Lexical Analysis

Lecture 03

Example or Regular Expression / Regular Definition:

Regular Expression for numbers

digit $\rightarrow$ 0|1|…|9

digits $\rightarrow$ digit digit*

optional_fraction $\rightarrow$ .digits|ε

optional_exponent $\rightarrow$ (E (+|-| ε) digits) | ε

num $\rightarrow$ digits optional_fraction optional_exponent

Using shorthands:

digit $\rightarrow$ 0|1|…|9

digits $\rightarrow$ digit$^+$

optional_fraction $\rightarrow$ (.digits)?

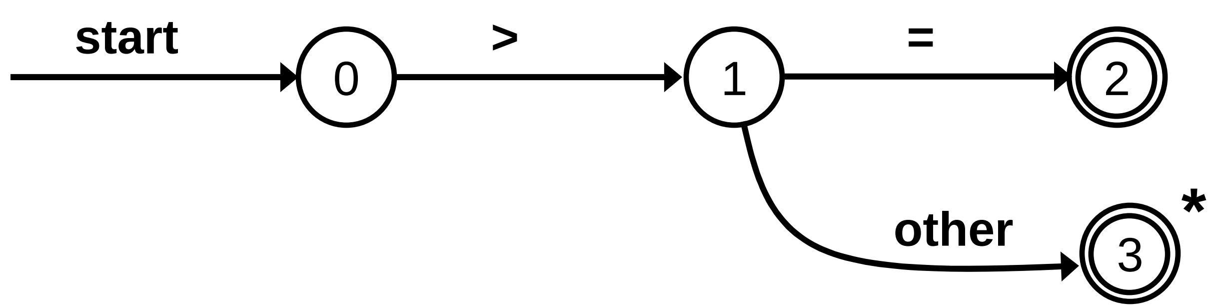
optional_exponent $\rightarrow$ (E (+|-| ε) digits) ?

num $\rightarrow$ digits optional_fraction optional_exponent

Transition Diagram

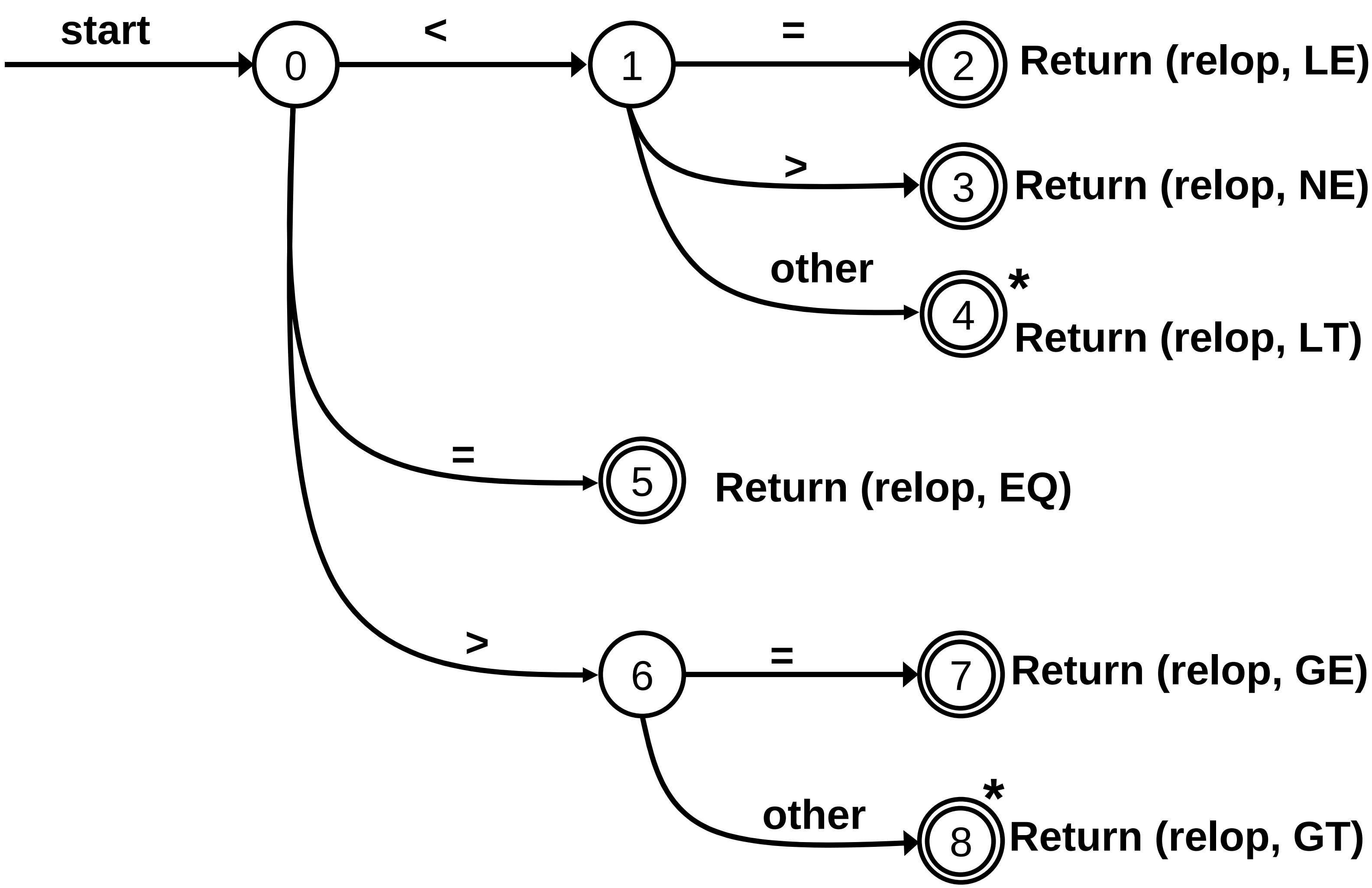
- A stylized flowchart produced intermediately in the construction of lexical analyzer
- Depicts the actions take place in lexical analyzer

- ***states***: positions in a transition diagram
- ***edges***: arrows connecting the states
- ***start state***: initial state of transition diagram
- ***accepting state***: token recognized
- ***action***: (optional) associated with a state that is executed when the state is entered

