innovate achieve lead

Implementation of Lexical Analyzer through Multiple Transition Diagrams

Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

BITS Pilani
Pilani Campus

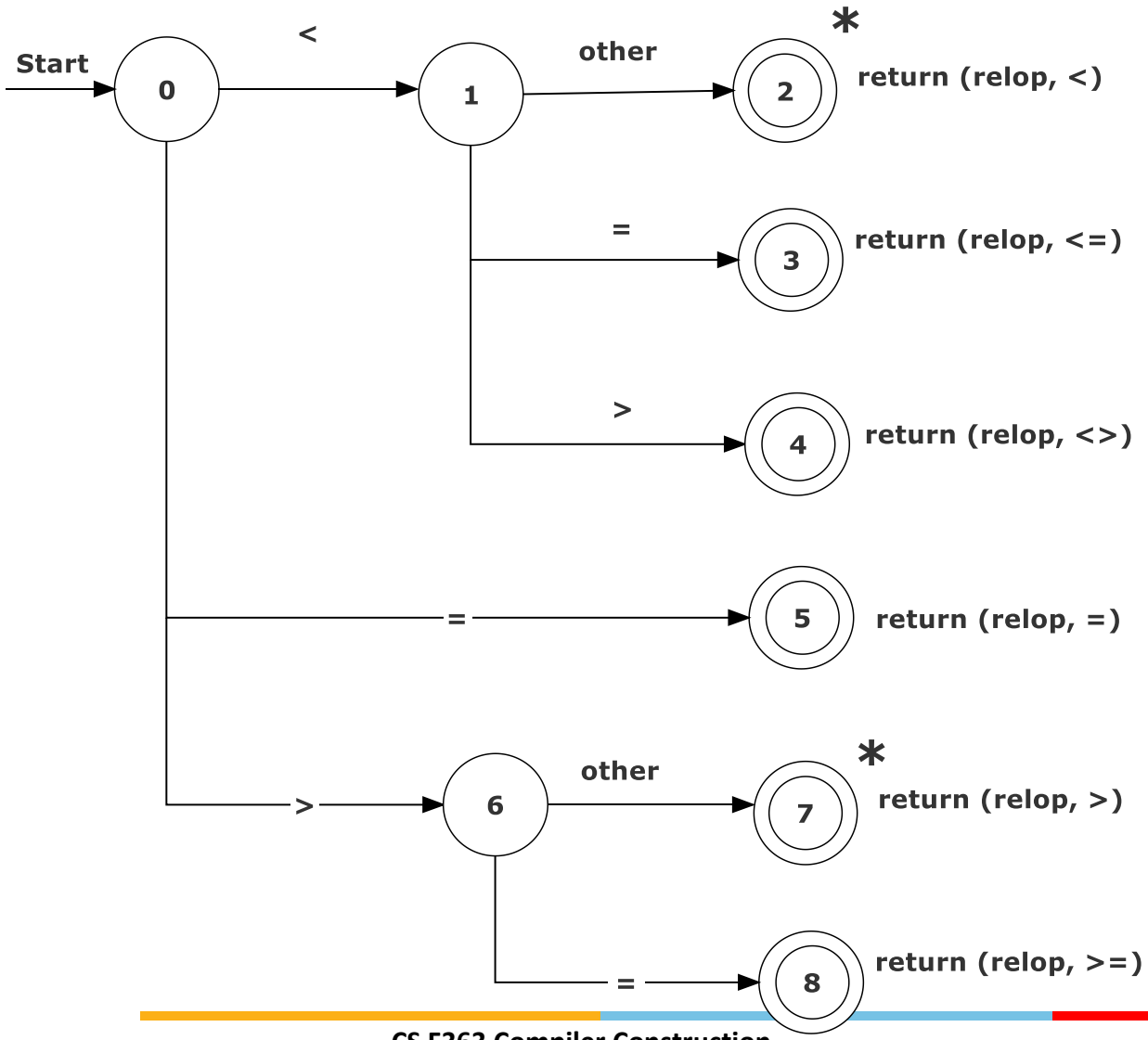
Process of Tokenization

Consider the following language specification in the form of regular definition as follows:

$$relop \rightarrow < | <= | <> | = | > | >=$$

Design a lexical analyzer that will return pair <token, lexeme> to the syntax analyzer

Transition Diagram for Relational Operators $relop \rightarrow < | <= | <> | = | > | >=$



Example

Design a regular definition and transition diagram notation for hexadecimal and octal constants. Consider hex notation for your compiler must initiate with `0x|0X` whereas octal notation should initiate with *0*. In addition, both the notations may or may not include the Qualifier (unsigned (`u|U`) or long (`l|L`) or `null`) as a suffix (at the end of their respective notations).

