Compilers

module of the course “Professional English”

Yulia Burkatovskaya

Department of Computer Engineering

Associate professor

2. Lexical analysis

- Basics of lexical analysis
- Regular languages and regular expressions
- Finite automata
- Regular expression into NFA
- NFA into DFA
- Table implementation of FA
- Lexical errors

2.1. Basics of lexical analysis

Removal of white spaces and comments

Before:

```
if (x==0) then
   y=1;           //case 1
else
   z=2;           //case 2
```

After:

```
if (x==0) then\n\ty=1;\nelse\n\tz=2;
```

Basics of lexical analysis

Token classes

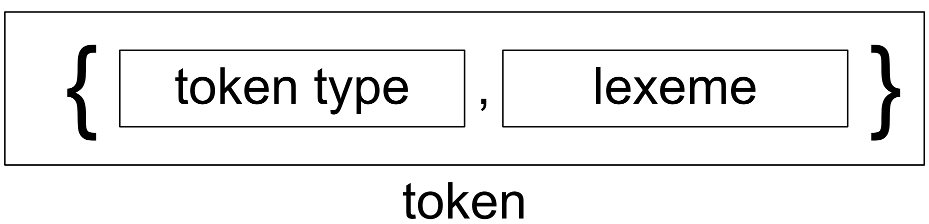
- **Identifiers**
  Sequence of letters and digits starting with a letter
- **Keywords**
  if, then, else…
- **Whitespaces**
  Non-empty sequence of blanks, newlines and tabs.
- **Integers**
  Non-empty sequence of digits
- **Operators**
  =, ==, <, >,…

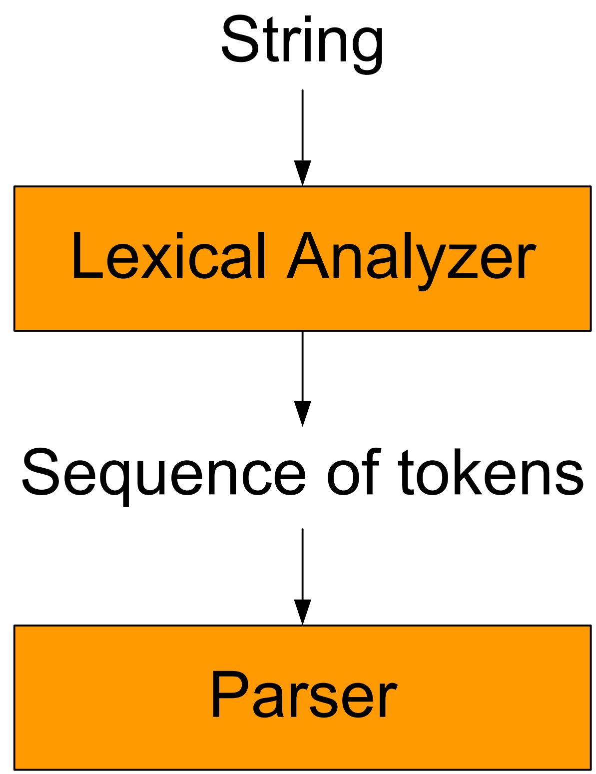
Etc.

Basics of lexical analysis

Lexical analysis tasks:

- To classify substrings according to token classes;
- To form tokens for the parser.





Basics of lexical analysis

FORTRAN EXAMPLE

```
do 5 N=1,25
```

Cycle till the label 5, the variable N changes from 1 to 25.

```
do 5 N=1.25
```

Blanks are unimportant.

Variables can be undeclared.

```
do5N=1.25
```

Assignment of the variable do5N.

We don’t know if ‘do’ is a keyword without going ahead.

Basics of lexical analysis

Reserved words

C++ example (keywords are reserved):

```
if els
   then
      the=els;
   else
      els=the;
```

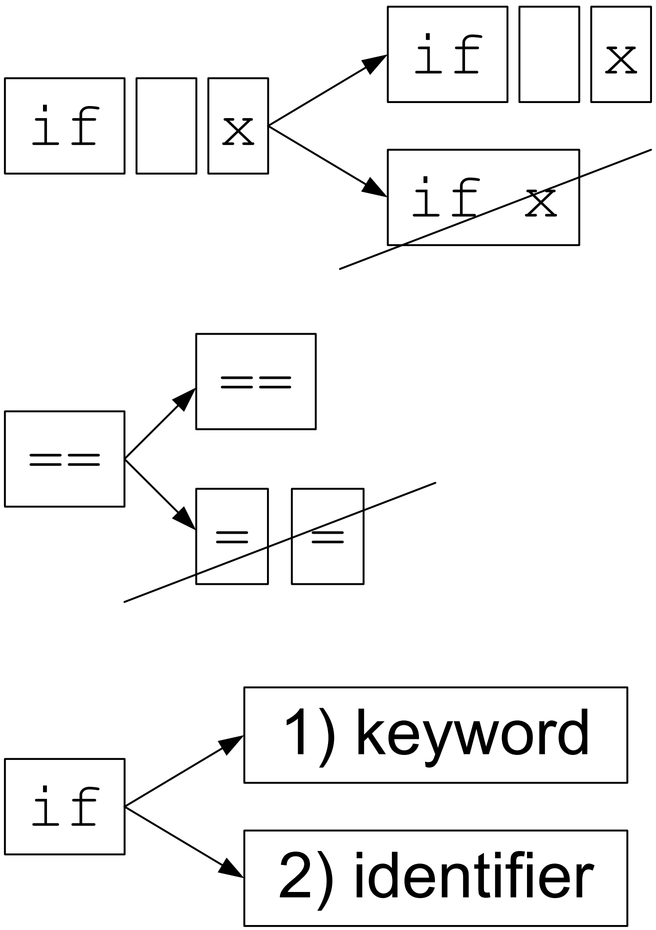
PL/1 example (keywords are not reserved):

```
if else
   then
      then=else
   else
      else=then
```

Basics of lexical analysis

Recognition:

- To read left-to-right recognizing one token at a time;
- To recognize correctly the token class for the considering lexeme;
- To minimize going ahead;
- To check reserved words and to use prioritizing.