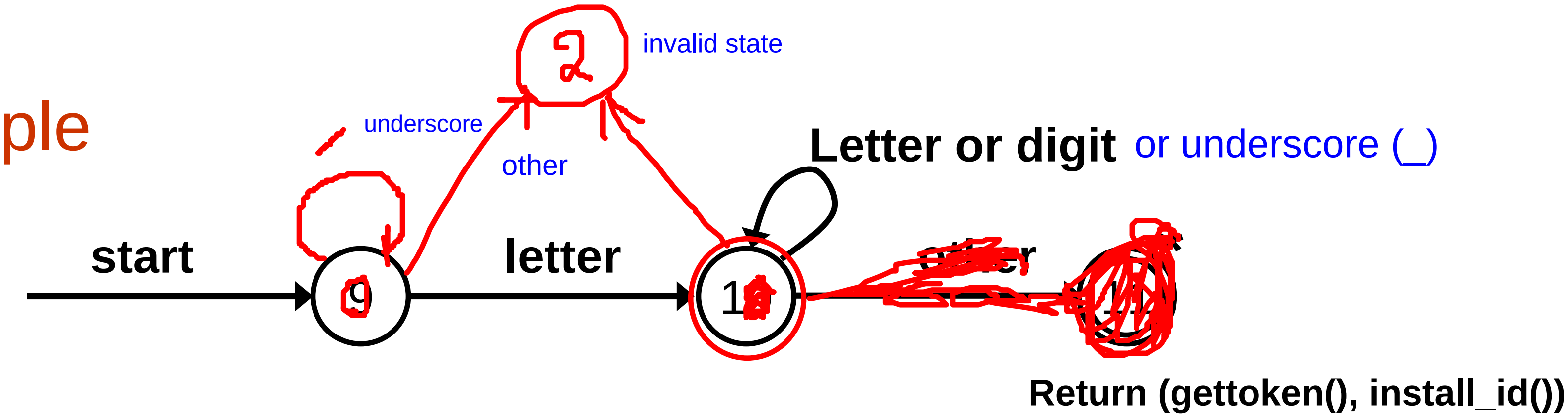


Example

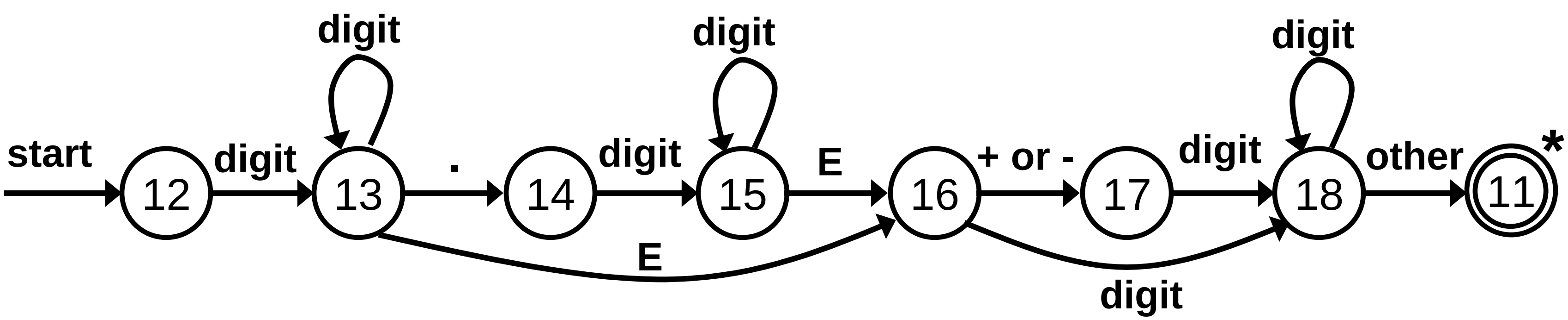
- Transition diagram for token **relop**



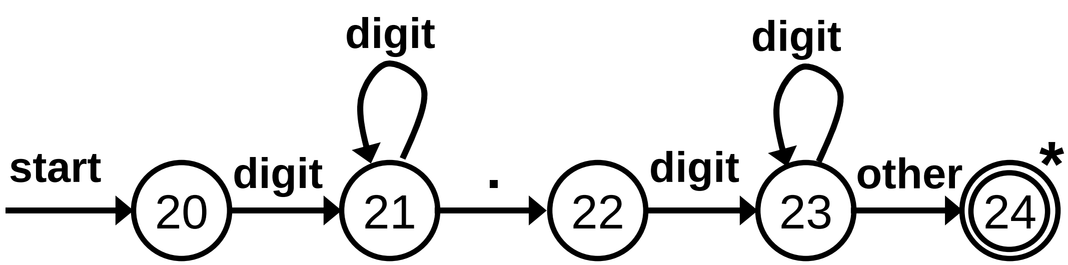
Example



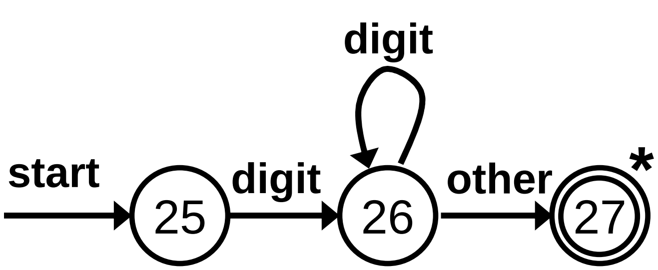
invalid state
2
underscore
other
Letter or digit
or underscore (_)
start
letter
9
Return (gettoken(), install_id())



start
digit
12
13
.
14
15
E
16
+ or -
17
18
other
11
*



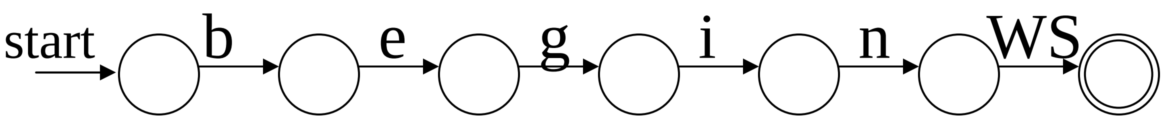
start
digit
20
21
.
22
23
other
24
*



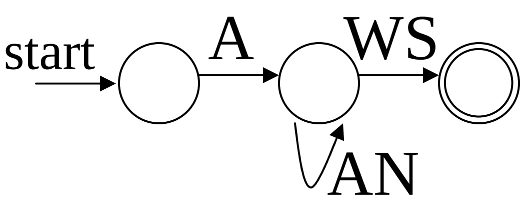
start
digit
25
26
other
27
*

Capturing Multiple Tokens

Capturing keyword “begin”



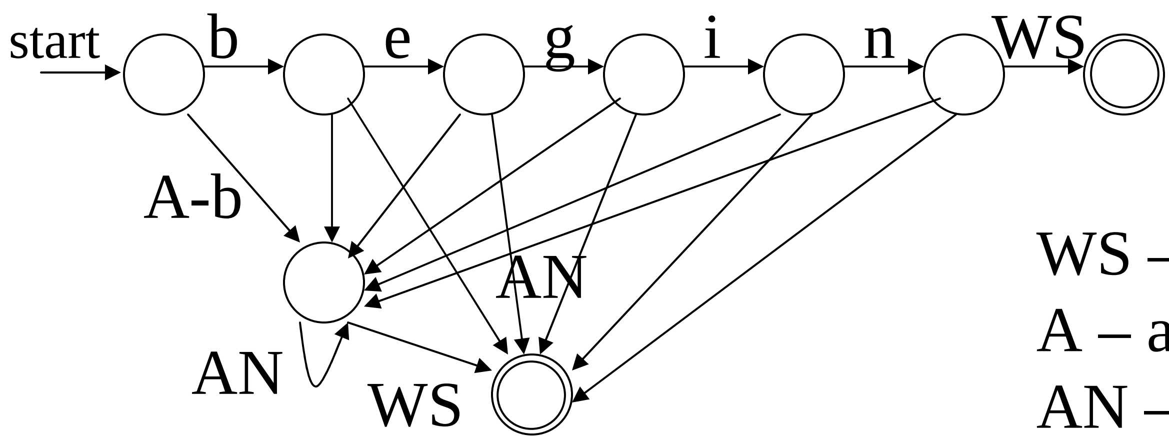
Capturing variable names



WS – white space
A – alphabetic
AN – alphanumeric

What if both need to happen at the same time?

Capturing Multiple Tokens



WS – white space
A – alphabetic
AN – alphanumeric

Machine is much more complicated – just for these two tokens!