Regular Definition for Hexadecimal and Octal Constants

$$hex \rightarrow 0|1|2|---|9|A|B|C|---|F$$

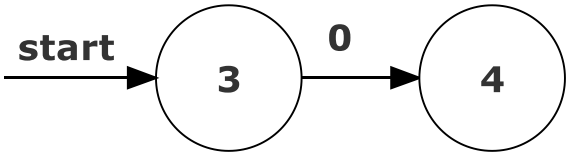
$$oct \rightarrow 0|1|2|---|7$$

$$Qualifier \rightarrow u|U|l|L$$

$$Octal\ Cons\tan t \rightarrow 0\, oct^{+}\ (Qualifier|\in)$$

$$Hexadecimal\ Cons\tan t \rightarrow 0\,(x|X)\, hex^{+}\ (Qualifier|\in)$$

Transition Diagram for Hex and Oct Constants



$hex \rightarrow 0|1|2|---|9|A|B|C|---|F$

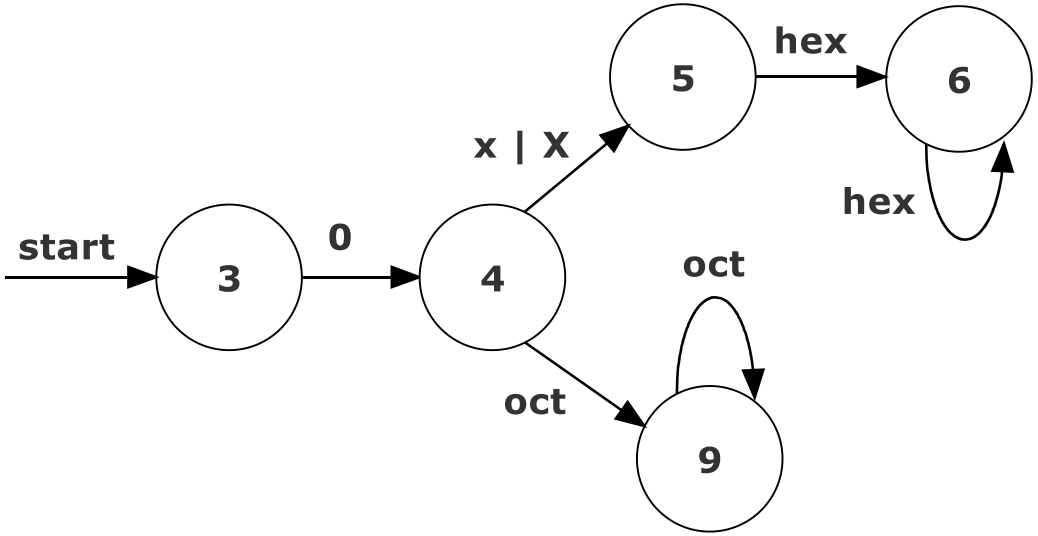
$oct \rightarrow 0|1|2|---|7$

$Qualifier \rightarrow u|U|l|L$

$Octal\ Constant \rightarrow 0\ oct^{+}\ (Qualifier|\in)$

$Hexadecimal\ Constant \rightarrow 0\ (x|X)\ hex^{+}\ (Qualifier|\in)$

Transition Diagram for Hex and Oct Constants



$hex \rightarrow 0|1|2|---|9|A|B|C|---|F$

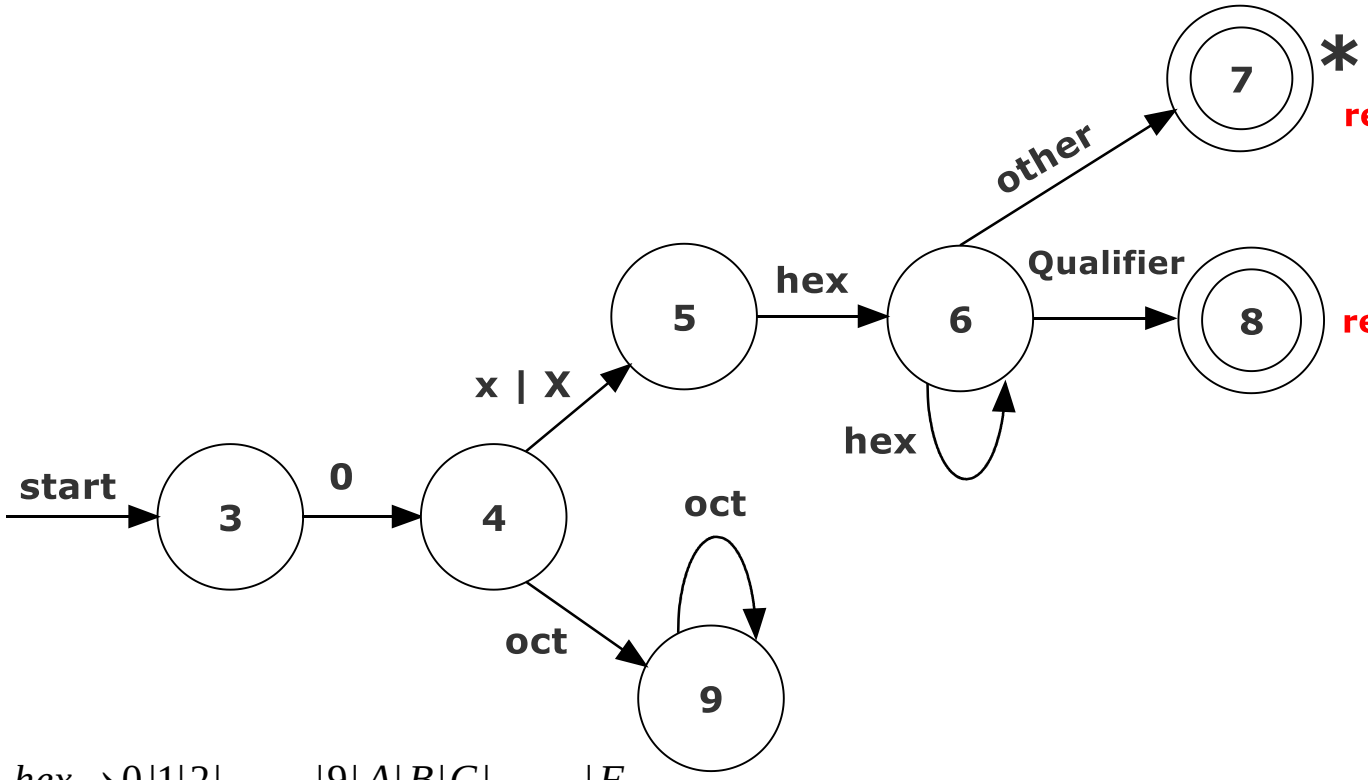
$oct \rightarrow 0|1|2|---|7$

