Compiler Design

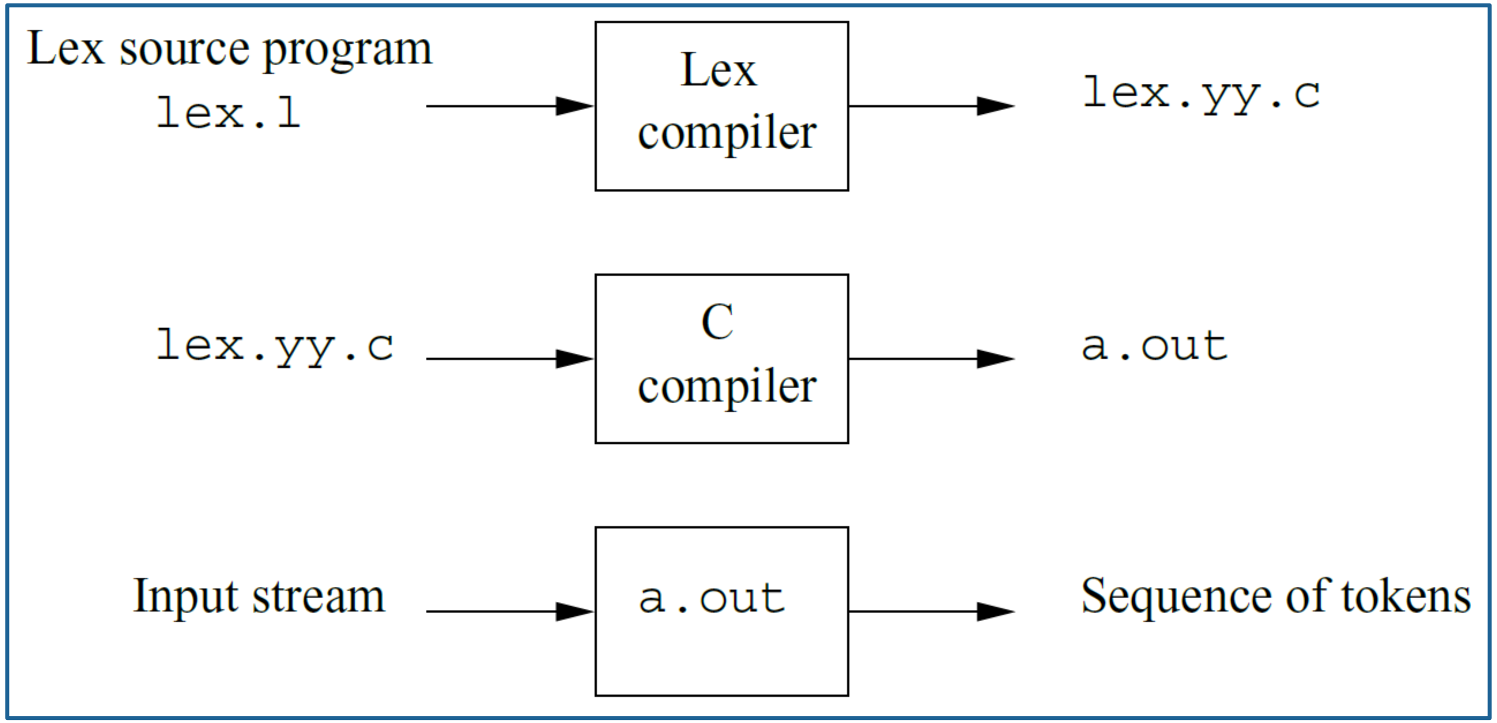
Fatemeh Deldar

Isfahan University of Technology

1403-1404

The Lexical-Analyzer Generator Lex

- **Lex**, or in a more recent implementation **Flex**, allows one to specify a lexical analyzer by specifying regular expressions to describe patterns for tokens