2.2. Regular languages and regular expressions

Let X be an alphabet, ε is an empty symbol.

A **string** is any sequence of symbols from X.

X* is the set of all strings. Any $L \subseteq X^*$ is a **language**.

- Union $L1 \cup L2 = \{\alpha: \alpha \in L1 \text{ or } \alpha \in L2\}$
- Concatenation $L1L2 = \{\alpha\beta: \alpha \in L1, \beta \in L2\}$
- Iteration: $L^* = \{\varepsilon\} \cup L \cup LL \cup LLL\ldots$

Example

L1={a,bc}, L2={aa,b,bc}

$L1 \cup L2$={a,bc,aa,b}

L1L2={aaa,ab,abc,bcaa,bcb,bcbc}

L1*={ε,a,bc,aa,abc,bca,bcbc,aaa,aabc,abca,abcbc,bcaa,…}

Regular languages and regular expressions

Regular languages:

- $\varnothing$ is a Rlang;
- $\{\varepsilon\}$ is a Rlang;
- $\{x\}$ for any $x \in X$ is a Rlang;
- If L1 and L2 are Rlangs then $L1 \cup L2$ is a Rlang;
- If L1 and L2 are Rlangs then L1L2 is a Rlang;
- If L is a Rlang then L* is a Rlangs;
- There are no other Rlangs.

Regular languages and regular expressions

Regular expressions:

- $\varnothing$ is a Rexp;
- ε is a Rexp;
- $x \in X$ is a Rexp;
- If R1 and R2 are Rexps then R1+R2 is a Rexp;
- If R1 and R2 are Rexps then R1R2 is a Rexp;
- If R is a Rexp then R* is a Rexp;
- There are no other Rexps.

Rexp	Rlang
$\varnothing$	$\varnothing$
ε	{ε}
x	{x}
R1+R2	$L(R1) \cup L(R2)$
R1R2	L(R1)L(R2)
R*	(L(R))*

Regular languages and regular expressions

Prioritizing:

- Iteration
- Concatenation
- Union

Examples:

- (01+1)*(001+000+0+ε) is a Rexp;
- 0^n1^n: n>0 isn't Rexp.

Regular languages and regular expressions

Laws for Rexps

1) $R + S = S + R$, $R + R = R$, $(R + S) + T = R + (S + T)$, $\emptyset + R = R$;
2) $R\varepsilon = \varepsilon R = R$, $(RS)T = R(ST)$, $\emptyset R = R\emptyset = \emptyset$;
3) $R(S + T) = RS + RT$, $(R + S)T = RT + ST$;
4) $R^* = \varepsilon + R + \ldots + R^k R^*$, $RR^* = R^* R$, $R(SR)^* = (RS)^* R$;
5) $R^* RR^* = RR^*$, $RR^* + \varepsilon = R^*$.

Example

$$b(b + aa^*b) = b(\varepsilon b + aa^*b) = b(\varepsilon + aa^*)b = ba^*b.$$

Regular languages and regular expressions

Extensions of Rexps

- One or more instances: R+=RR*
- Zero or one instance: R?
- Character classes: [a-z]=a+…+z
- Exclusion: [^a-z] – everything except [a-z]
- Any symbol except eof: .

2.3. Finite automata

Regular expression = specification

Finite automata = implementation

Finite automata A=<X,Q,q0,F,Ψ>:

- $X \neq \varnothing$ – an input alphabet;
- Q – a set of states;
- $q0 \in Q$ – a start state;
- $F \subseteq Q$ – a set of accepting (final) states;
- $\Psi: X \times Q \rightarrow Q$ – a transition function.

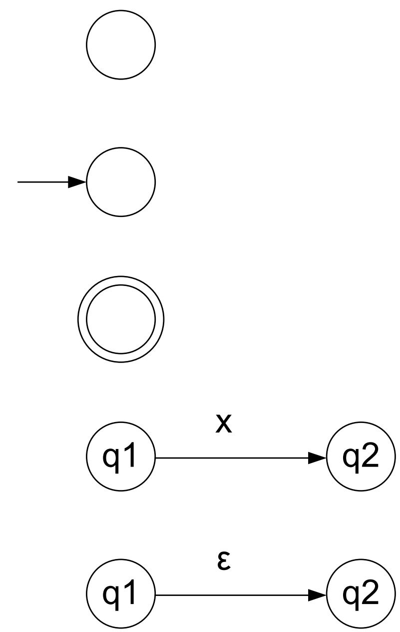
state

start state

accepting state

transition

ε-transition



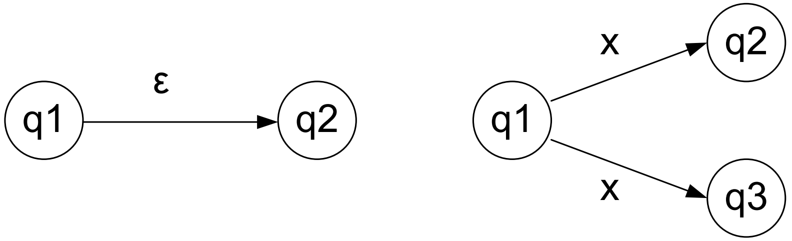
Finite automata

Deterministic finite automata (DFA):

- no ε-transitions;
- one transition per one pair 'symbol-state'.

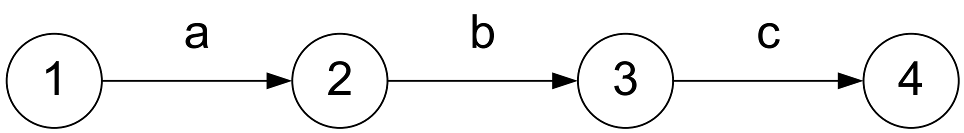
Nondeterministic finite automata (NFA):

- ε-transitions are allowed;
- multiply transitions per one pair 'symbol-state' are allowed.