$Qualifier \rightarrow u|U|l|L$

$Octal\ Cons\tan t \rightarrow 0\, oct^{+}\ (Qualifier|\in)$

$Hexadecimal\ Cons\tan t \rightarrow 0\,(x|X)\, hex^{+}\ (Qualifier|\in)$

Transition Diagram for Hex and Oct Constants



$hex \rightarrow 0|1|2|---|9|A|B|C|---|F$

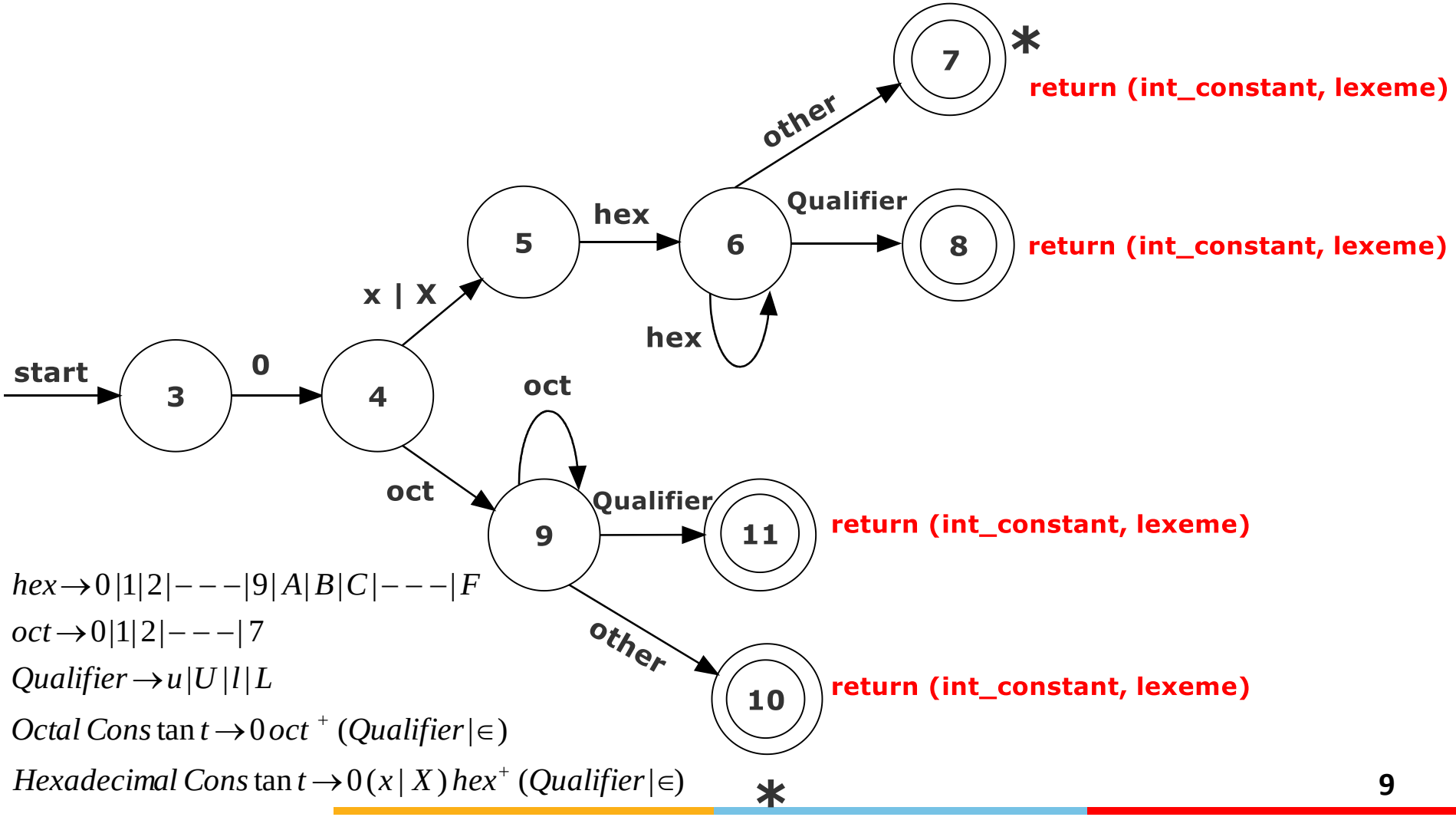
$oct \rightarrow 0|1|2|---|7$

$Qualifier \rightarrow u|U|l|L$

$Octal\ Constant \rightarrow 0\ oct^{+}\ (Qualifier|\in)$