The Lexical-Analyzer Generator Lex

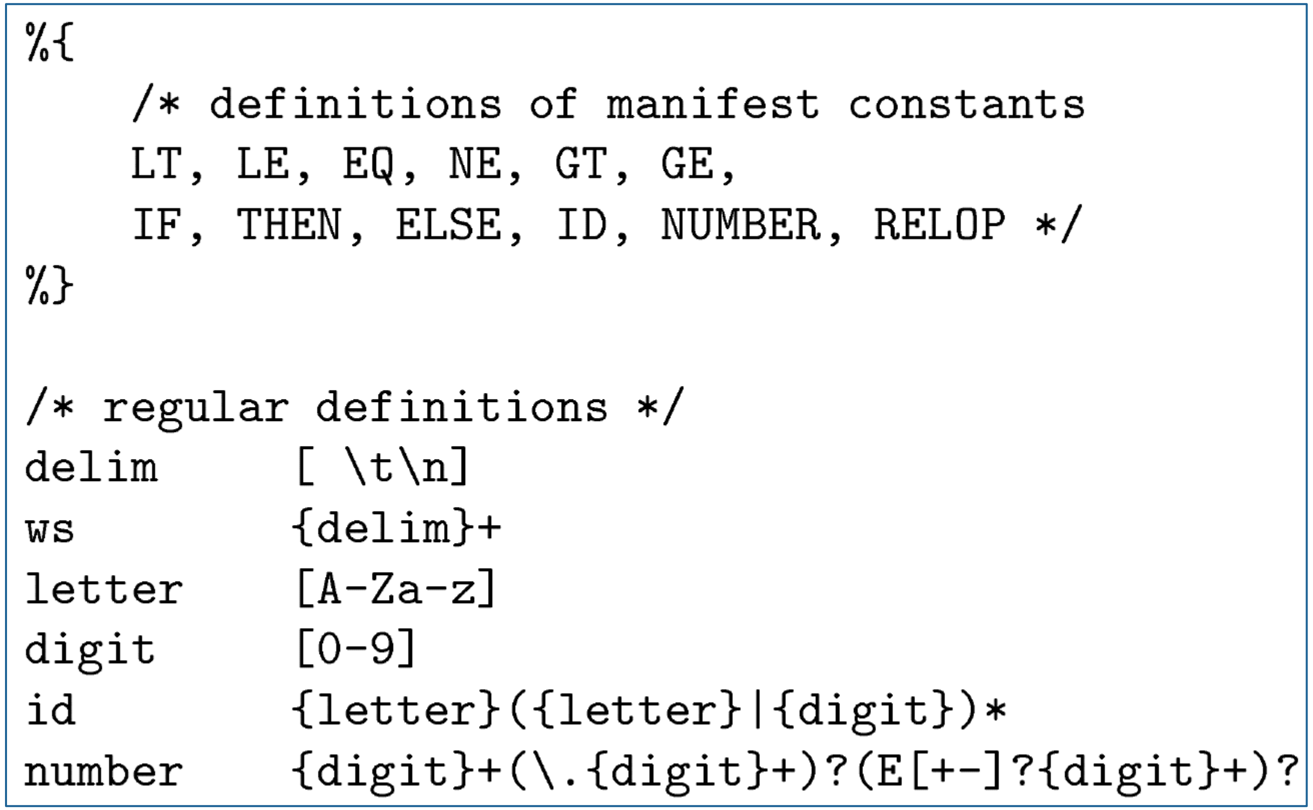
- **A Lex program has the following form:**

```
declarations
%%
translation rules
%%
auxiliary functions
```

The translation rules each have the form: Pattern { Action }

The Lexical-Analyzer Generator Lex

- **Example (declarations):**



```
%{
    /* 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}+)?
```