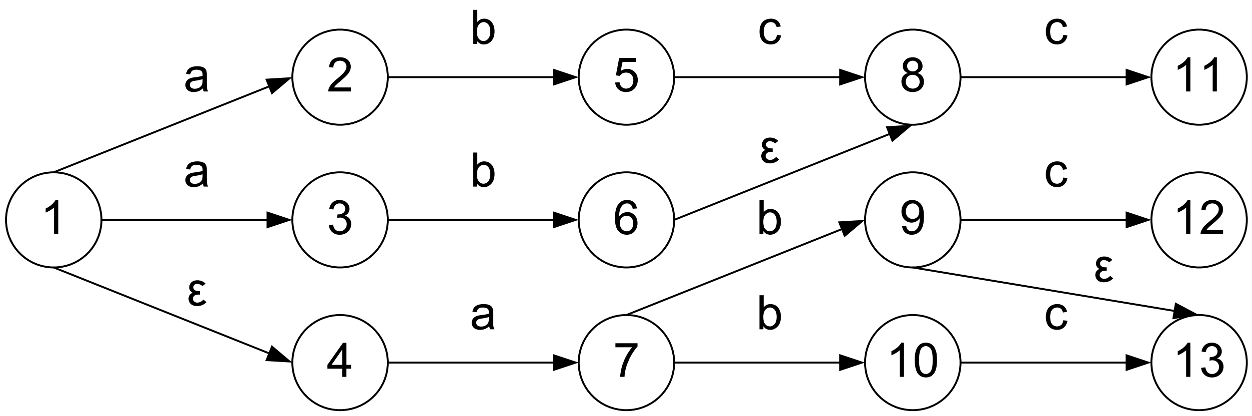


Finite automata

- DFA (one path for one string)



- NFA (several paths)



Finite automata

The finite automata A **accepts** the string α=a1a2…an if there is a path in the automata diagram from the start state to one of accepting states where arches are marked by the symbols a1, a2,…,an and probably by ε.

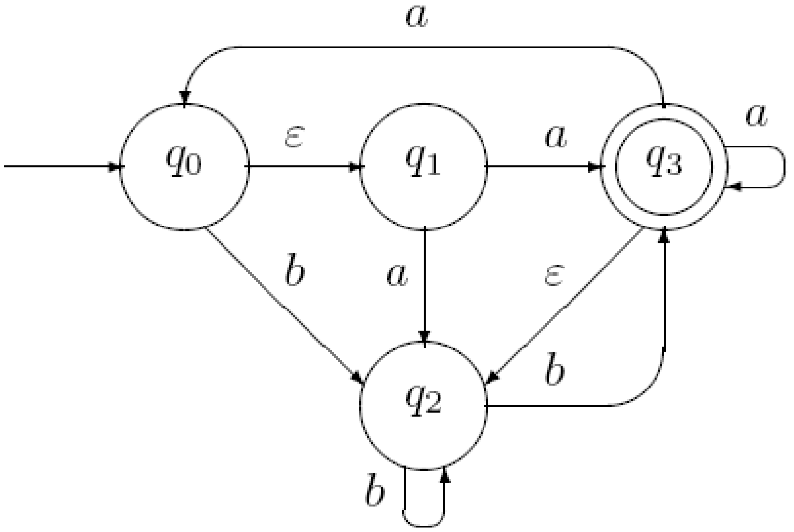
The set of accepted strings form the **accepted language L(A).**