$Hexadecimal\ Constant \rightarrow 0\ (x|X)\ hex^{+}\ (Qualifier|\in)$

Transition Diagram for Hex and Oct Constants



$hex \rightarrow 0|1|2|---|9|A|B|C|---|F$

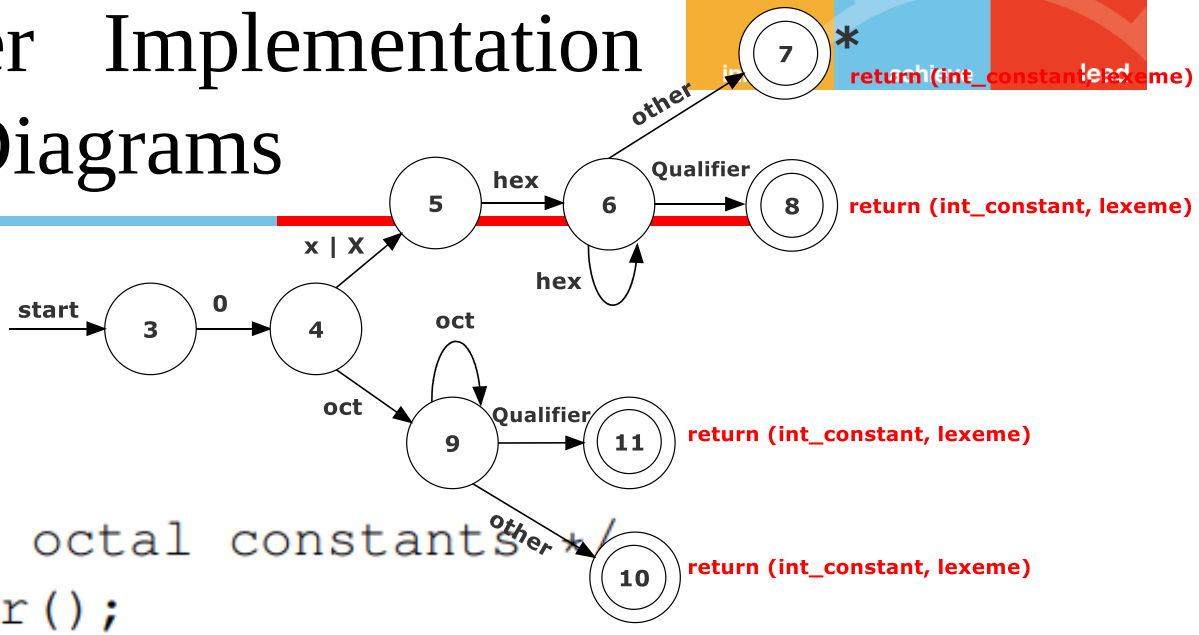
$oct \rightarrow 0|1|2|---|7$

$Qualifier \rightarrow u|U|l|L$

$Octal\ Constant \rightarrow 0\ oct^{+}\ (Qualifier|\in)$

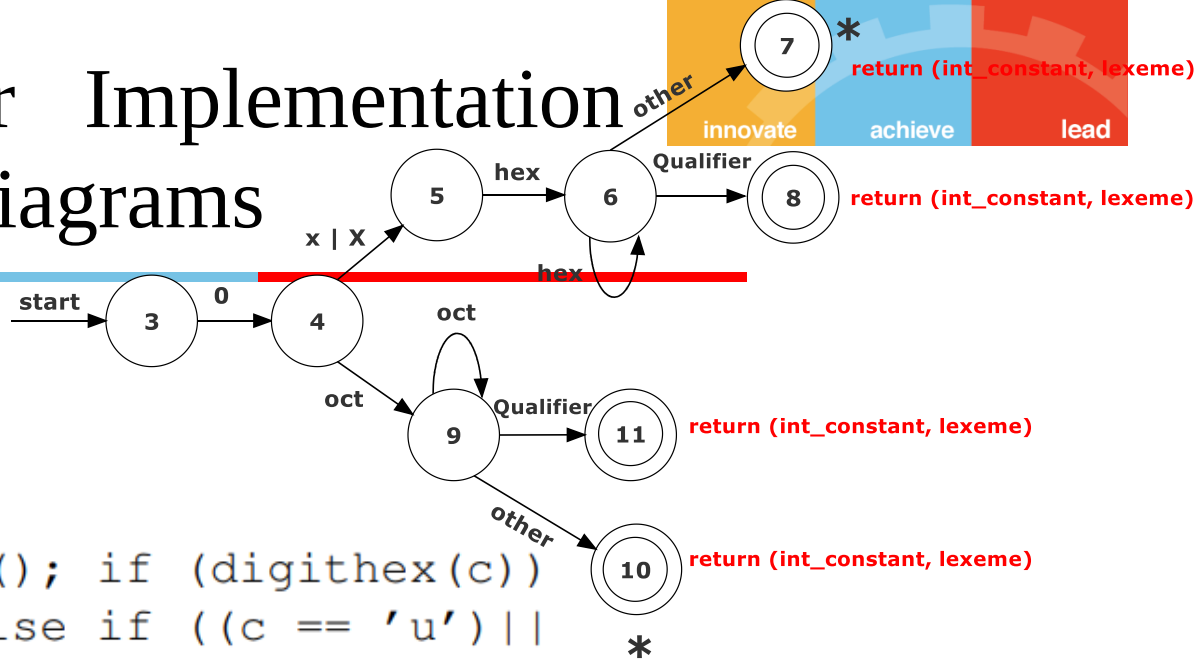
$Hexadecimal\ Constant \rightarrow 0\ (x|X)\ hex^{+}\ (Qualifier|\in)$

