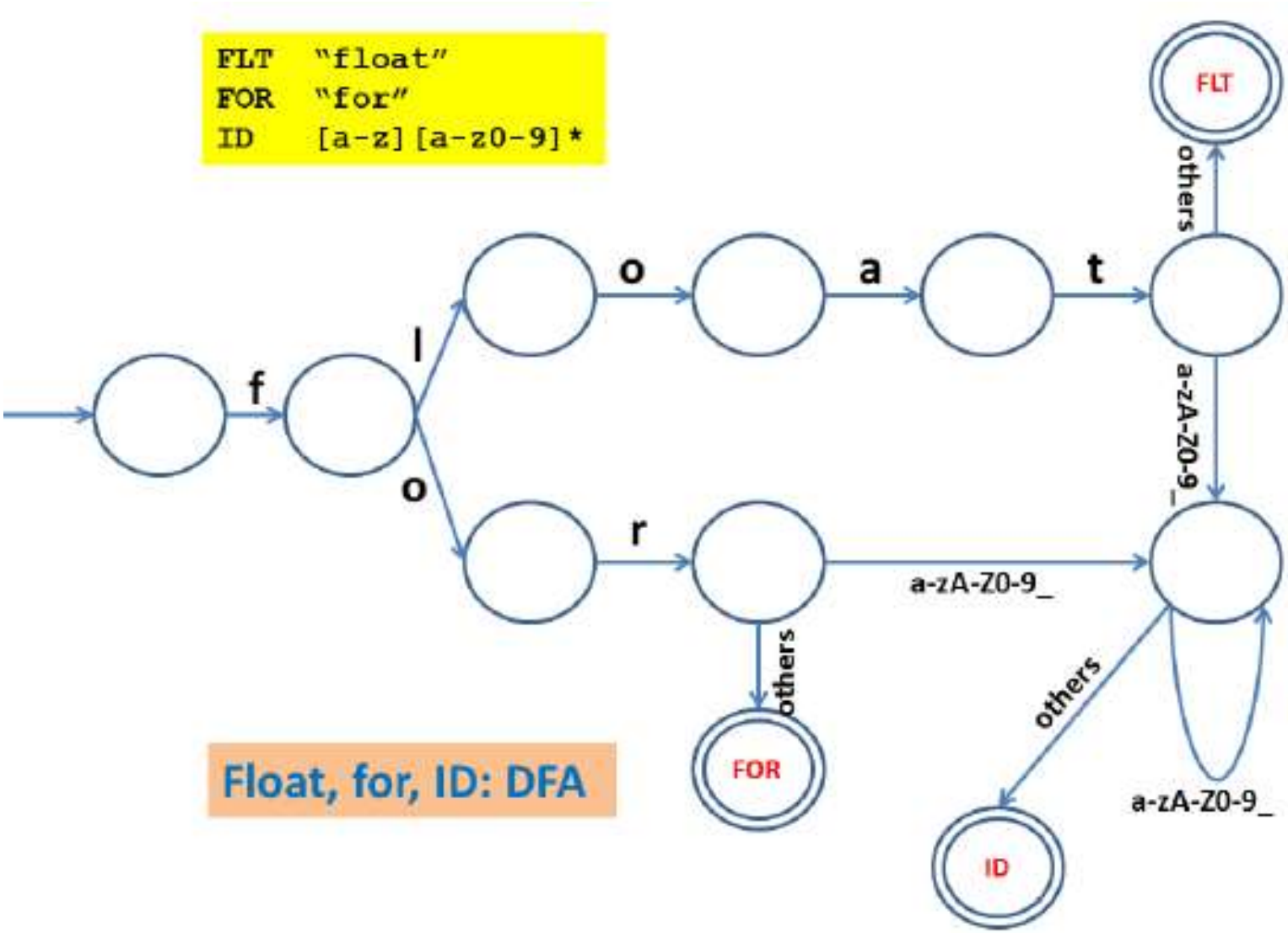
$$\begin{aligned} \text{alpha} &= a - z A - Z \\ \text{digit} &= 0 - 9 \end{aligned}$$