Languages accepted by FA are called **automata languages.**

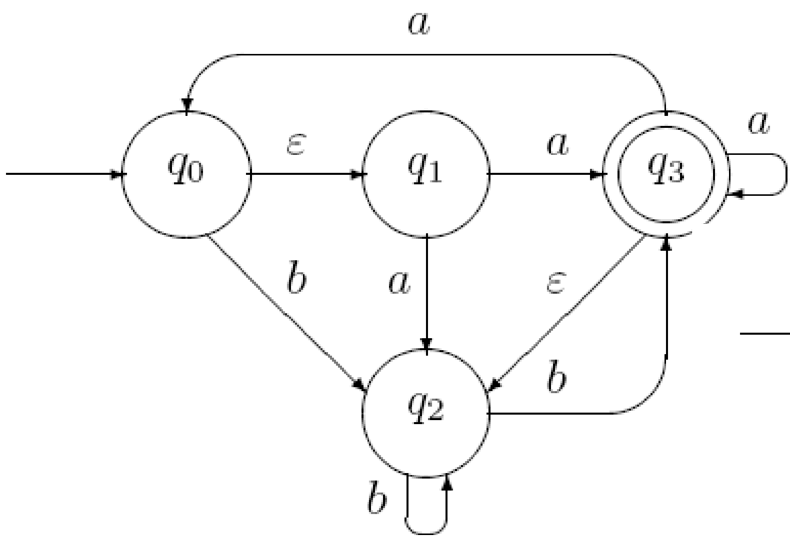
Example

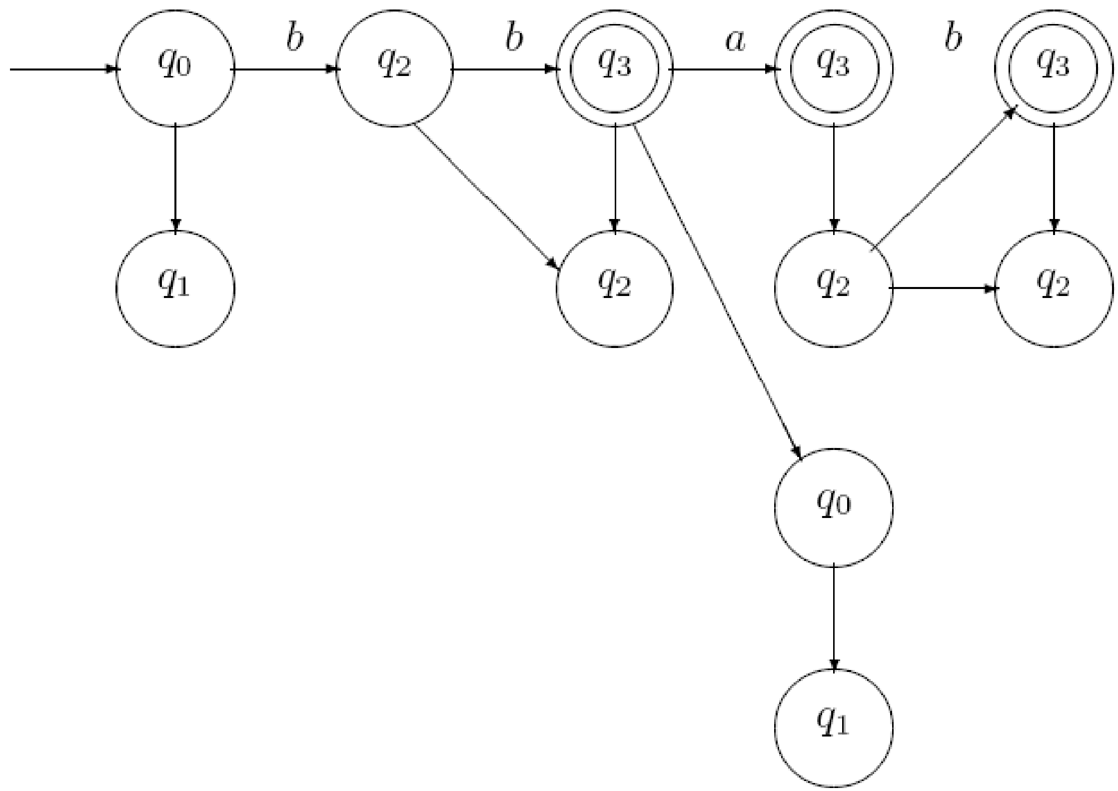
a, aa, ab, bbb are accepted

b is not accepted

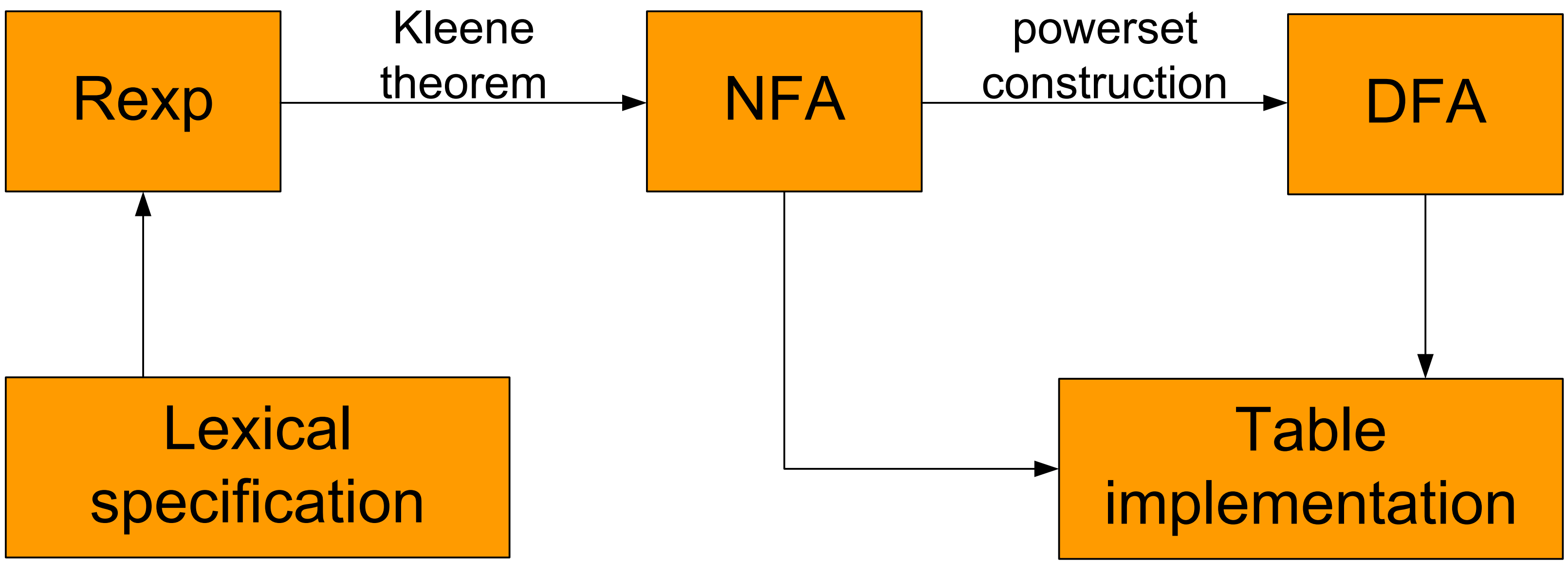


Finite automata





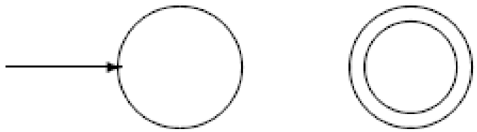
Finite automata



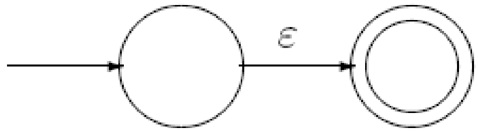
2.4. Rexp into NFA

Kleene theorem. For every regular expression R, we can construct a DFA accepting the same language.