The Lexical-Analyzer Generator Lex

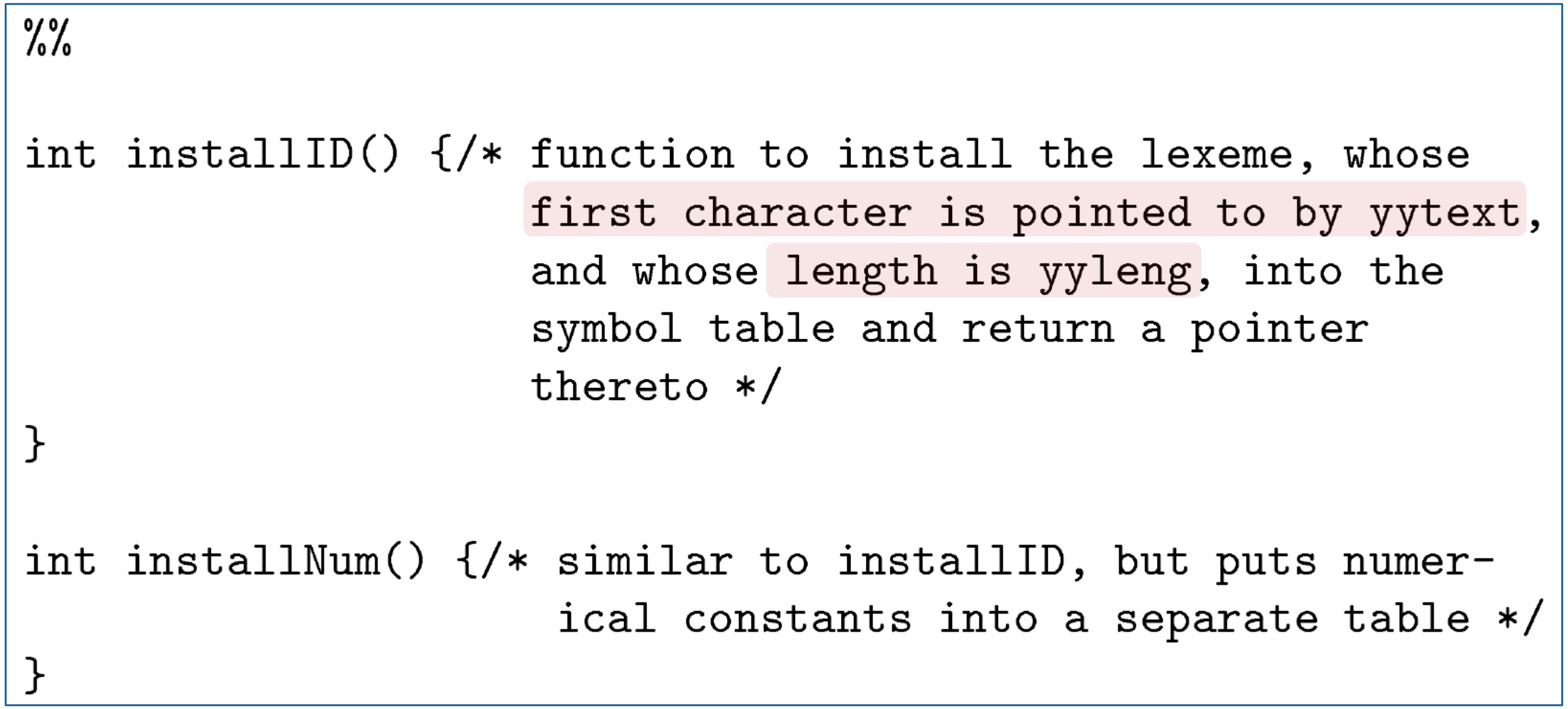
- **Example (translation rules):**

yylval برای ارسال اطلاعات اضافی در مورد lexeme به پارسر (تحلیل‌گر نحوی) استفاده می‌شود.

```
%%

{ws}      {/* no action and no return */}
if        {return(IF);}
then      {return(THEN);}
else      {return(ELSE);}
{id}      {yylval = (int) installID(); return(ID);}
{number}  {yylval = (int) installNum(); 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);}
```

The Lexical-Analyzer Generator Lex

- **Example (auxiliary functions):**



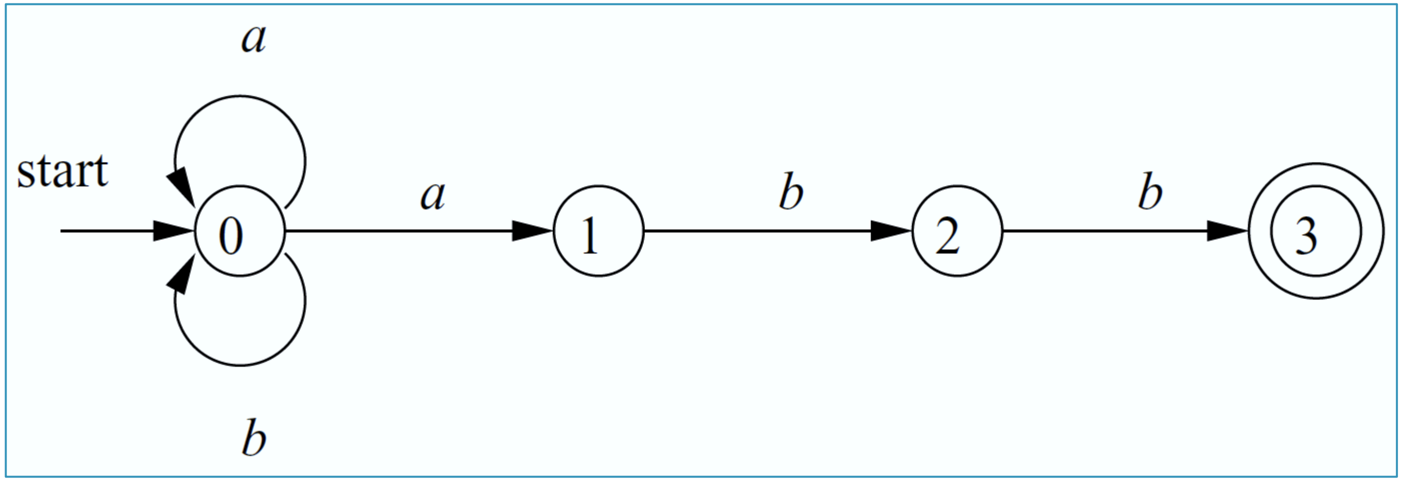
```
%%

int installID() {/* function 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 */
}

int installNum() {/* similar to installID, but puts numer-
                     ical constants into a separate table */
}
```

