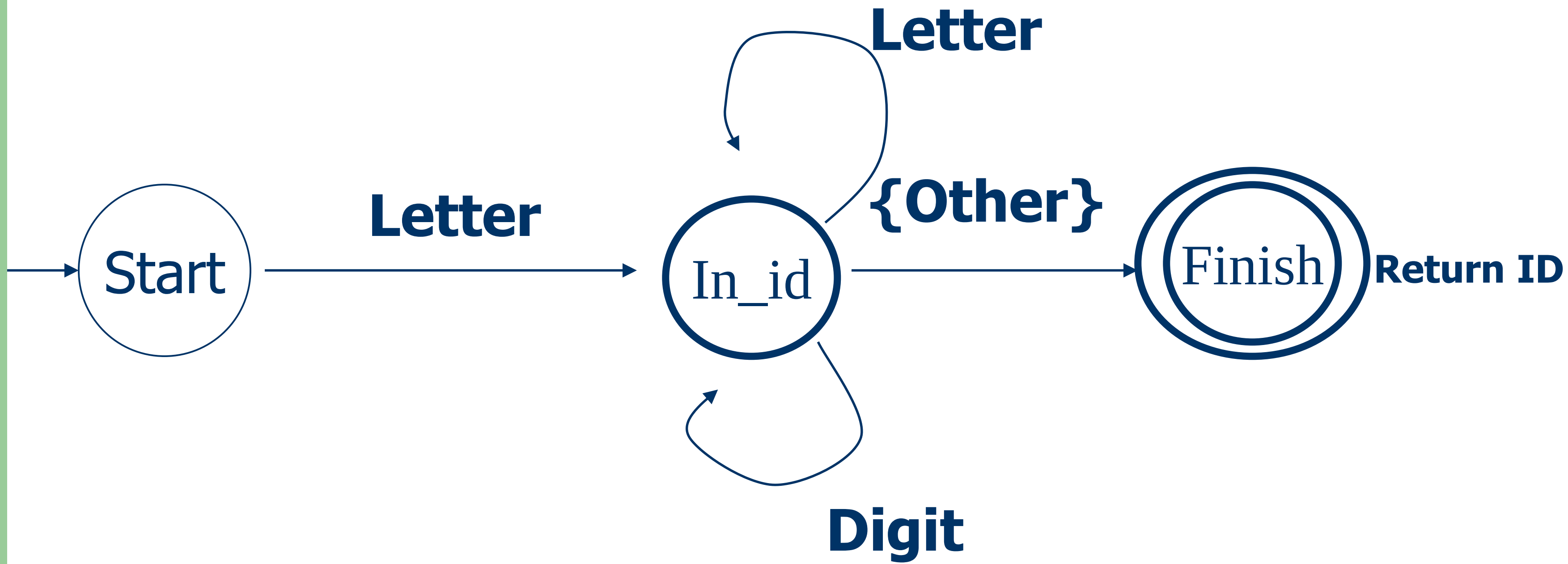
IMPLEMENTATION OF FINITE AUTOMAT IN CODE

There are several ways to translate either a DFA or an NFA into code.

Consider , again the example of a DFA that accepts identifiers consisting of a letter followed by a sequence of letters and/ or digits in its amended form that includes lookahead and the principal of longest substring.

IMPLEMENTATION OF FINITE AUTOMAT IN CODE (cont'd)



Simulation of the DFA

```
    { Starting in state 1}
If the next character is a letter then
   advance the input:
  { now in state 2}
While the next character is a letter or a digit do
   advance the input { stay in state 2}
End while;
{ go to state 3 without consuming input }
Accept
Else
{ Error or other cases }
End if;
```

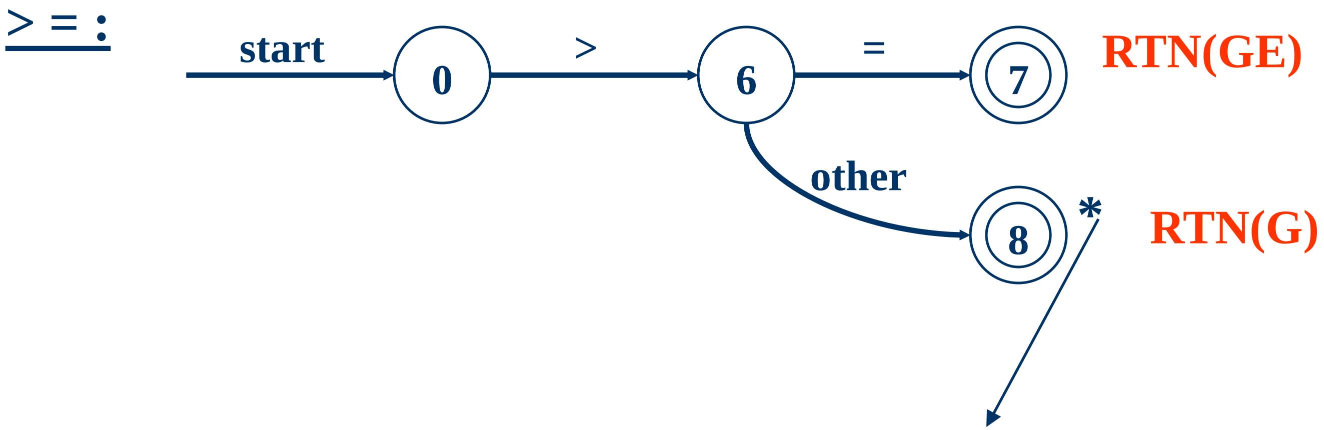
Constructing Transition Diagrams for Tokens