Lexical Analyzer Implementation from Transition Diagrams



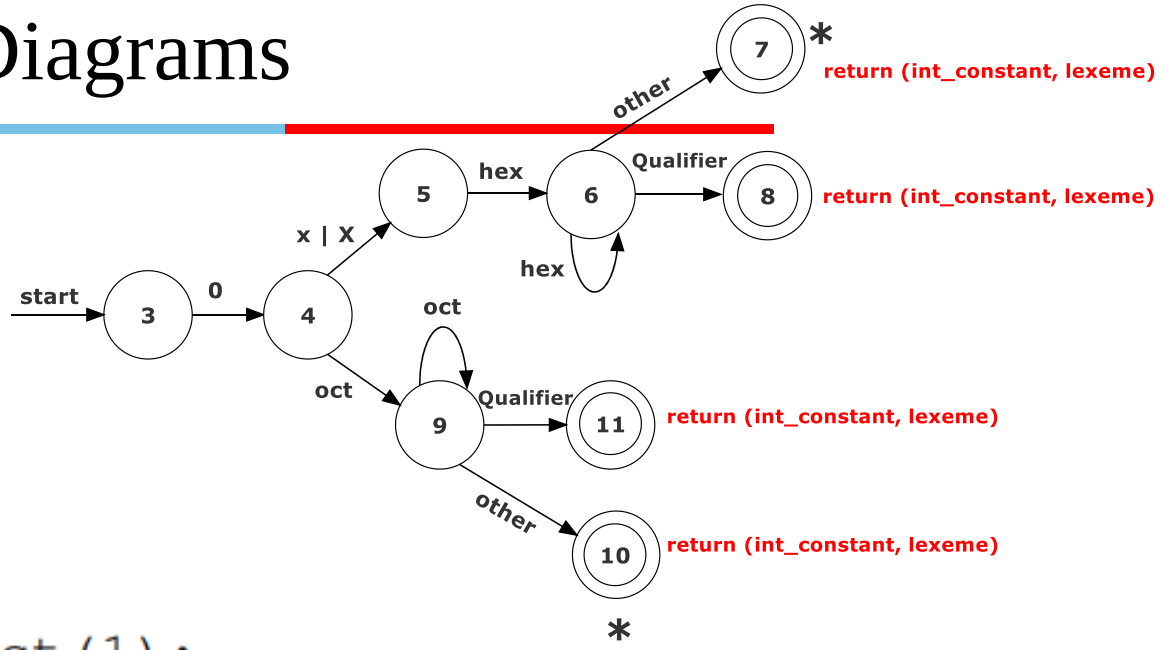

```
/* recognize hexa and octal constants */
  case 3: c = nextchar();
          if (c == '0') state = 4; break;
          else state = failure();
  case 4: c = nextchar();
          if ((c == 'x') || (c == 'X'))
          state = 5; else if (digitoct(c))
          state = 9; else state = failure();
          break;
  case 5: c = nextchar(); if (digithex(c))
          state = 6; else state = failure();
          break;
```

Lexical Analyzer Implementation from Transition Diagrams




```
    case 6: c = nextchar(); if (digithex(c))
            state = 6; else if ((c == 'u')||
            (c == 'U')||(c == 'l')||
            (c == 'L')) state = 8;
            else state = 7; break;
    case 7: retract(1);
/* fall through to case 8, to save coding */
    case 8: mytoken.token = INT_CONST;
            mytoken.value = eval_hex_num();
            return(mytoken);
    case 9: c = nextchar(); if (digitoct(c))
            state = 9; else if ((c == 'u')||
            (c == 'U')||(c == 'l')||(c == 'L'))
            state = 11; else state = 10; break;
```

Lexical Analyzer Implementation from Transition Diagrams




```
    case 10: retract(1);
/* fall through to case 11, to save coding */
    case 11: mytoken.token = INT_CONST;
             mytoken.value = eval_oct_num();
             return(mytoken);
```

