

Operations on sets: Implementation on DFAs

Member (x, X)	:	returns true if $x \in X$, false otherwise.
Complement (X)	:	returns $U \setminus X$.
Intersection (X, Y)	:	returns $X \cap Y$.
Union (X, Y)	:	returns $X \cup Y$.
Empty (X)	:	returns true if $X = \emptyset$, false otherwise.
Universal (X)	:	returns true if $X = U$, false otherwise.
Included (X, Y)	:	returns true if $X \subseteq Y$, false otherwise.
Equal (X, Y)	:	returns true if $X = Y$, false otherwise.

Assumption: each object encoded by one word, and viceversa.

Membership: trivial, linear in length of word.

Complement: trivial, swap final and non-final states.
Linear (or even constant) time.

Intersection(X, Y) : returns $X \cap Y$.

Union(X, Y) : returns $X \cup Y$.

SetDifference(X, Y) : returns $X - Y, X \setminus Y$

Symmetric set difference: returns $(X \setminus Y) \cup (Y \setminus X)$

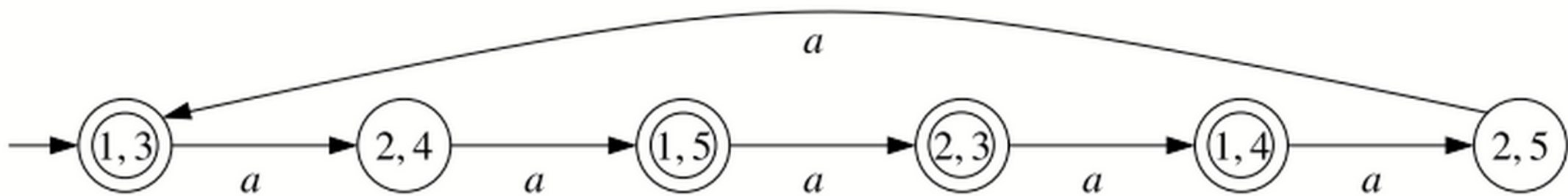
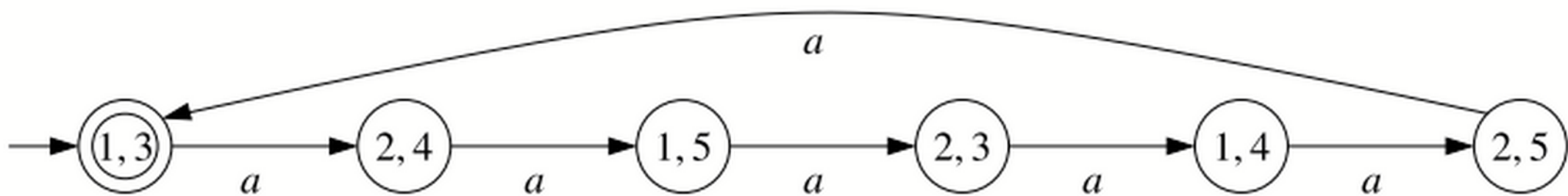
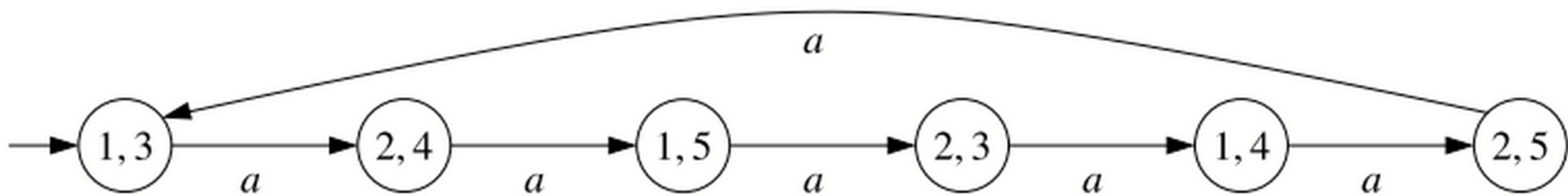
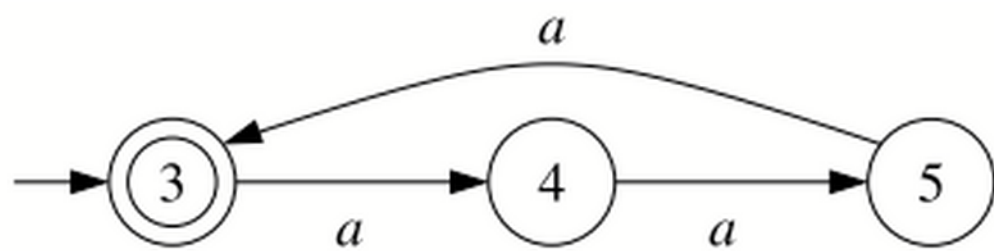
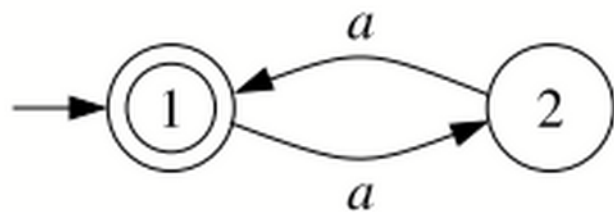
Op(X, Y, Z): returns $(X \cup Y) \setminus Z$