$$\text{Id} = \text{alpha}(\text{alpha} \mid \text{digit})^*$$

NFA recognizes float, for, ID



DFA recognizes float, for, ID



Lexical Analysis Rules

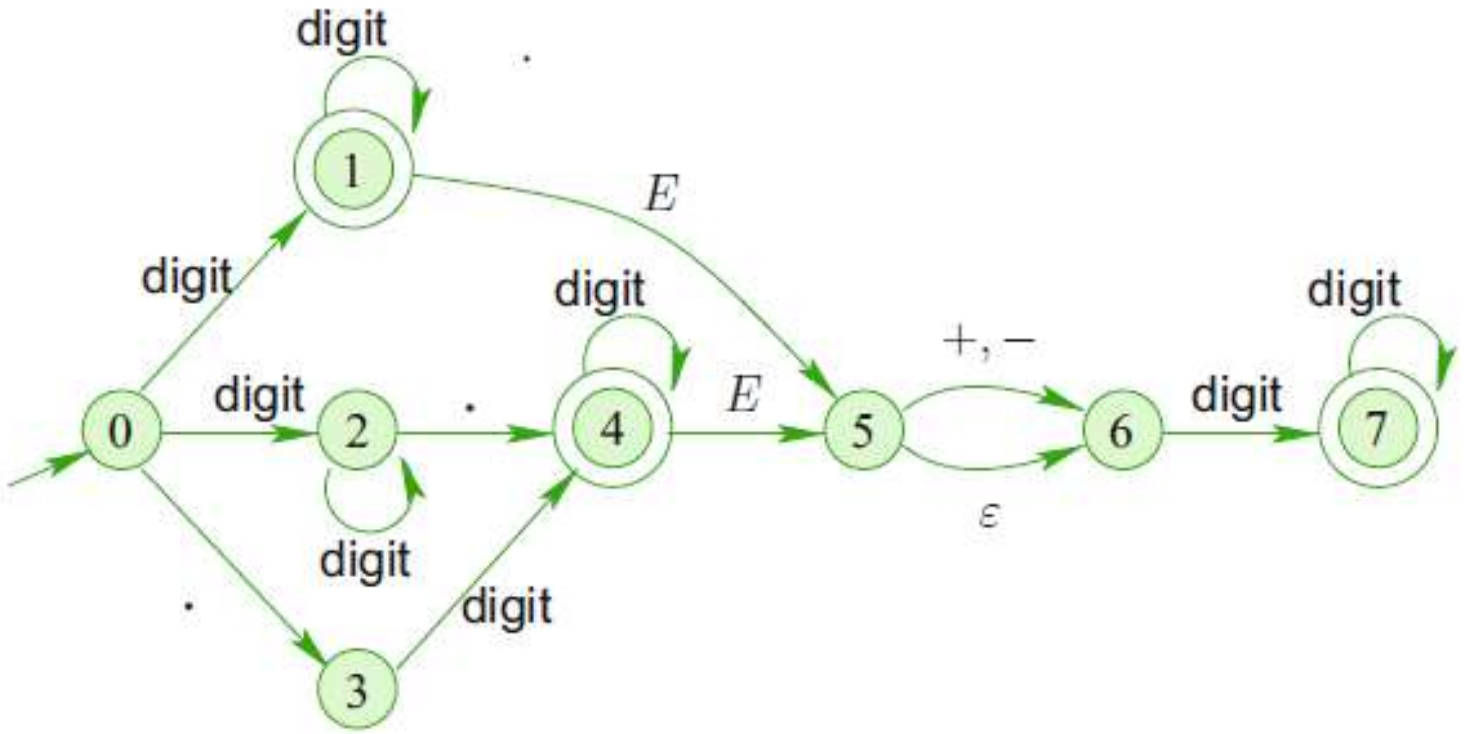
number → digits optFrac optExp
digit → 0 | 1 | 2 | …| 9
digits → digit digit*
optFrac → . digit | ε
optExp → (E (+ | - | ε) digit) | ε

integer and float constants

id → letter (letter | digit)*
letter → A | B | C … | Z | a | b | c … | z
digit → 0 | 1 | 2 | … | 9

Character class

FA to recognize unsigned *int*- and *float*-constants



Token representation

Lexemes	Token Name	Attribute Value
Any ws	-	-
if	**if**	-
then	**then**	-
else	**else**	-
Any id	**Id**	Pointer to ST
Any number	**Number**	Pointer to ST
<	**relop**	LT
<=	**relop**	LE
=	**relop**	EQ
!=	**relop**	NE
<>		
>	**relop**	GT
>=	**relop**	GE

FSM for logical operators