Implementing a Transition Diagram

- Systematic approach for all transition diagrams
  - Program size $\propto$ number of states and edges
- We try each diagram and when we fail then we go to try the next diagram

- Transition diagram for WS should be placed at the beginning rather at the end
  - Generalize: frequently occurring tokens should come earlier

Finite State Automata (FSAs)

- **AKA “Finite State Machines”, “Finite Automata”, “FA”**

- One start state
- Many final states
- Each state is labeled with a state name
- Directed edges, labeled with symbols
- Two types
  - Deterministic (DFA)
  - Non-deterministic (NFA)

Nondeterministic Finite Automata

A **nondeterministic finite automaton** (NFA) is a mathematical model that consists of

1. A set of states S
   - $S = \{s_0, s_1, \ldots, s_N\}$
2. A set of input symbols Σ
   - $\Sigma = \{a, b, \ldots\}$
3. A transition function that maps state/symbol pairs to a set of states:

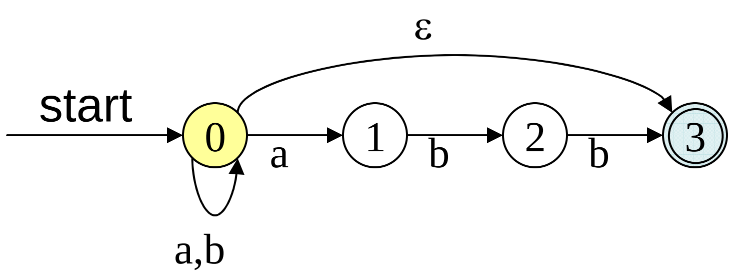
   $$\mathbf{S \times \{\Sigma + \varepsilon\} \rightarrow \text{set of } S}$$

4. A special state s_0 called the start state
5. A set of states F (subset of S) of final states

INPUT: string

OUTPUT: yes or no

Example: NFA



$S = \{ 0,1,2,3\}$
$S_0 = 0$
$\Sigma = \{a,b\}$
$F = \{3\}$

Transition Table:

STATE	a	b	ε
0	0,1	0	3
1		2	
2		3	
3			

Deterministic Finite Automata

A **deterministic finite automaton** (DFA) is a mathematical model that consists of

1. A set of states S
   - $S = \{s_0, s_1, \ldots, s_N\}$
2. A set of input symbols Σ
   - $\Sigma = \{a, b, \ldots\}$
3. A transition function that maps state/symbol pairs to a state:

   $\mathbf{S \times \Sigma \rightarrow S}$
4. A special state s_0 called the start state
5. A set of states F (subset of S) of final states

INPUT: string

OUTPUT: yes or no

DFA Execution

```
DFA(int start_state) {
   state current = start_state;
   input_element = next_token();
   while (input to be processed) {
       current = transition(current,table[input_element])
       if current is an error state return No;
       input_element = next_token();
    }
    if current is a final state return Yes;
    else return No;
}
```

Relation between RE, NFA and DFA

1. There is an algorithm for converting any RE into an NFA.
2. There is an algorithm for converting any NFA to a DFA.
3. There is an algorithm for converting any DFA to a RE.

These facts tell us that REs, NFAs and DFAs have equivalent expressive power.

All three describe the class of regular languages.

DFA vs NFA

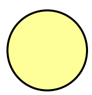
- Both DFA and NFA are the recognizers of regular sets.
- But – time-space trade space exists
- DFAs are faster recognizers
  - Can be much bigger too..

Converting Regular Expressions to NFAs

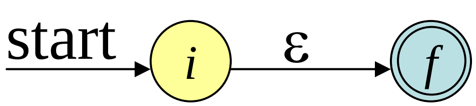
Thompson's Construction

The **regular expressions** over finite Σ are the strings over the alphabet Σ + {), (, |, *} such that:

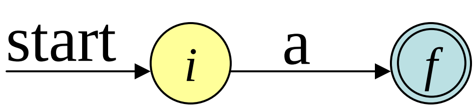
- { } (empty set) is a regular expression for the empty set



- Empty string ε is a regular expression denoting { ε }



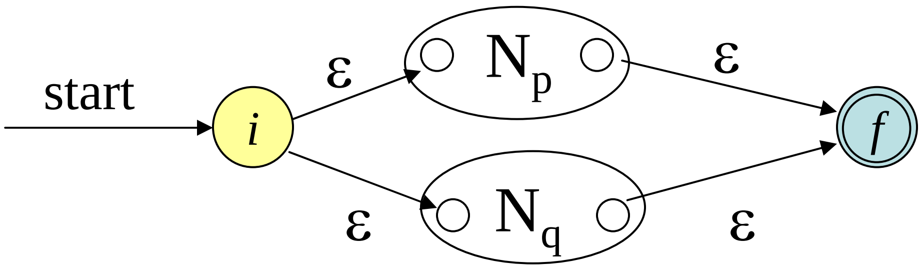
- *a* is a regular expression denoting {*a* } for any *a* in Σ



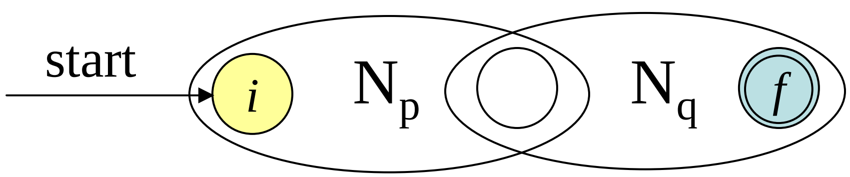
Converting Regular Expressions to NFAs

If P and Q are regular expressions with NFAs N_p, N_q:

P | Q (union)



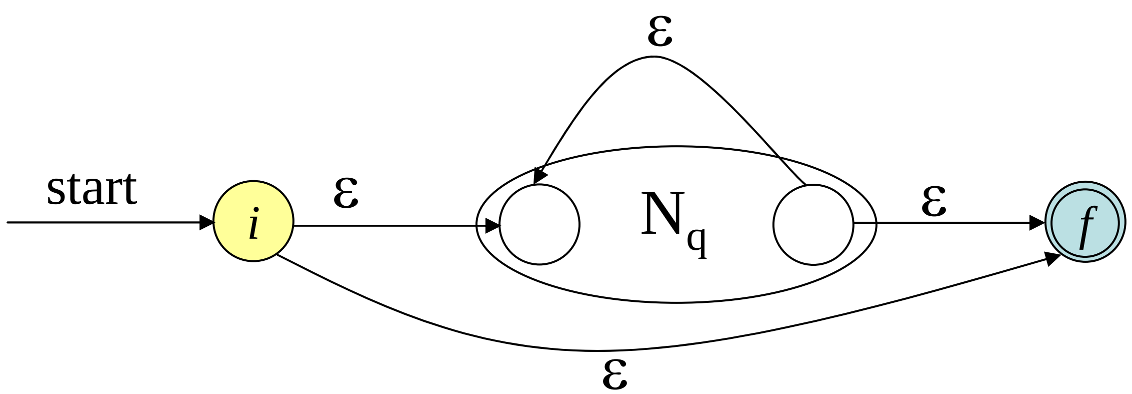
PQ (concatenation)



Converting Regular Expressions to NFAs

If Q is a regular expression with NFA N_q:

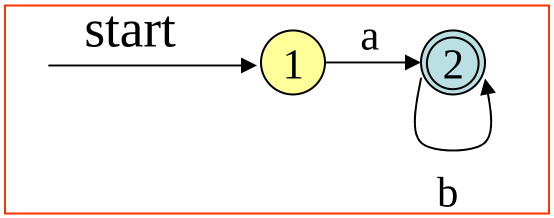
Q* (closure)



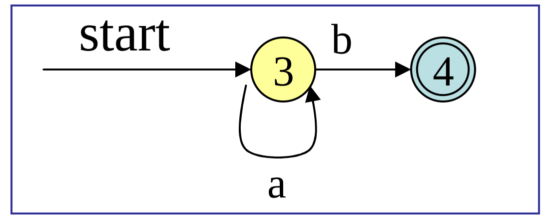
Example (ab* | a*b)*

Starting with:

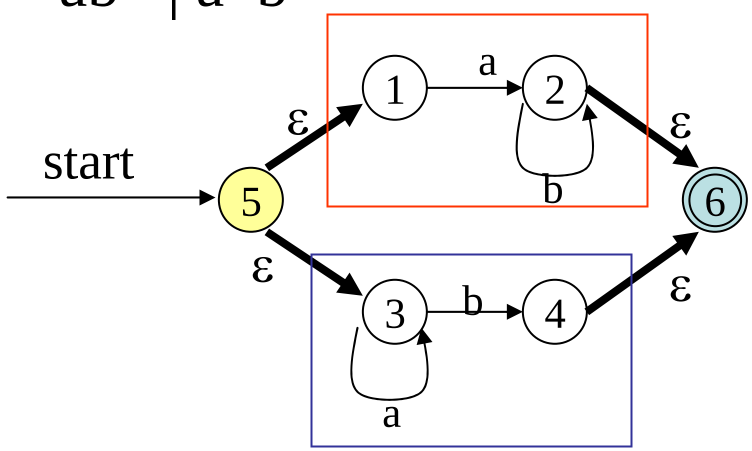
ab*



a*b

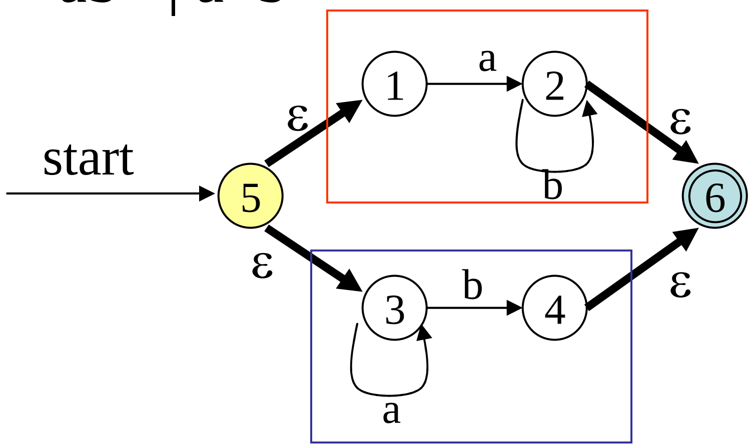


ab* | a*b

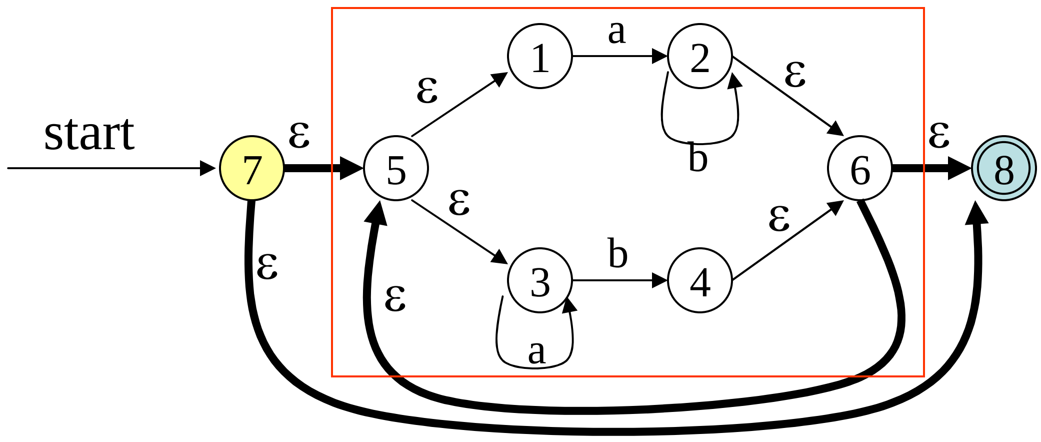


Example (ab* | a*b)*

ab* | a*b

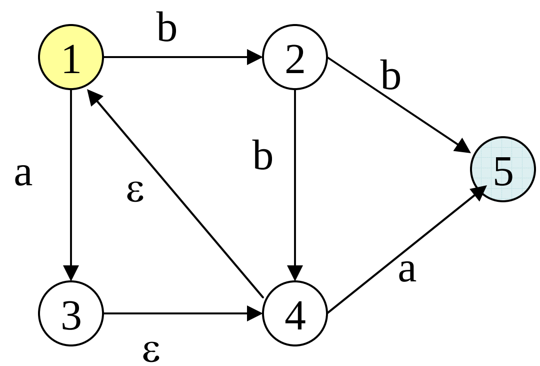


(ab* | a*b)*



Terminology: ε-closure

Defn: ε-closure(T) = T + all NFA states reachable from any state in T using only ε transitions.



ε-closure({1,2,5}) = {1,2,5}
ε-closure({4}) = {1,4}
ε-closure({3}) = {1,3,4}
ε-closure({3,5}) = {1,3,4,5}

Converting NFAs to DFAs (subset construction)

- **Idea**: Each state in the new DFA will correspond to some set of states from the NFA. The DFA will be in state $\{s_0, s_1, \ldots\}$ after input if the NFA could be in *any* of these states for the same input.

- **Input**: NFA N with state set S_N, alphabet Σ, start state s_N, final states F_N, transition function T_N: $S_N \times \{\Sigma \cup \varepsilon\} \rightarrow S_N$
- **Output**: DFA D with state set S_D, alphabet Σ, start state $s_D = \varepsilon\text{-closure}(s_N)$, final states F_D, transition function T_D: $S_D \times \Sigma \rightarrow S_D$

Algorithm: Computation of ε-closure

push all states a ∈ T onto stack STK

initialize: ε-closure(T) = T

while STK is not empty **do begin**

    pop *t*, the top element, off STK

    for each stat *u* with and edge from *t* to *u* labeled ε **do begin**

        if *u* is not in ε-closure(T) **do begin**

            add *u* to ε-closure(T)

            push *u* onto STK

        end if

    end for

end while

Algorithm: Subset Construction

$s_D = \varepsilon$-closure(s_N) -- create start state for DFA

$S_D = \{s_D\}$ (unmarked)

while there is some unmarked state R in S_D

  mark state R

  for all a in Σ do

    s = ε-closure(T_N(R,a));

    if s not already in S_D then add it (unmarked)

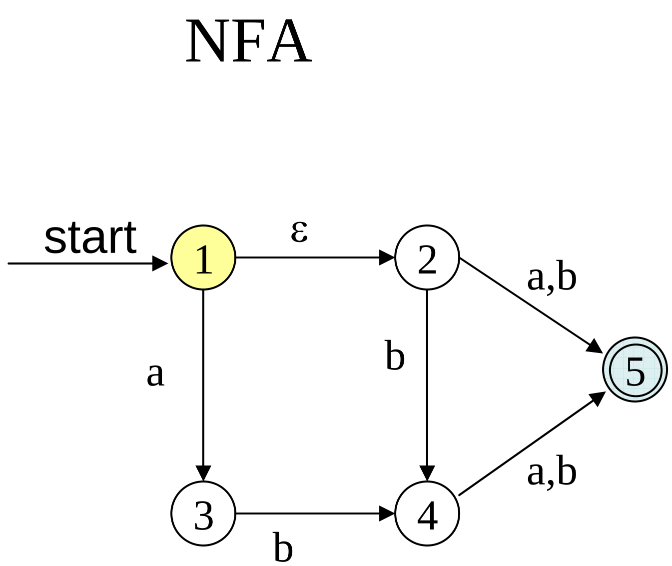
    T_D(R,a) = s;