Nondeterministic Finite Automata

- **A nondeterministic finite automaton (NFA) consists of:**
  1. A finite set of states S
  2. A set of input symbols $\sum$, the input alphabet
  3. A transition function that gives, for each state, and for each symbol in $\sum \cup \{\epsilon\}$ a set of next states
  4. A state s_0 from S that is distinguished as the start state (or initial state)
  5. A set of states F, a subset of S, that is distinguished as the accepting states (or final states)

Nondeterministic Finite Automata

- **Example**
  - The transition graph for an NFA recognizing the language of regular expression $(a|b)^*abb$



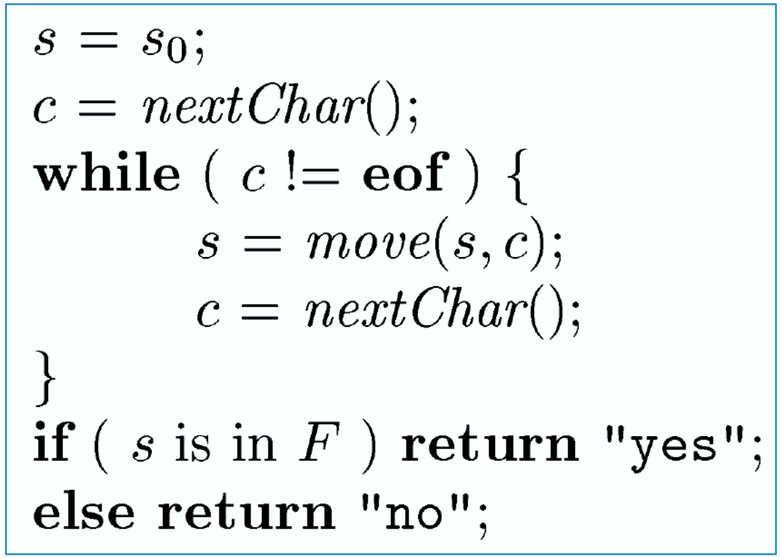
  - The string aabb is accepted by the above NFA

$$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$$

Deterministic Finite Automata

- A deterministic finite automaton (DFA) is a special case of an NFA where:
  1. There are no moves on input ϵ
  2. For each state s and input symbol a, there is exactly one edge out of s labeled a
- Every regular expression and every NFA can be converted to a DFA accepting the same language

Simulating a DFA

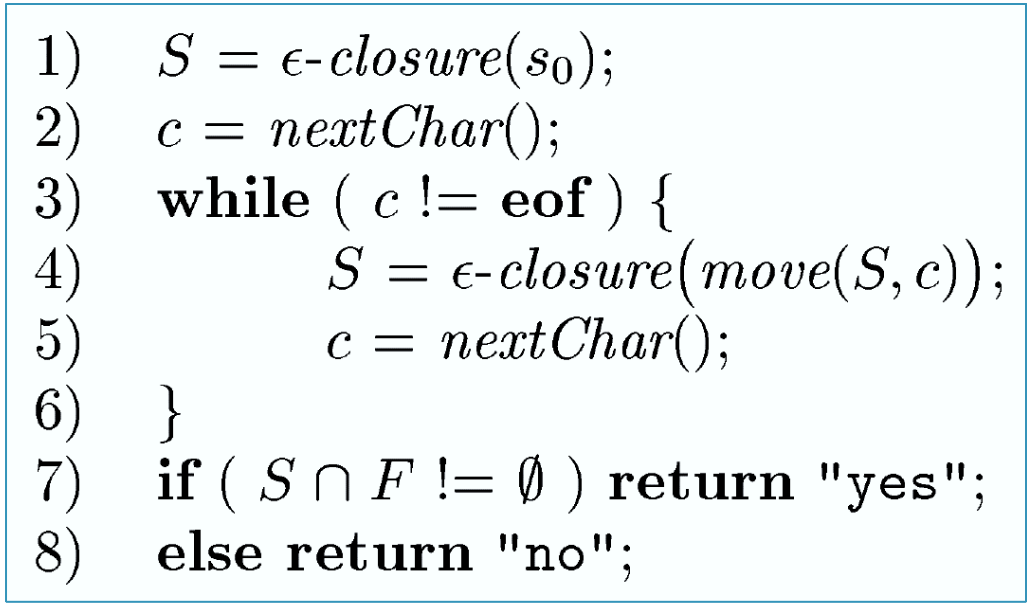


```
$s = s_0$;
$c = nextChar()$;
while ( $c$ != eof ) {
        $s = move(s, c)$;
        $c = nextChar()$;
}
if ( $s$ is in $F$ ) return "yes";
else return "no";
```