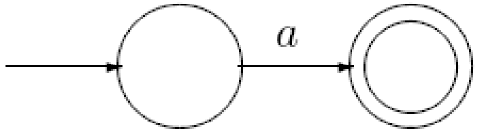
- R=∅



- R=ε

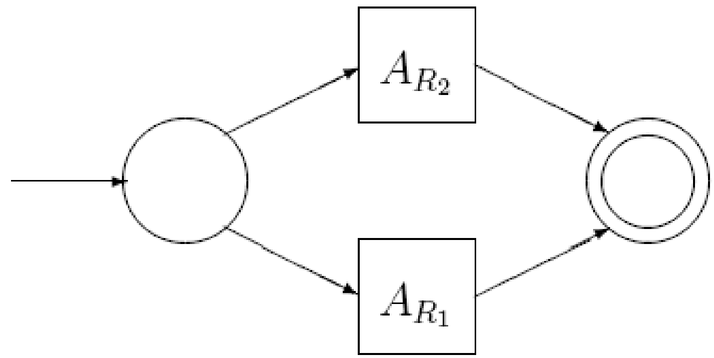


- R=a

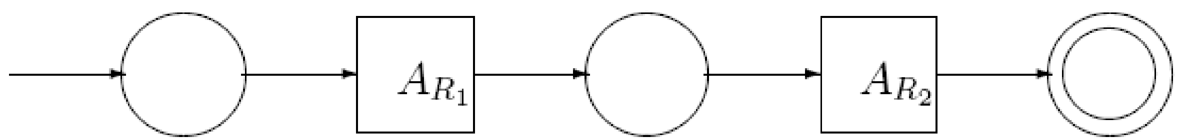


Rexp into NFA

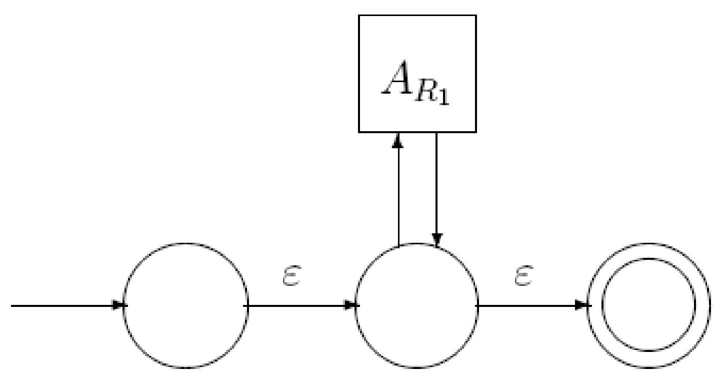
- R=R1+R2



- R=R1R2

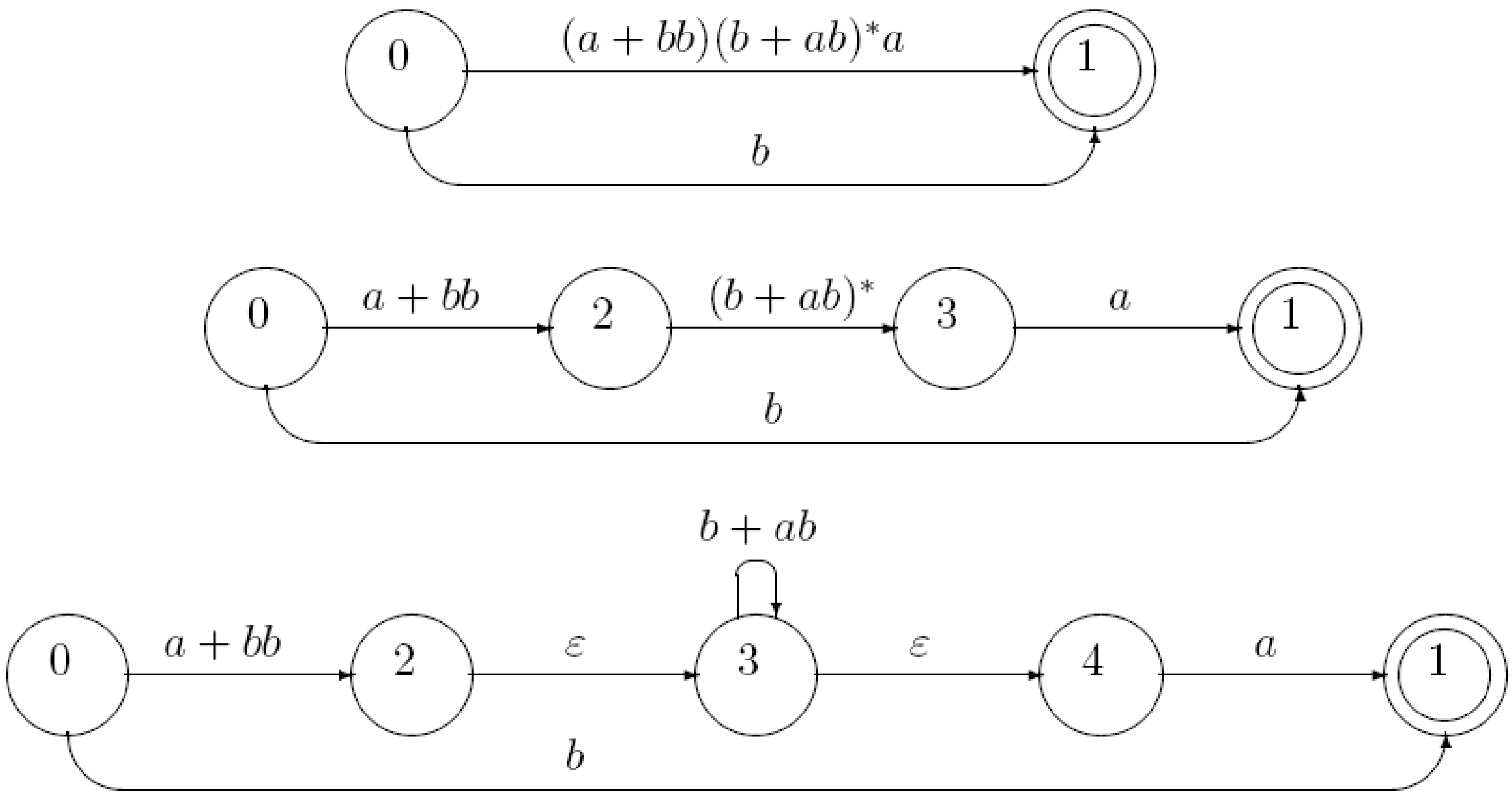