Generalized Transition Diagram for Unsigned Numbers

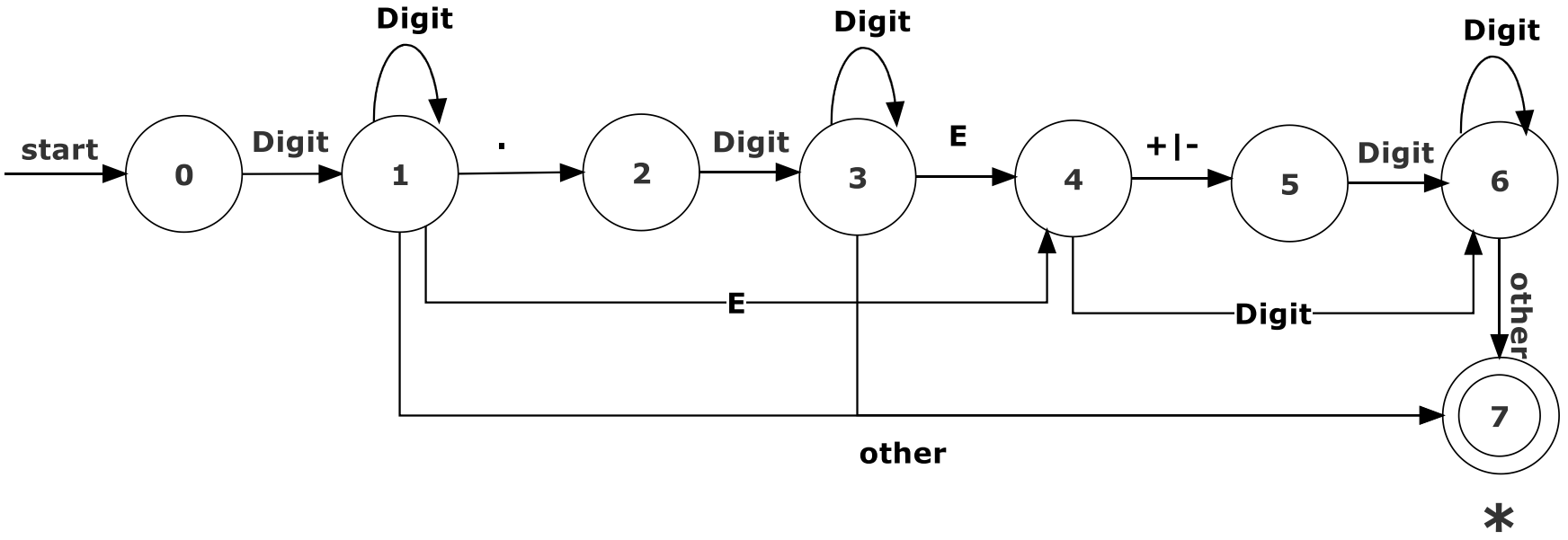
$$Digit \rightarrow 0|1|2|----|9$$

$$Digits \rightarrow Digit^{+}$$

$$Fraction \rightarrow '.'Digits|\in$$

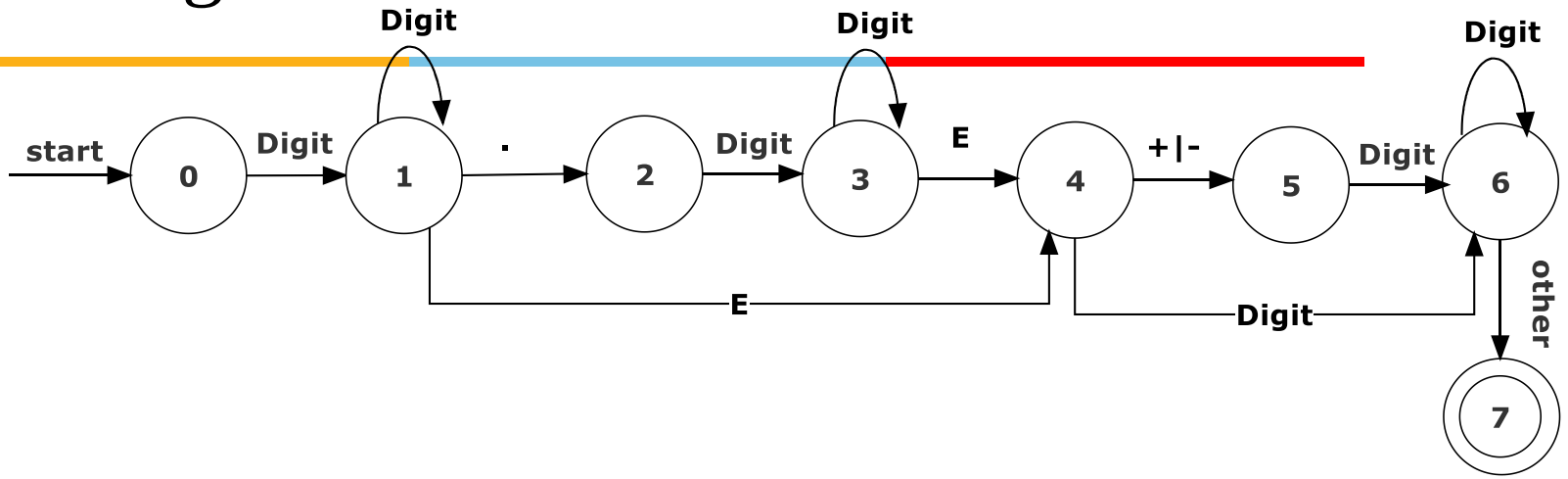
$$Exponent \rightarrow (E\,(+|-|\in)\,Digits)|\in$$