چند نکته

- DFA از سرعت بالاتری نسبت به NFA برخوردار است. اگر طول رشته n باشد پذیرش در DFA از O(n) است (خطی) در حالی که در NFA از $O(k^n)$ است (نمایی) که k تعداد حالات NFA است.
- تعداد حالات DFA معمولا از NFA بیشتر است، پس مصرف حافظه DFA از NFA بیشتر است.
- کلاس ماشین‌های متناهی غیرقطعی (نامعین) با کلاس ماشین‌های متناهی قطعی (معین) هم‌ارز هستند، پس به ازای هر NFA یک DFA معادل وجود است.

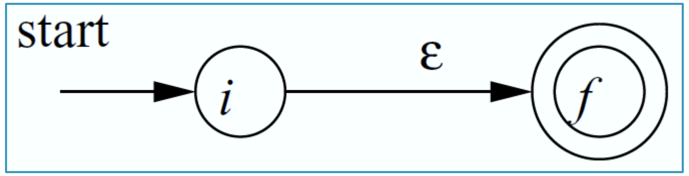
Simulation of an NFA

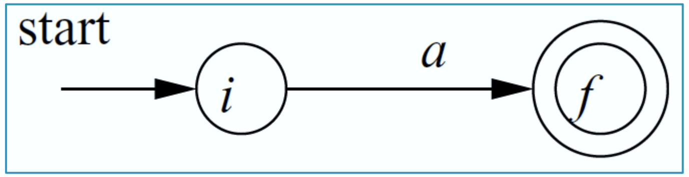


```
$S = \epsilon\text{-}closure(s_0)$;
$c = nextChar()$;
while ( $c$ != eof ) {
        $S = \epsilon\text{-}closure(move(S, c))$;
        $c = nextChar()$;
}
if ( $S \cap F$ != $\emptyset$ ) return "yes";
else return "no";
```