- R=R1*

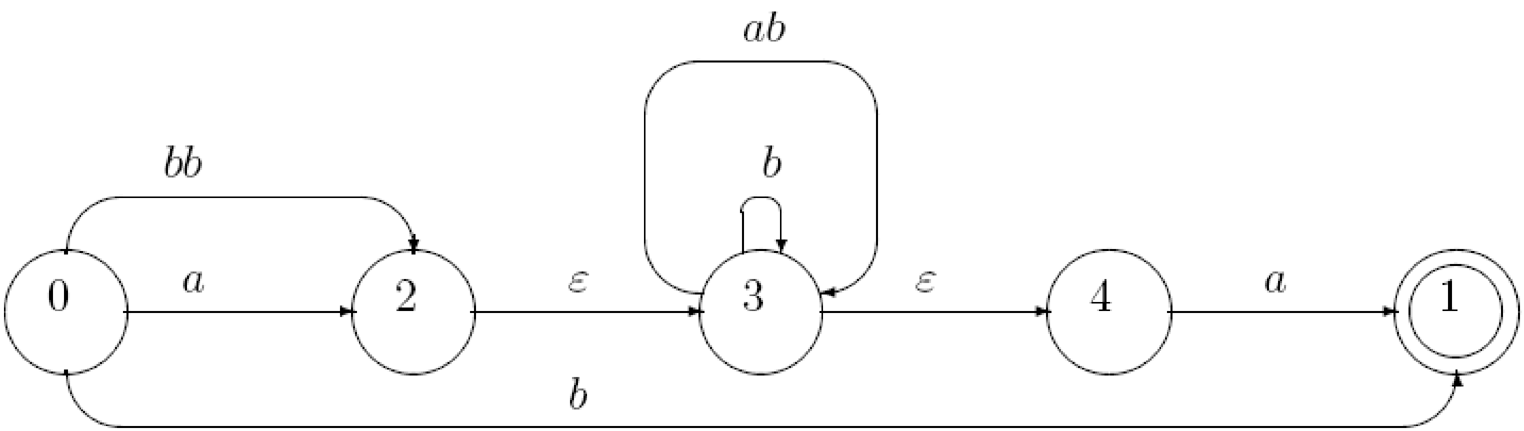


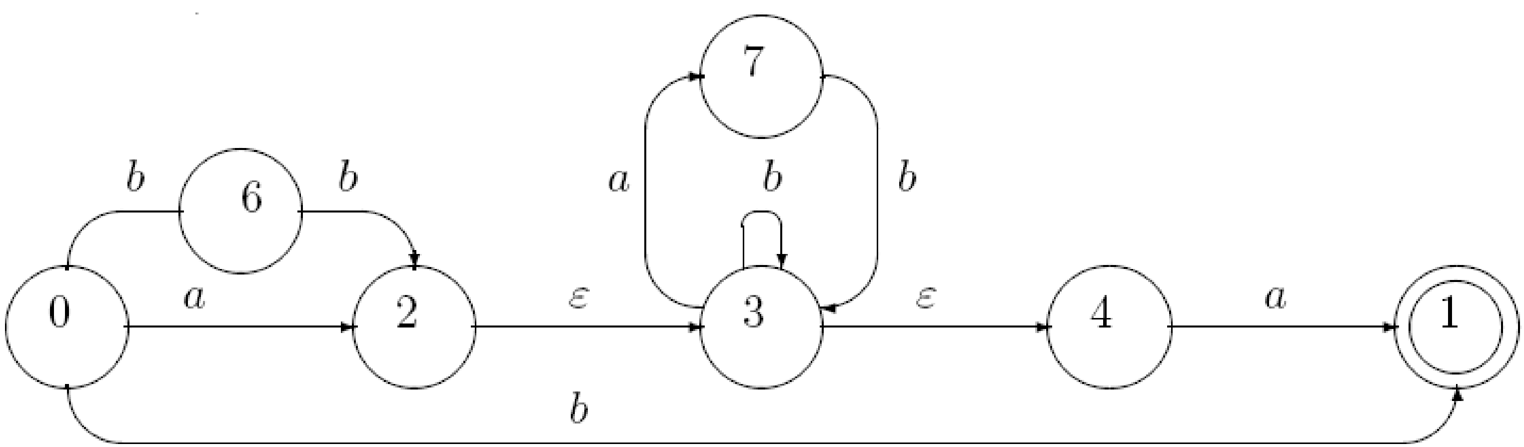
Rexp into NFA

Example. $R = b + (a + bb)(b + ab)^*a$

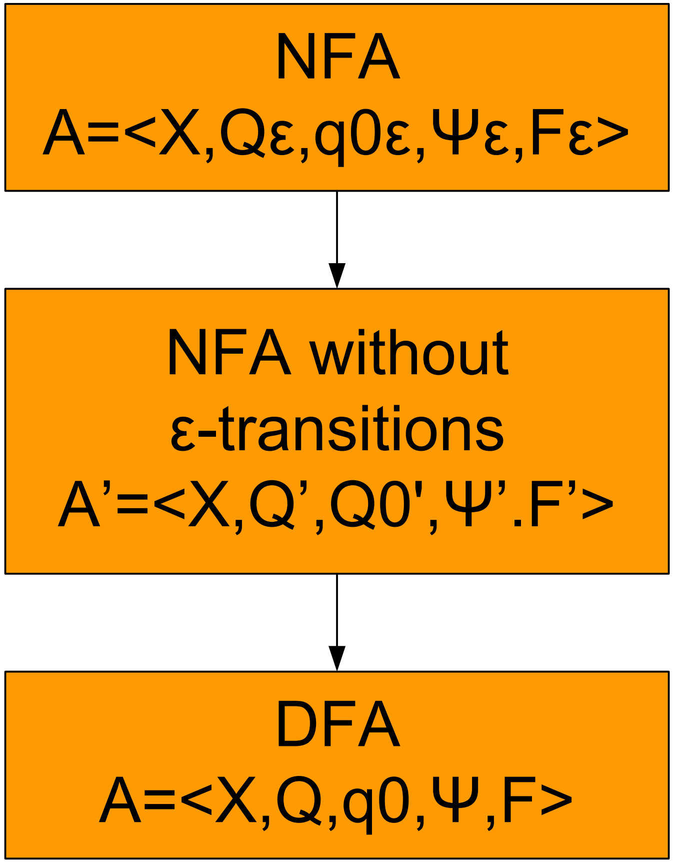


Rexp into NFA





2.5. NFA into DFA



NFA into DFA

ε-closure of the state q:

$$[q]=\{p: p=\Psi\varepsilon(\varepsilon,q)\}$$

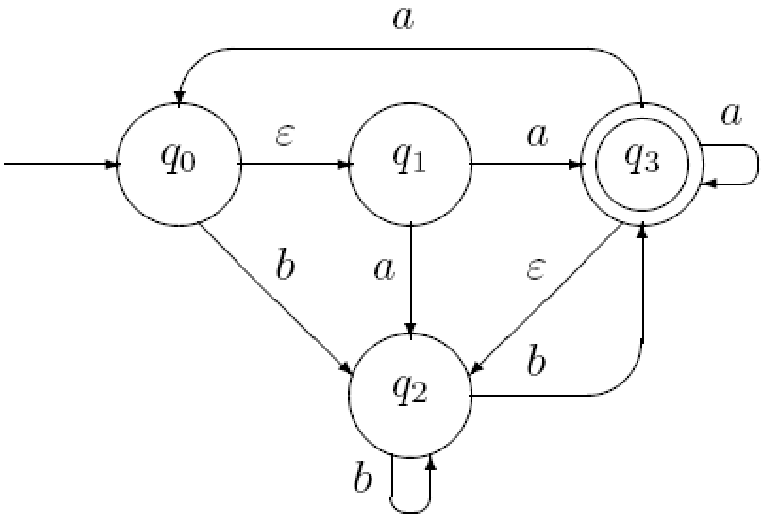
Example

[q0]={q0,q1}

[q1]={q1}

[q2]={q2}

[q3]={q3,q2}



NFA into DFA

Q'={[p]: p∈Qε}

Q0'={[p]: p∈[q0ε]}

F0'={[p]: [p]∩Fε≠∅}

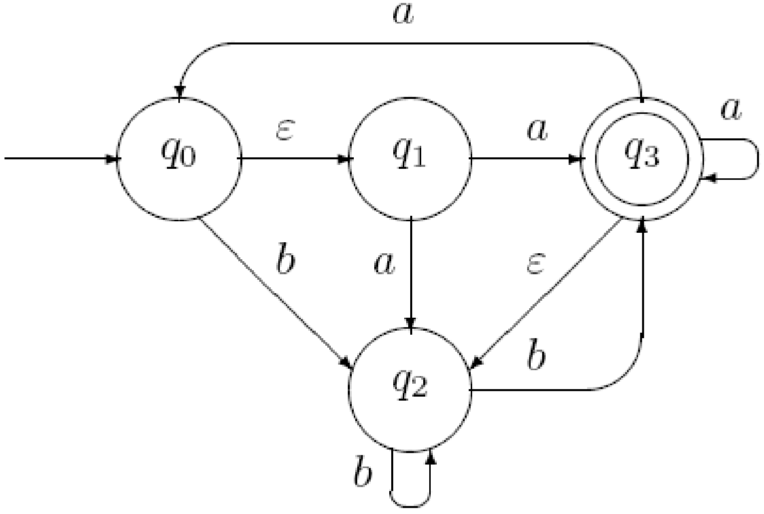
Example

[q0]={[q0],[q1]}

[q1]={q1}

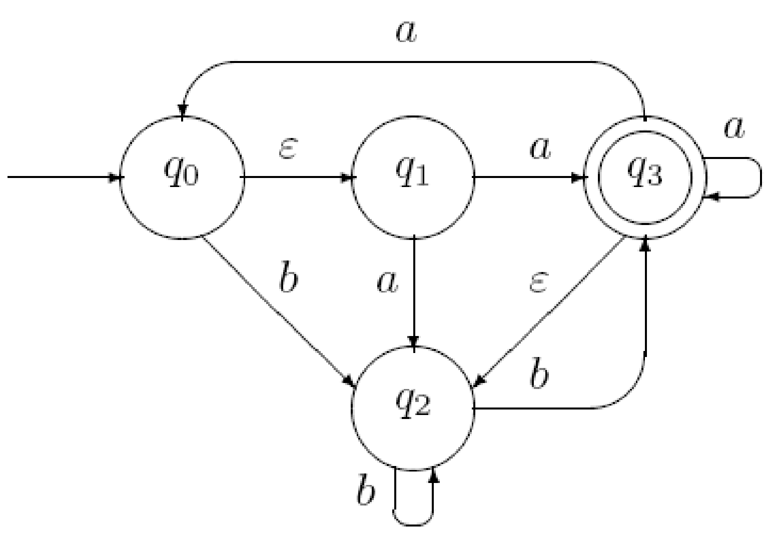
[q2]={q2}

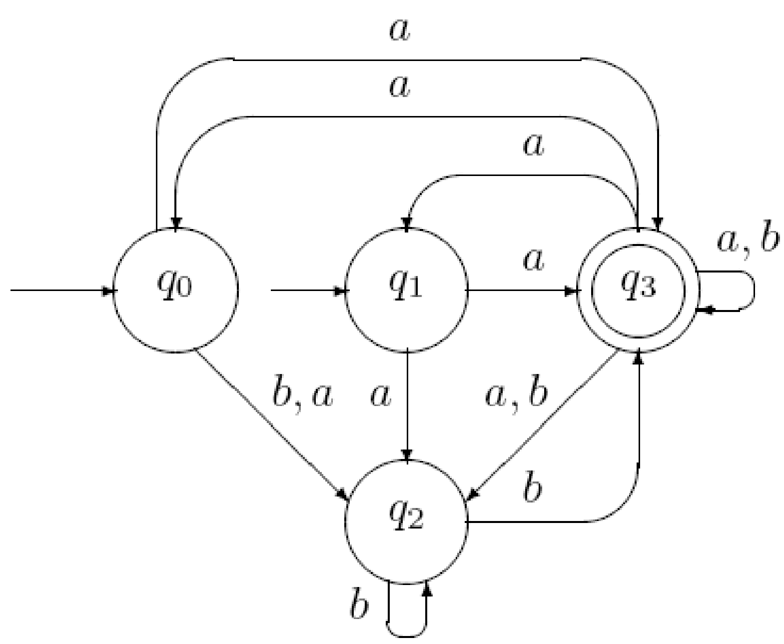
[q3]={q3,q2}



NFA into DFA

Ψ'(x,[p])={ [s]: s∈{ [Ψε(x,q)], q ∈[p] } }





NFA into DFA

$Q=2^{Q'}$

$q0=Q0'$

$F=\{ P\in 2^{Q'}: P\cap F'\neq\emptyset \}$

$\Psi(x\ P)=\{\Psi'(x,p): p \in P\}$

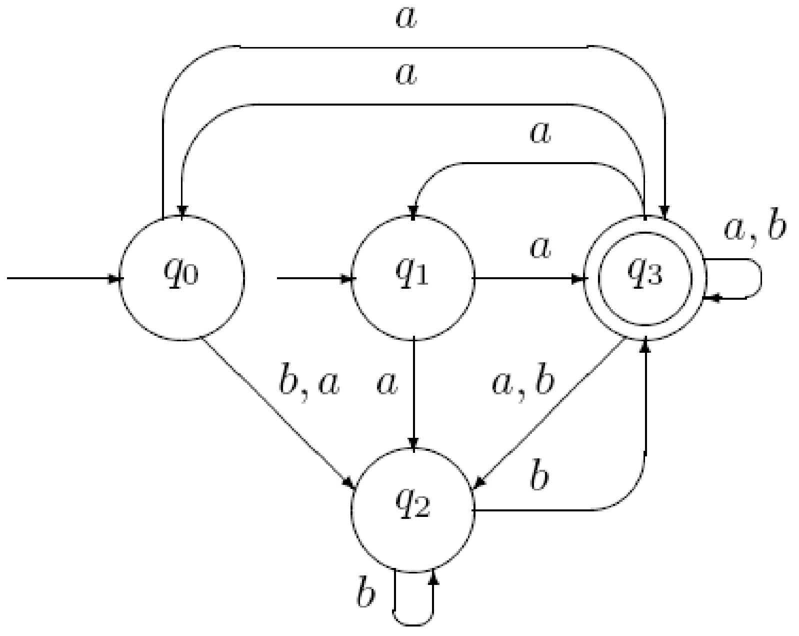
Example

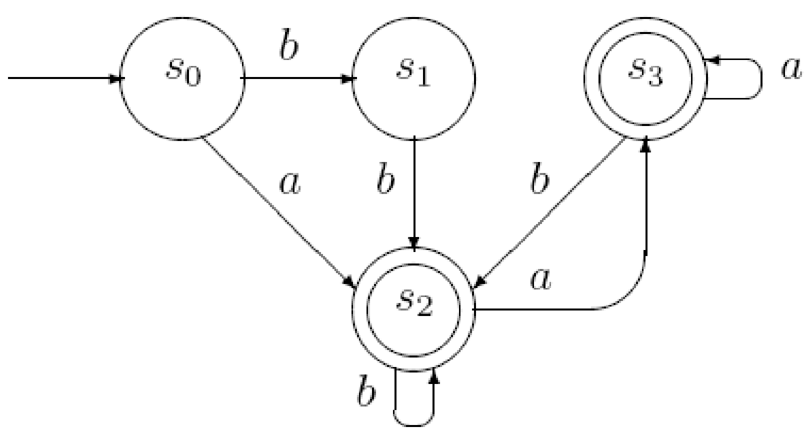