  end for

end while

F_D = any element of S_D that contains a state in F_N

Example 1: Subset Construction



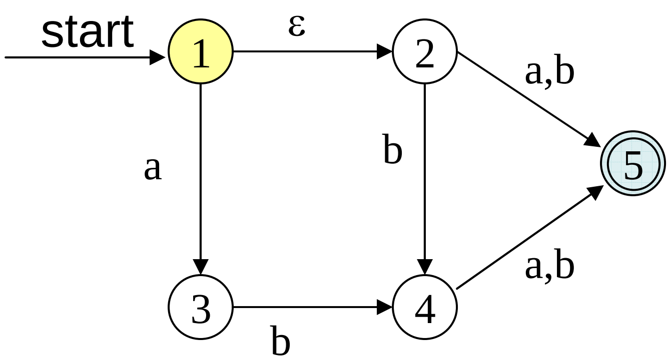
NFA N with

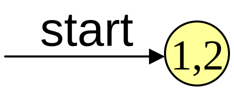
- State set S_N = {1,2,3,4,5},
- Alphabet Σ = {a,b}
- Start state s_N=1,
- Final states F_N={5},
- Transition function T_N: $S_N \times \{\Sigma \cup \varepsilon\} \rightarrow S_N$

	a	b	ε
1	3	-	2
2	5	5, 4	-
3	-	4	-
4	5	5	-
5	-	-	-

Example 1: Subset Construction

NFA

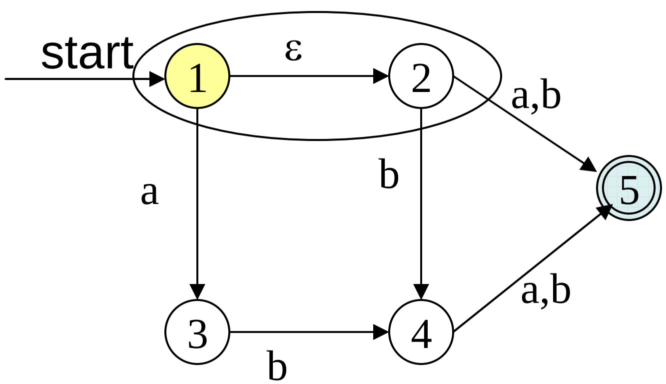


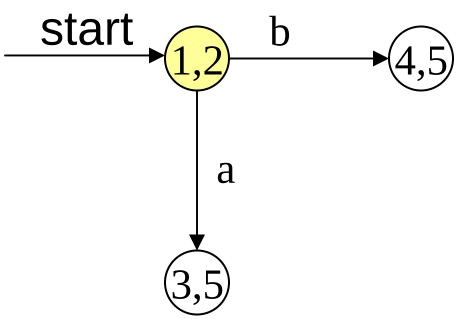


	a	b
{1,2}		

Example 1: Subset Construction

NFA

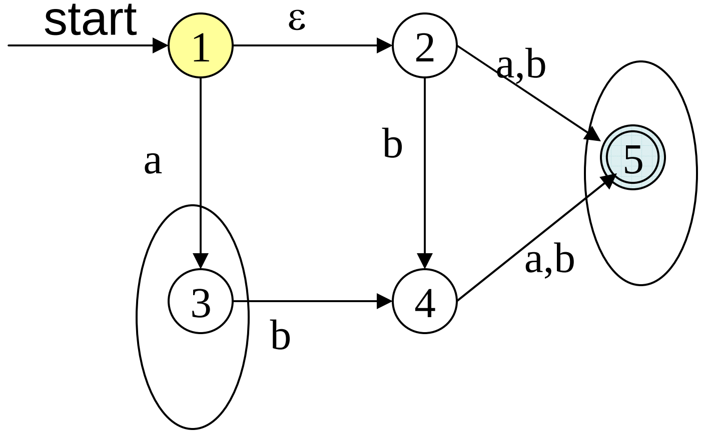


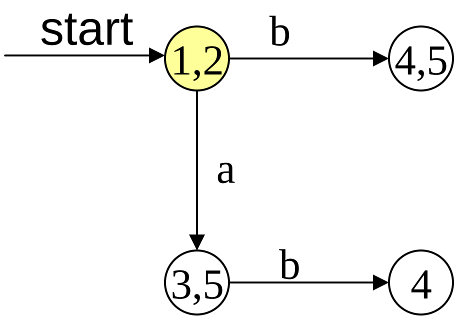


	a	b
{1,2}	{3,5}	{4,5}
{3,5}		
{4,5}		

Example 1: Subset Construction

NFA

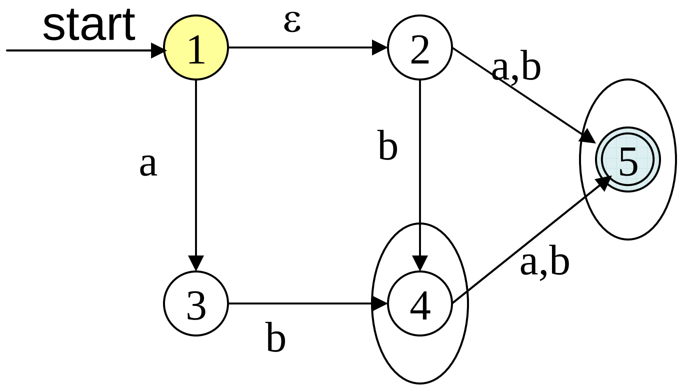


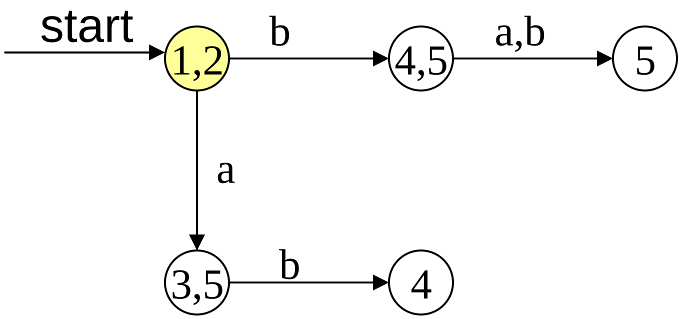


	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}		
{4}		

Example 1: Subset Construction

NFA

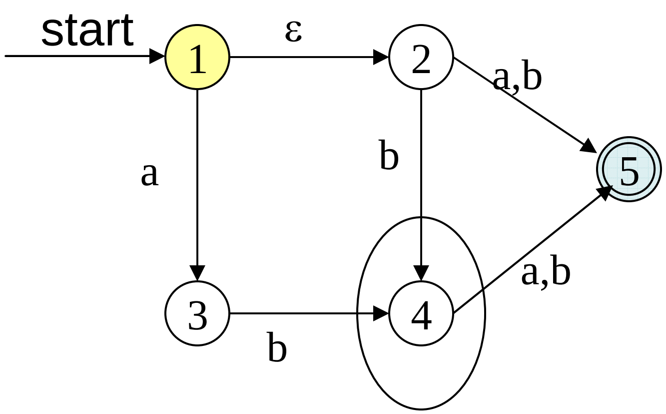


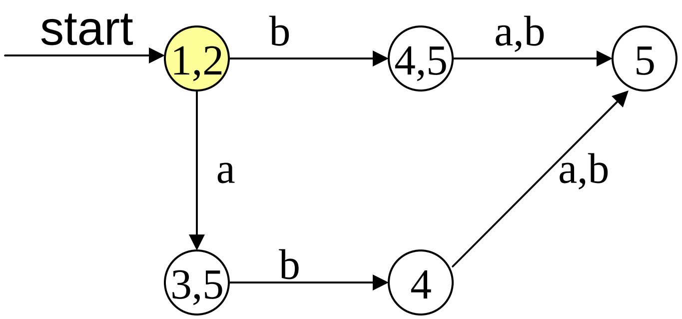


	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}	{5}	{5}
{4}		
{5}		

Example 1: Subset Construction

NFA

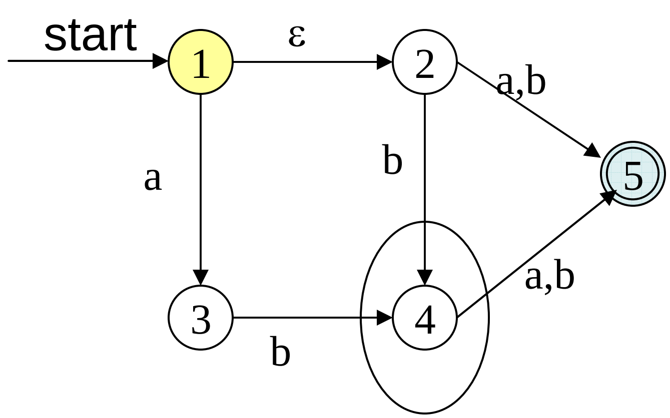


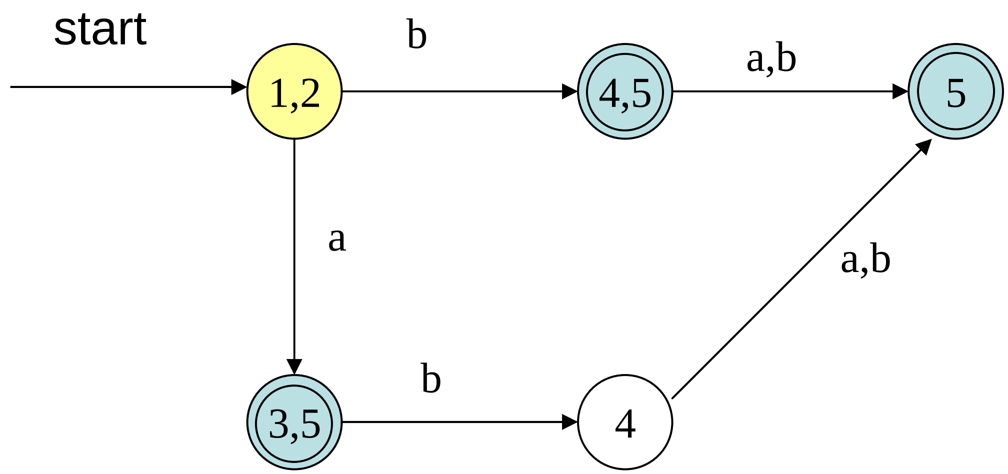


	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}	{5}	{5}
{4}	{5}	{5}
{5}	-	-

Example 1: Subset Construction

NFA



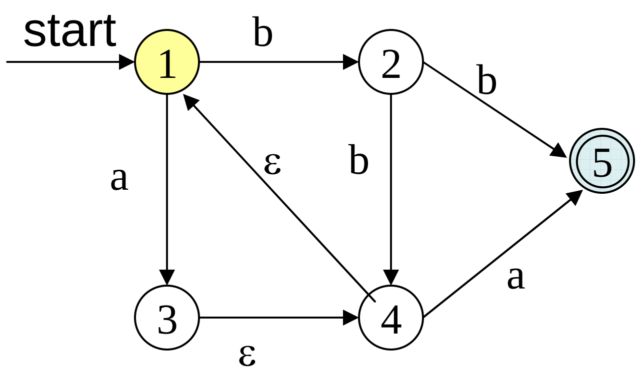


	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}	{5}	{5}
{4}	{5}	{5}
{5}	-	-