Construction of an NFA from a Regular Expression

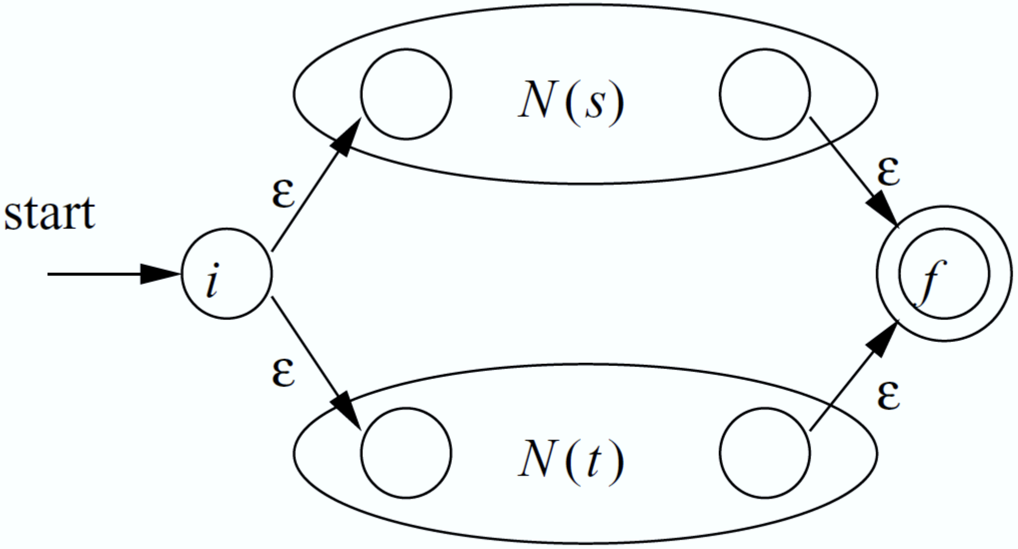
- **Basis**





Construction of an NFA from a Regular Expression

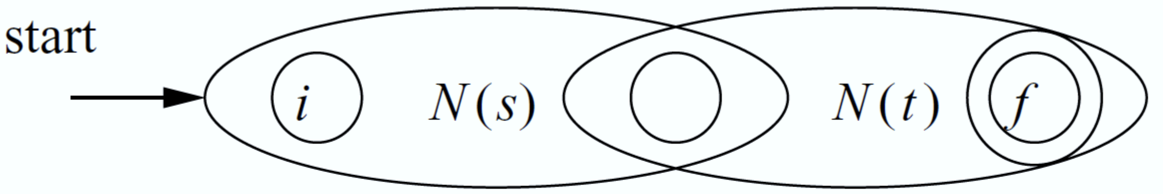
- **Induction**



NFA for the union of two regular expressions

Construction of an NFA from a Regular Expression