- Transition Diagrams (TD) are used to represent the tokens
- As characters are read, the relevant TDs are used to attempt to match lexeme to a pattern

- Each TD has:
  - States : Represented by Circles
  - Actions : Represented by Arrows between states
  - Start State : Beginning of a pattern (Arrowhead)
  - Final State(s) : End of pattern (Concentric Circles)
- Each TD is Deterministic - No need to choose between 2 different actions !

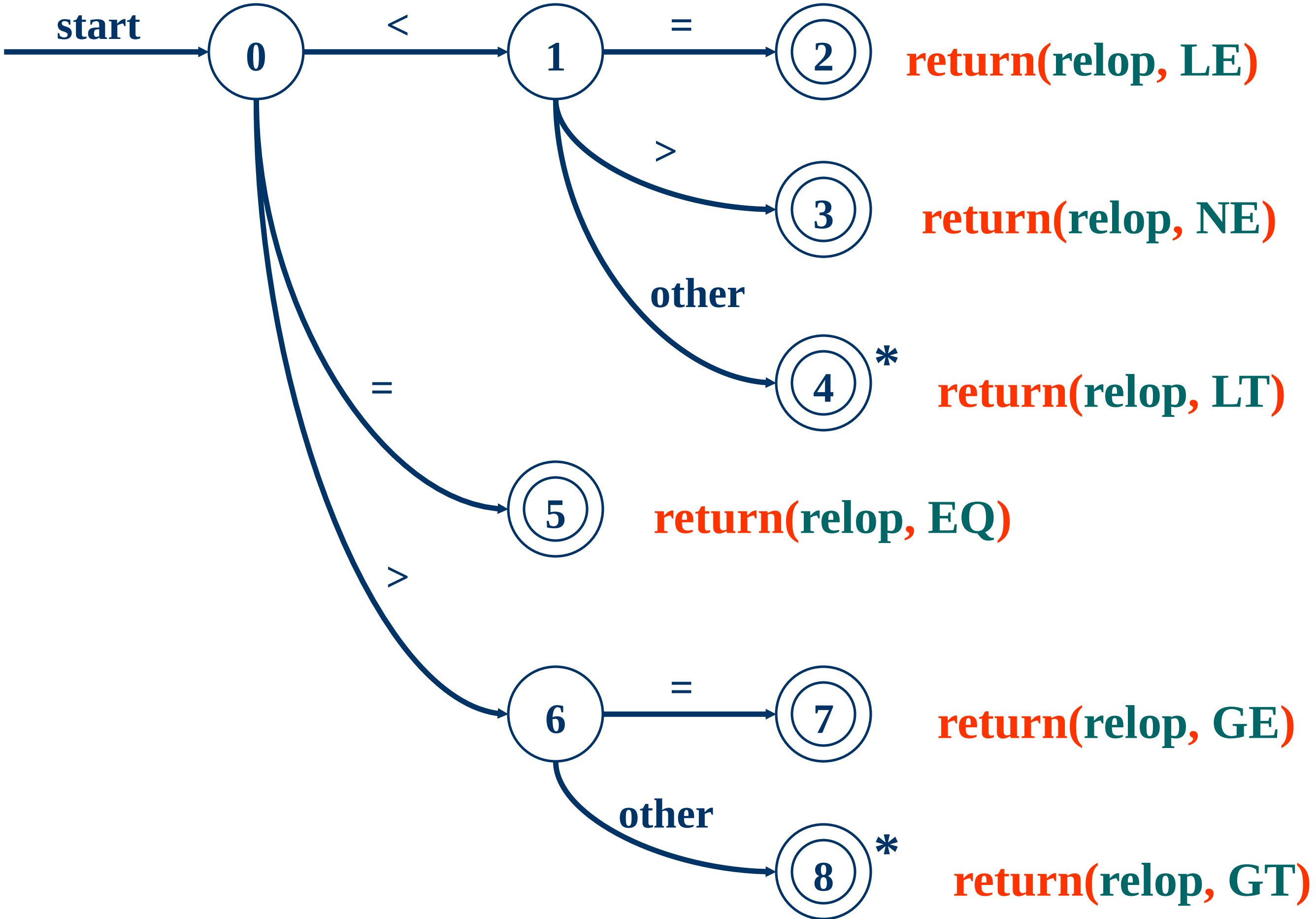
Example TDs

- Recognition Of Relational Operators



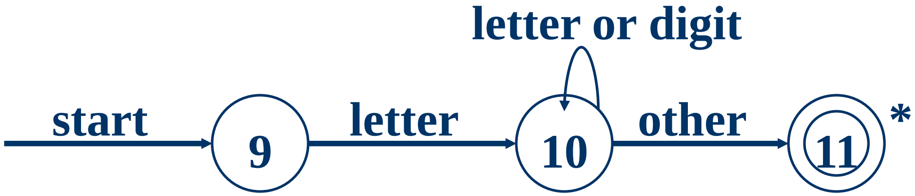
We've accepted ">" and have read other char that must be unread (means push back into input stream)

Example : All RELOPs



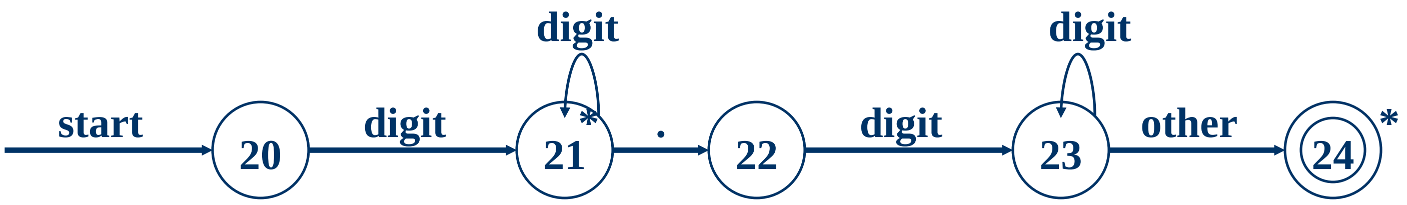
Example TDs : id

id :

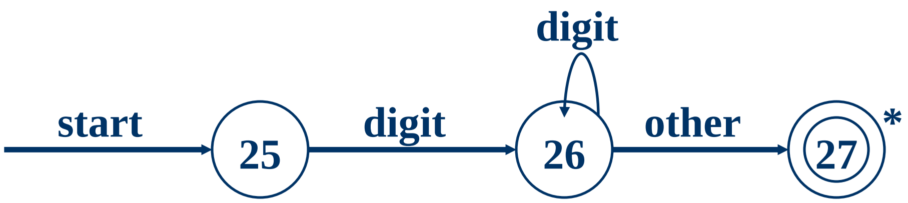


return(get_token(), install_id())

Example TDs : Unsigned #s