All final states since the
NFA final state is included

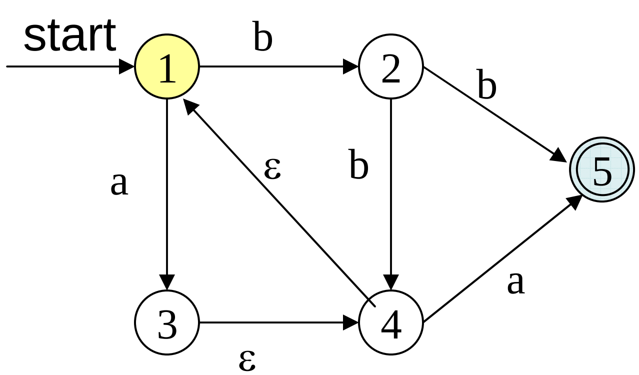
Example 2: Subset Construction

NFA

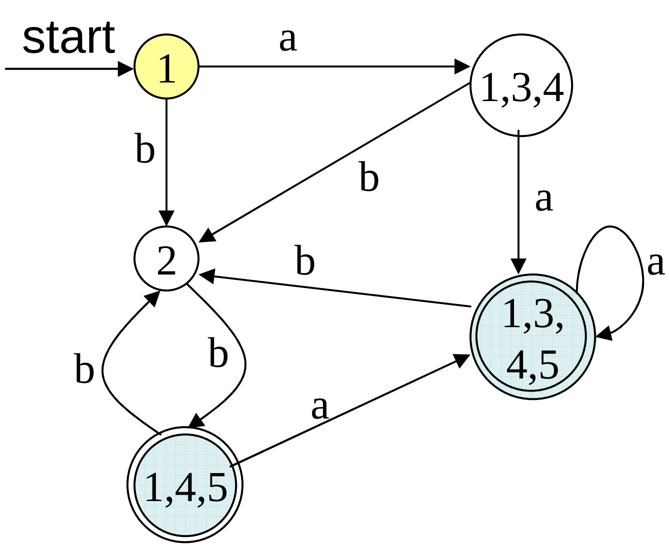


Example 2: Subset Construction

NFA

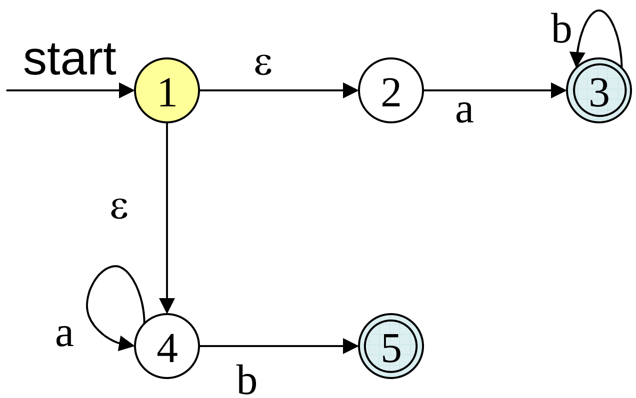


DFA

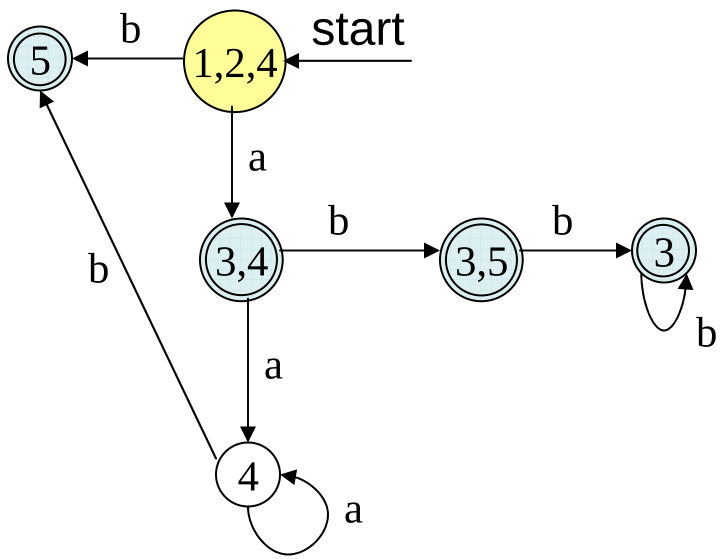


Example 3: Subset Construction

NFA



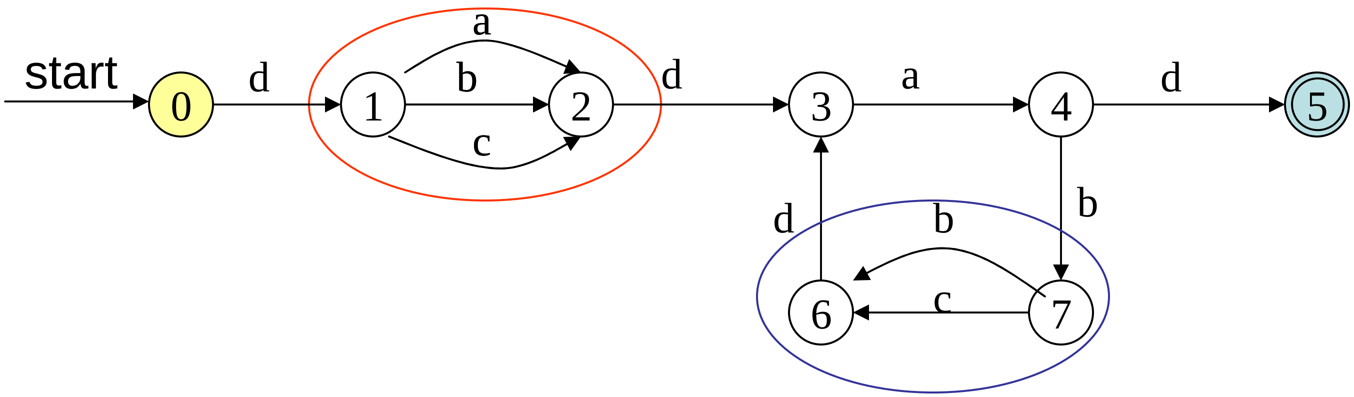
DFA



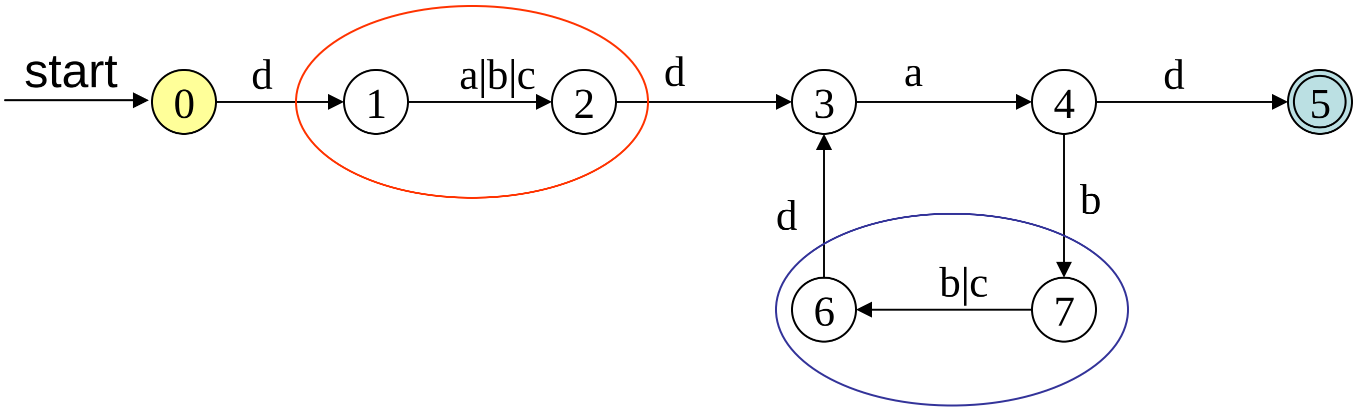
Converting DFAs to REs

1. Combine serial links by concatenation
2. Combine parallel links by alternation
3. Remove self-loops by Kleene closure
4. Select a node (other than initial or final) for removal. Replace it with a set of equivalent links whose path expressions correspond to the in and out links
5. Repeat steps 1-4 until the graph consists of a single link between the entry and exit nodes.