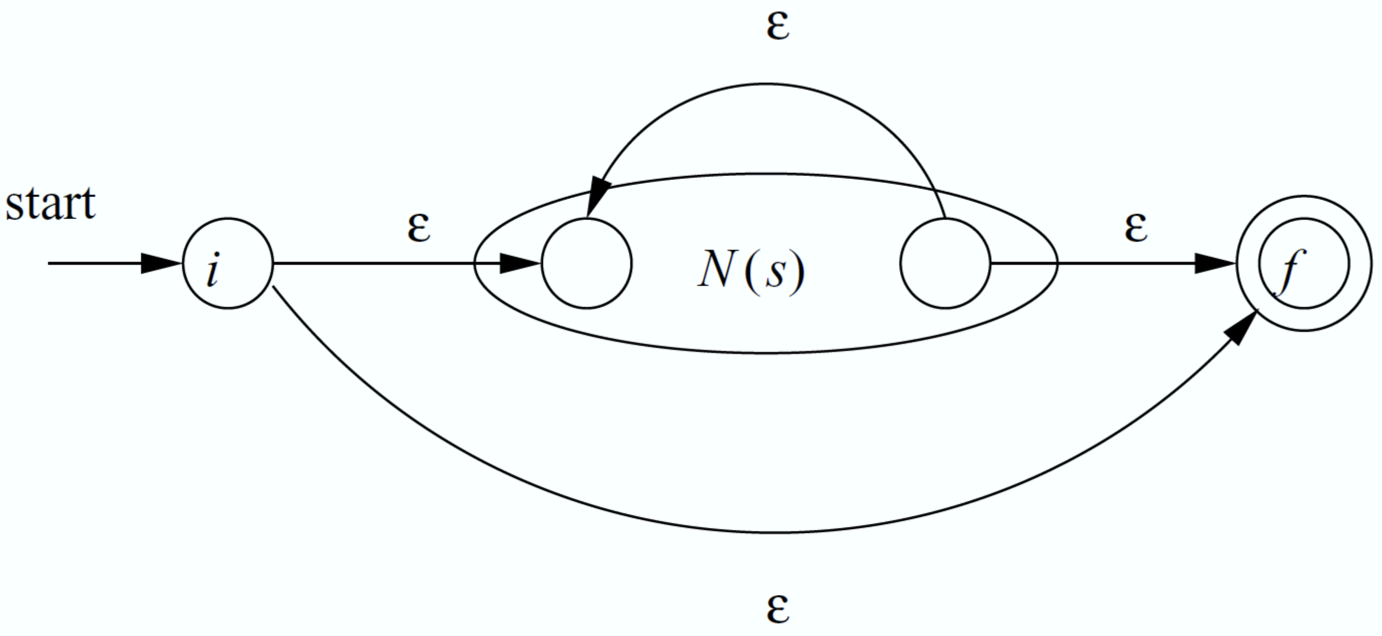
- **Induction**



NFA for the concatenation of two regular expressions

Construction of an NFA from a Regular Expression

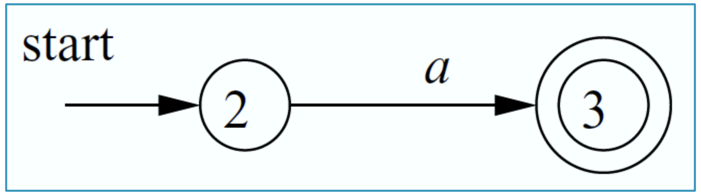
- **Induction**

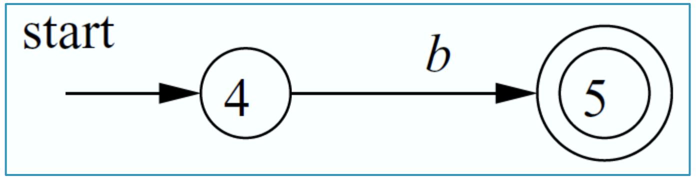


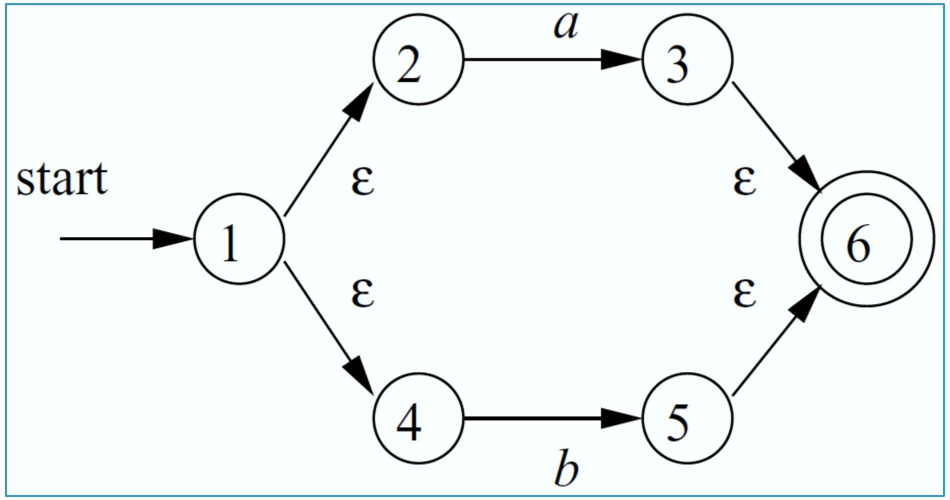
NFA for the closure of a regular expression

Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^*abb$

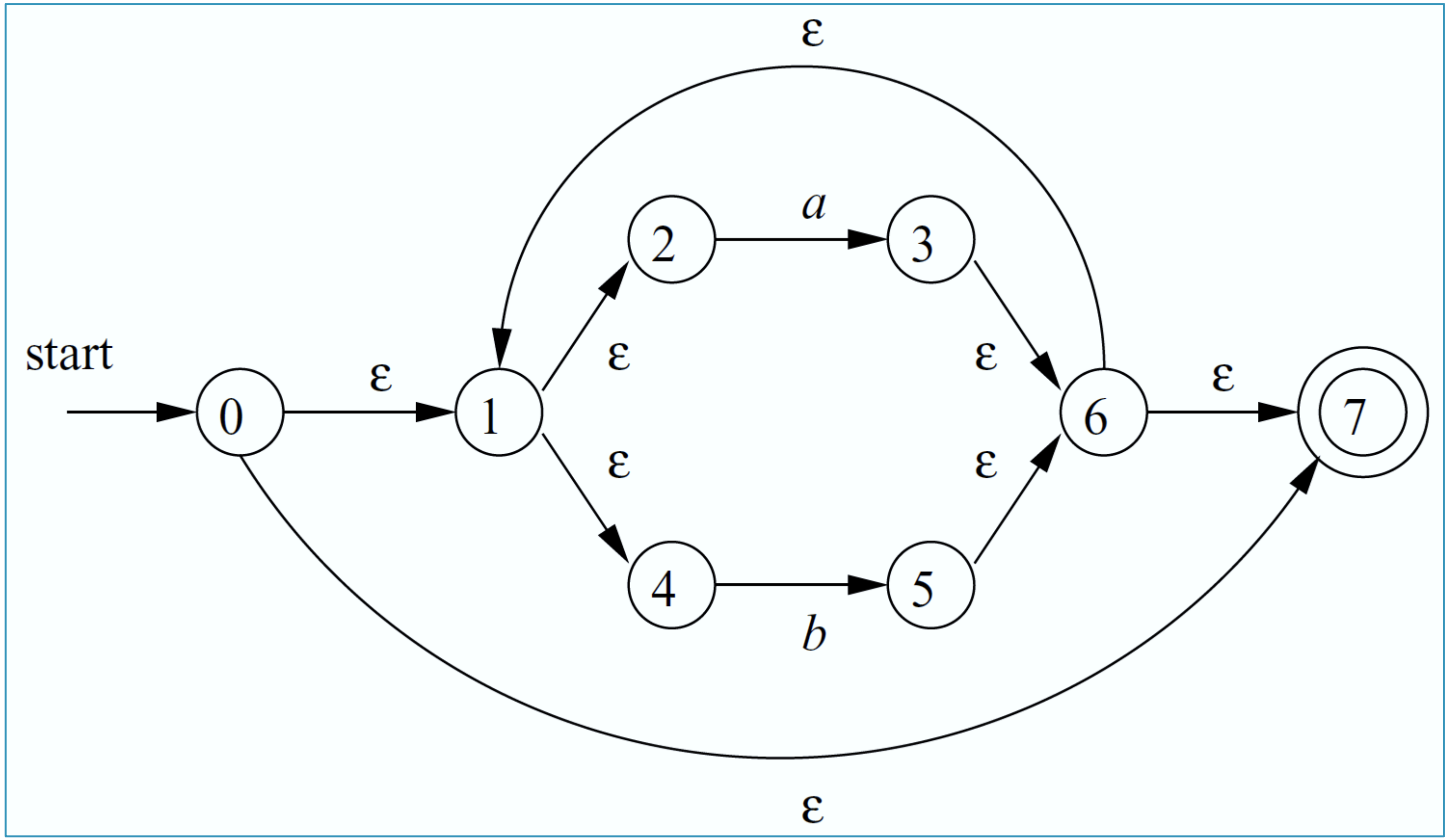






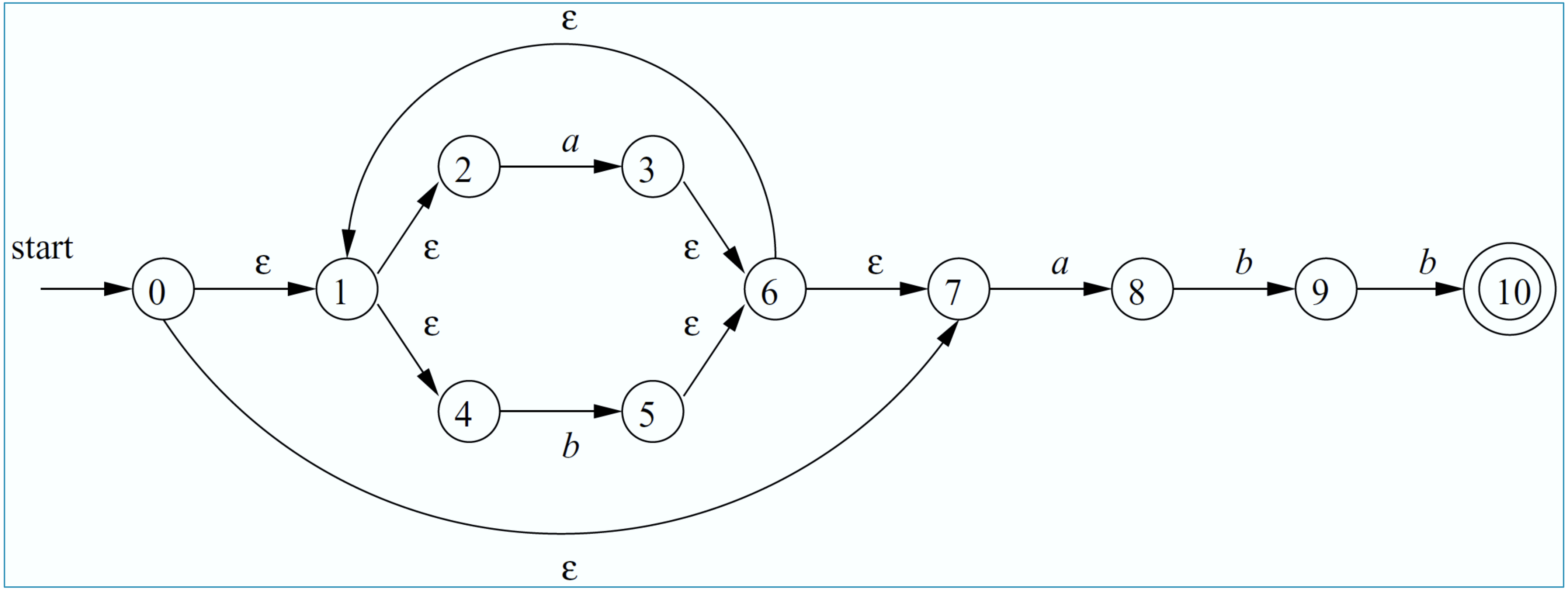
Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^*abb$



Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^*abb$



ε
a
start
0
ε
1
ε
2
3
ε
6
ε
7
a
8
b
9
b
10
ε
ε
4
5
b
ε