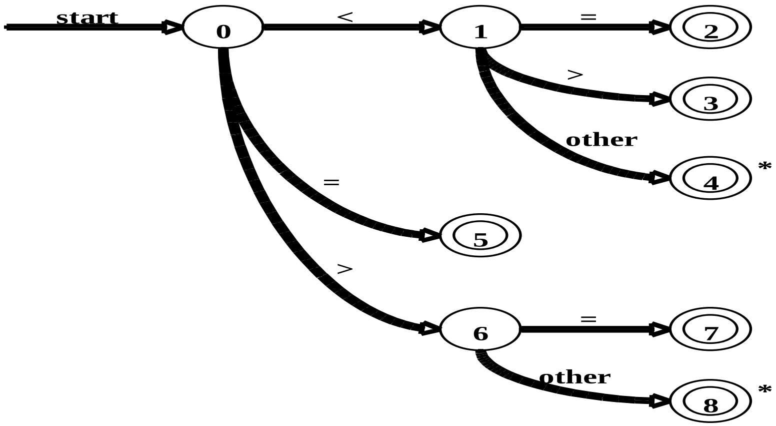


return(num, install_num())



Implementing Transition Diagrams

```
class Scanner {

  char        _la; // The lookahead character
  Token  nextToken() {
   startLexeme();    // reset window at start
      while(true) {
            switch(_state) {
                  case 0: {
                  _la = getChar();
                  if (_la == ‘<’) _state = 1;
                                  else if (_la == ‘=’) _state = 5;
                                  else if (_la == ‘>’) _state = 6;
                                  else failure(state);
                  }break;
                  case 6: {
                                  _la = getChar();
                                  if  (_la == ‘=’) _state = 7;
                                  else _state = 8;
                  }break;
            }
      }
  }
}
```