s0={q0,q1}

s1={q2}

s2={q2,q3}

s3={q0,q1,q2,q3}





NFA into DFA

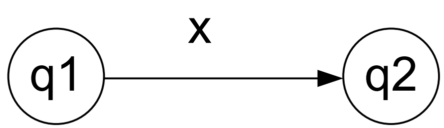
DFA:

- faster to execute (one transition per pair symbol-state);
- bigger number of states (probably exponentially).

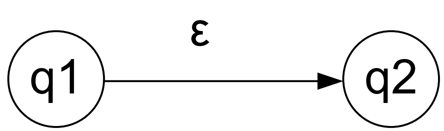
NFA:

- smaller number of states (probably exponentially);
- slower to execute (more then one transition).

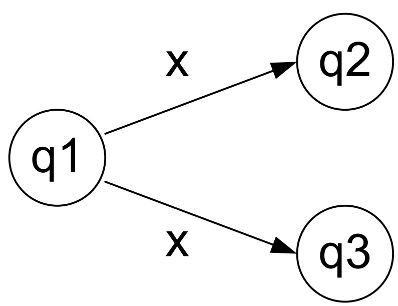
2.6. Table implementation of FA



	q1
x	q2



	q1
ε	q2



	q1
x	{q2, q3}

2.7. Lexical errors

Lexical errors include misspellings of identifiers, keywords or operators, e.g.

- ===
- 8abcdef
- ifff
- …

If A is a FA and $\alpha \notin L(A)$ then α contains a lexical error.