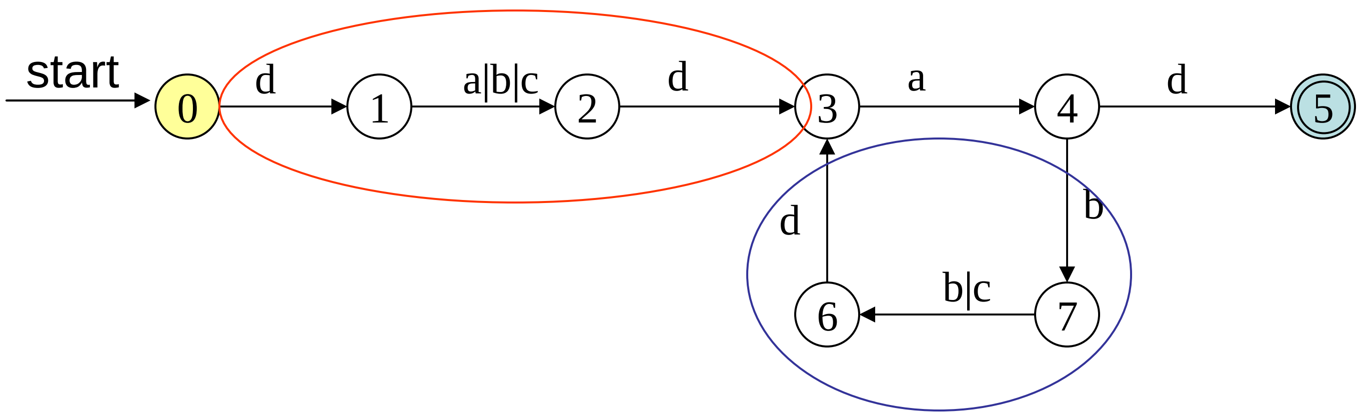
Example



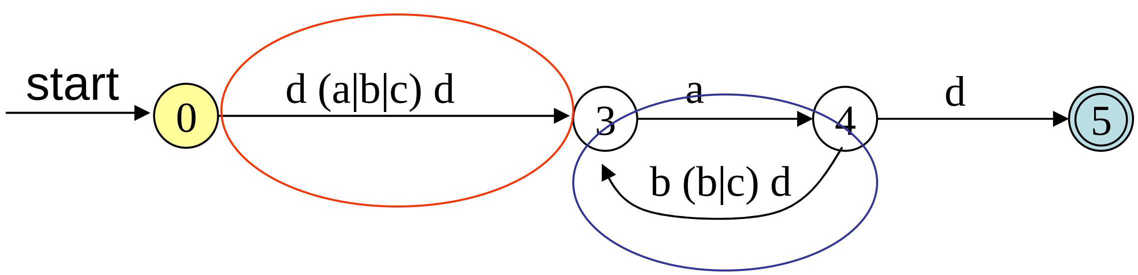
parallel edges become alternation



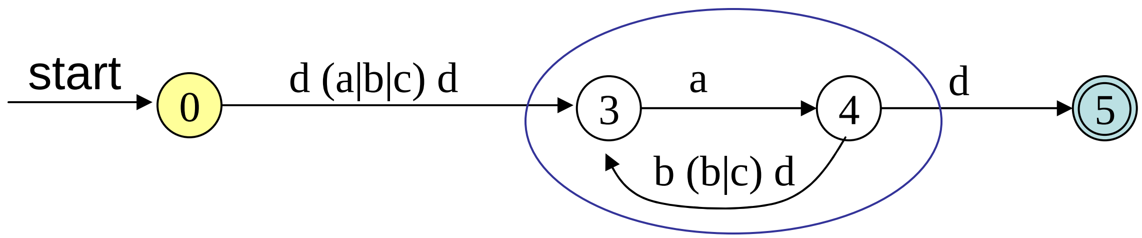
Example



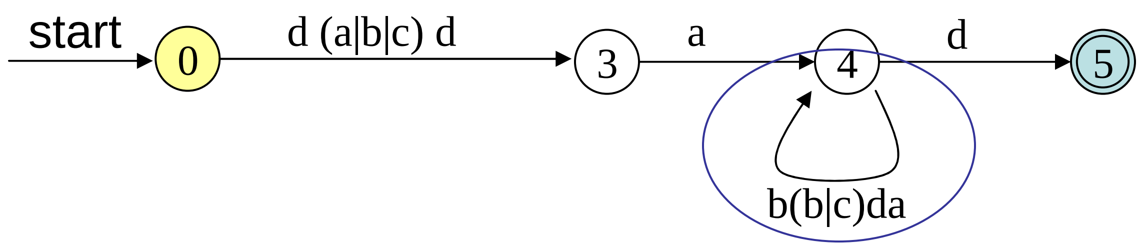
serial edges become concatenation



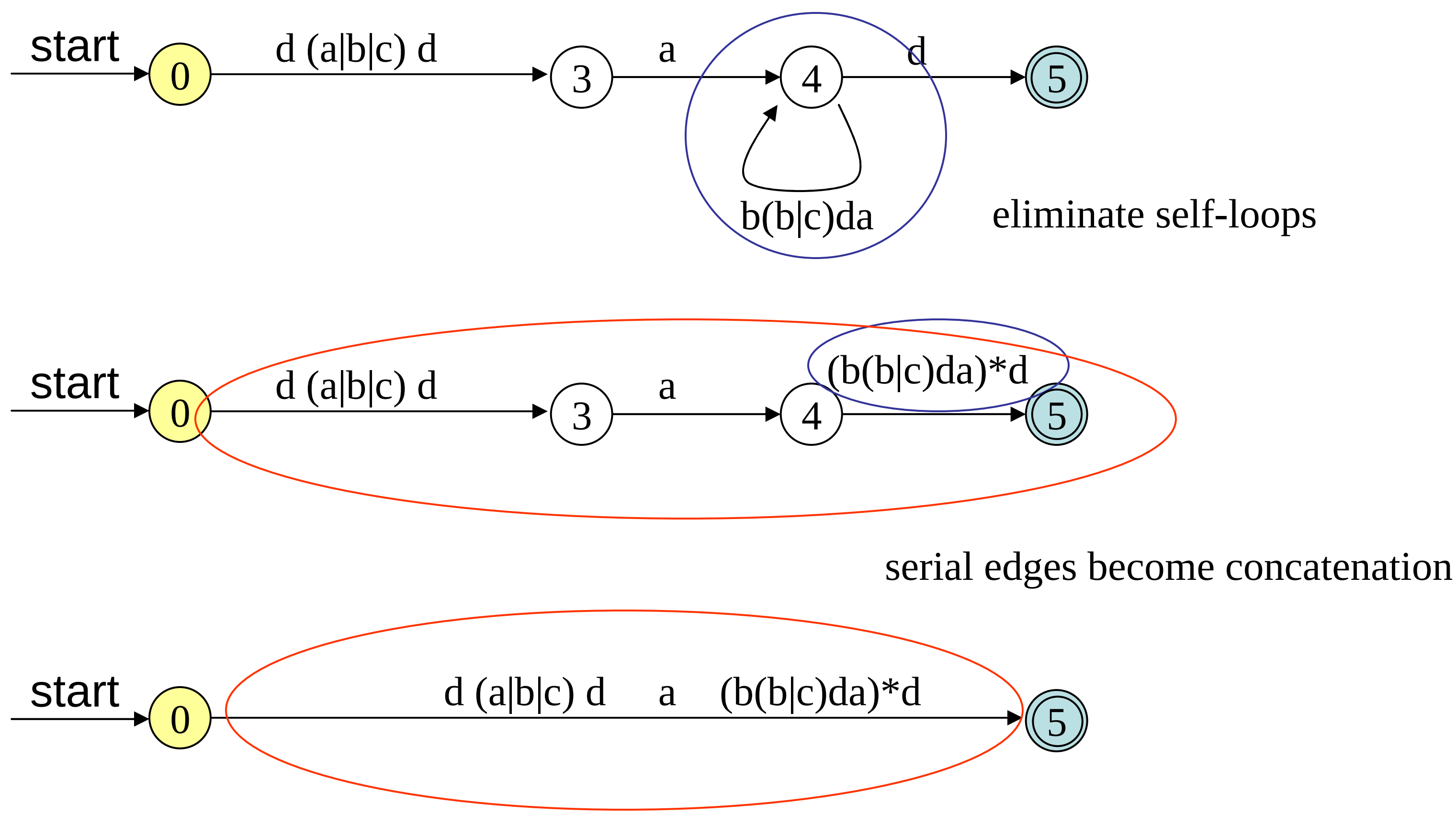
Example



Find paths that can be "shortened"



Example


start
0
d (a|b|c) d
3
a
4
d
5
b(b|c)da
eliminate self-loops
start
0
d (a|b|c) d
3
a
4
(b(b|c)da)*d
5
serial edges become concatenation
start
0
d (a|b|c) d a (b(b|c)da)*d
5

Describing Regular Languages

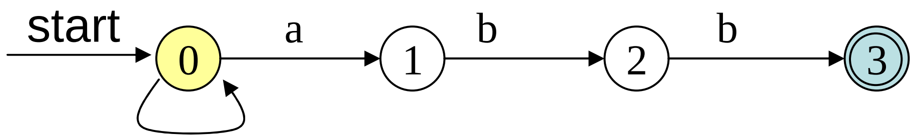
- Generate ***all*** strings in the language
- Generate ***only*** strings in the language

Try the following:

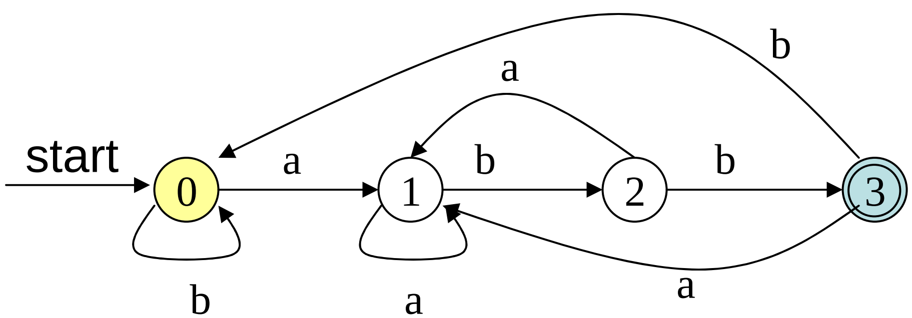
- Strings of {*a*,*b*} that end with ‘*abb*’
- Strings of {*a*,*b*} where every *a* is followed by at least one *b*

Strings of (a|b)* that end in abb

re: (a|b)*abb



NFA



DFA

Relationship among RE, NFA, DFA

- The set of strings recognized by an NFA can be described by a Regular Expression.
- The set of strings described by a Regular Expression can be recognized by an NFA.
- The set of strings recognized by an DFA can be described by a Regular Expression.
- The set of strings described by a Regular Expression can be recognized by an DFA.
- DFAs, NFAs, and Regular Expressions all have the same “power”. They describe “Regular Sets” (“Regular Languages”)
- The DFA may have a lot more states than the NFA. (May have exponentially as many states, but...)

Suggestions for writing NFA/DFA/RE

- Typically, one of these formalisms is more natural for the problem. Start with that and convert if necessary.
- In DFAs, each state typically captures some partial solution
- Be sure that you include all relevant edges (ask – does every state have an outgoing transition for all alphabet symbols?)

Non-Regular Languages

Not all languages are regular”

- The language *ww* where *w*=(a|b)*

Non-regular languages cannot be described using REs, NFAs and DFAs.