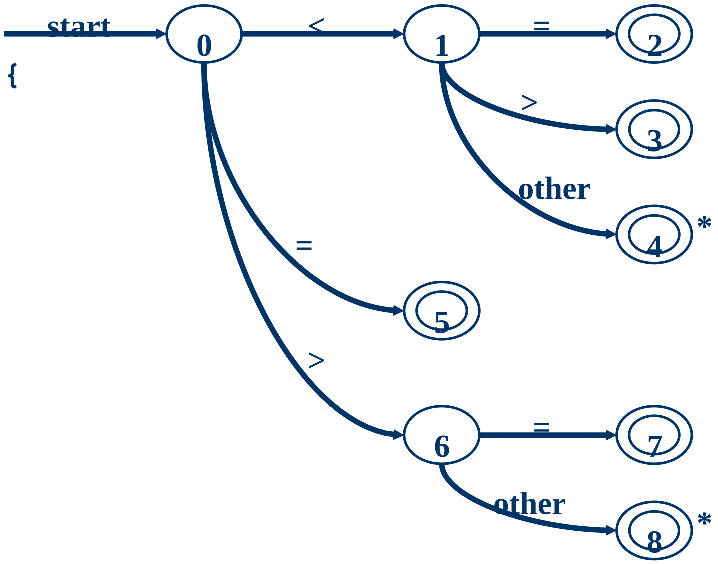
```
case 7: {
       return new Token(GEQUAL);
}break;

case 8: {
       pushBack(_la);
       return new Token(GREATER);
}
```

Implementing Transition Diagrams

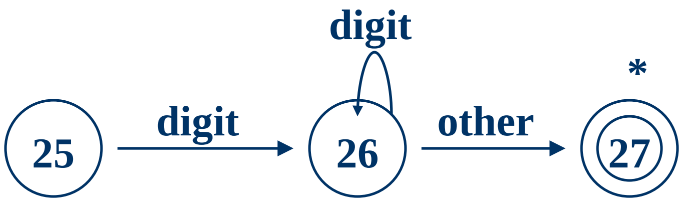
```
lexeme_beginning = forward;
state = 0;
token nexttoken()
{   while(1) {
        switch (state) {
        case 0:   c = nextchar();
           /* c is lookahead character */
           if (c== blank || c==tab || c== newline) {
              state = 0;
              lexeme_beginning++;
                 /* advance
                    beginning of lexeme */
           }
           else if (c == ‘<‘) state = 1;
           else if (c == ‘=‘) state = 5;
           else if (c == ‘>’) state = 6;
           else state = fail();
           break;
           … /* cases 1-8 here */
```

repeat
until
a “return”
occurs

FUNCTIONS USED
nextchar(), forward, retract(),
install_num(), install_id(), gettoken(),
isdigit(), isletter(), recover()



Implementing Transition Diagrams, II



```
. . . . . . . . . . . . .

case 25;  c = nextchar();
          if (isdigit(c)) state = 26;
          else state = fail();
          break;
case 26;  c = nextchar();
          if (isdigit(c)) state = 26;
          else state = 27;
          break;
case 27;  retract(1); lexical_value = install_num();
          return ( NUM );

. . . . . . . . . . . . .
```

advances `forward`

Case numbers correspond to transition diagram states !

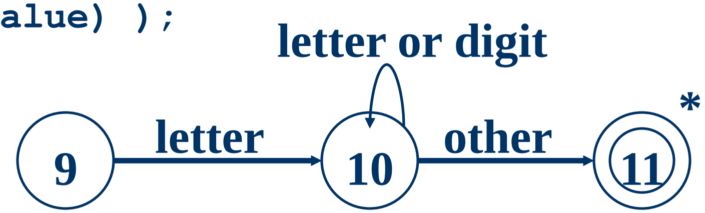
retracts `forward`

looks at the region `lexeme_beginning ... forward`

Implementing Transition Diagrams, III

```
. . . . . . . . . . . . . .

 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;
 case 11;   retract(1); lexical_value = install_id();
           return ( gettoken(lexical_value) );
. . . . . . . . . . . . . .
```

reads token
name from ST



When Failures Occur:

```
Init fail()
{    start = state;
     forward = lexeme beginning;
     switch (start) {
        case 0:   start = 9;  break;
        case 9:   start = 12; break;
        case 12:  start = 20; break;
        case 20:  start = 25; break;
        case 25:  recover();  break;
        default:   /* lex error */
     }
     return start;
}
```

Switch to
next transition
diagram

What Else Does Lexical Analyzer Do?

All Keywords / Reserved words are matched as ids

- **After the match, the symbol table or a special keyword table is consulted**
- **Keyword table contains string versions of all keywords and associated token values**

if	**15**
then	**16**
begin	**17**
…	**…**

- **When a match is found, the token is returned, along with its symbolic value, i.e., "then", 16**
- **If a match is not found, then it is assumed that an id has been discovered**

ASSINGMENT

THANKS