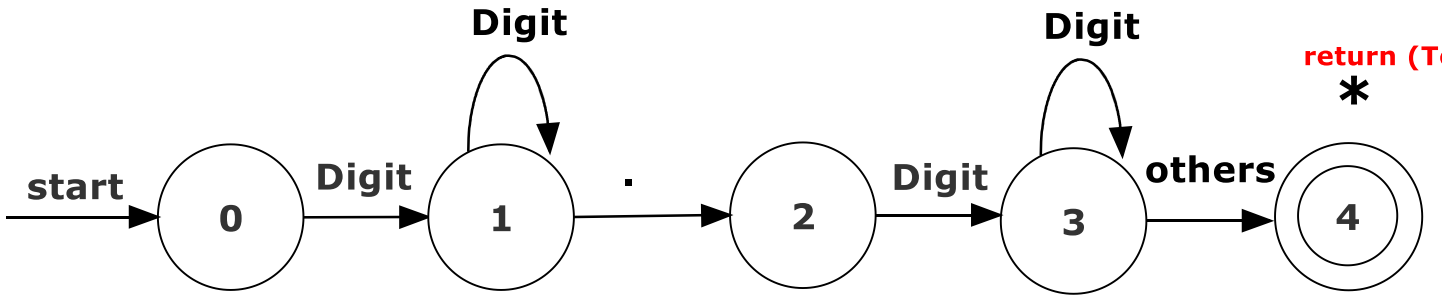
$$Number \rightarrow Digits\;Fraction\;Exponent$$

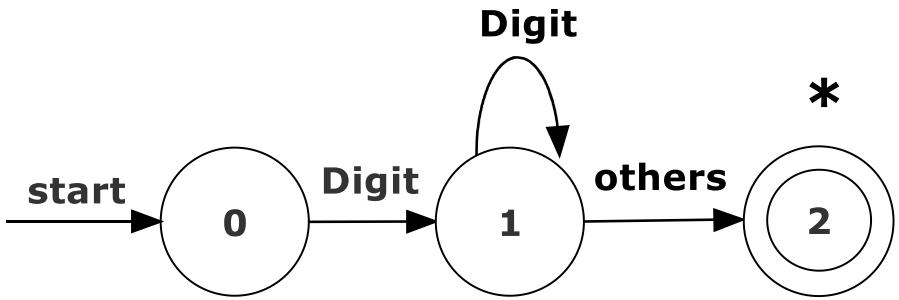


return (token, lexeme)

Multiple Transition Diagrams for Unsigned Numbers







Process of Tokenization in Transition Diagram

The matching process should always start with some transition diagram.

If failure occurs in one transition diagram.

- Retract the forward pointer to the start state.
- Activate the next transition diagram.

If failure occurs in all transition diagrams then throw the Lexical Error.

Generation of Lexical Analyzer from Transition Diagram

The more complex is your transition diagram

- More complex would be the equivalent source code.
- Since, each state will need to handle more decisions.

During implementation, complex transition diagrams may give rise to errors.

Generation of Lexical Analyzer from Transition Diagram

Different transition diagrams must be combined appropriately to generate a lexical analyzer.

- Merging different transition diagrams is not so easy.

Trace different transition diagrams one after another.

To find the longest match, all transition diagrams must be tried and the longest match must be used.