Pairing

Definition 4.1 Let $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be DFAs. The pairing $[A_1, A_2]$ of A_1 and A_2 is the tuple (Q, Σ, δ, q_0) where:

- $Q = \{ [q_1, q_2] \mid q_1 \in Q_1, q_2 \in Q_2 \};$
- $\delta = \{ ([q_1, q_2], a, [q'_1, q'_2]) \mid (q_1, a, q'_1) \in \delta_1, (q_2, a, q'_2) \in \delta_2 \};$
- $q_0 = [q_{01}, q_{02}].$

The run of $[A_1, A_2]$ on a word of Σ^* is defined as for DFAs.



Another example: even number of a's and even number of b's (and even number of c's ...)

Always remember:

an automaton for a regular language described as

``set of words satisfying some boolean combination of properties''

can be obtained by computing automata for the boolean properties, and then applying the composition operators.

A generic algorithm

$$L_1 \widehat{\odot} L_2 = \{w \in \Sigma^* \mid (w \in L_1) \odot (w \in L_2)\}$$

Language operation	$b_1 \odot b_2$
Union	$b_1 \vee b_2$
Intersection	$b_1 \wedge b_2$
Set difference ($L_1 \setminus L_2$)	$b_1 \wedge \neg b_2$
Symmetric difference ($L_1 \setminus L_2 \cup L_2 \setminus L_1$)	$b_1 \Leftrightarrow \neg b_2$

BinOp $[\odot](A_1, A_2)$

Input: DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: DFA $A = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}(A) = \mathcal{L}(A_1) \widehat{\odot} \mathcal{L}(A_2)$

```

1   $Q \leftarrow \emptyset; F \leftarrow \emptyset$ 
2   $q_0 \leftarrow [q_{01}, q_{02}]$ 
3   $W \leftarrow \{q_0\}$ 
4  while  $W \neq \emptyset$  do
5      pick  $[q_1, q_2]$  from  $W$ 
6      add  $[q_1, q_2]$  to  $Q$ 
7      if  $(q_1 \in F_1) \odot (q_2 \in F_2)$  then add  $[q_1, q_2]$  to  $F$ 
8      for all  $a \in \Sigma$  do
9           $q'_1 \leftarrow \delta_1(q_1, a); q'_2 \leftarrow \delta_2(q_2, a)$ 
10         if  $[q'_1, q'_2] \notin Q$  then add  $[q'_1, q'_2]$  to  $W$ 
11         add  $([q_1, q_2], a, [q'_1, q'_2])$  to  $\delta$ 
12 return  $(Q, \Sigma, \delta, q_0, F)$ 
```

Complexity: the pairing of DFAs with n_1 and n_2 states has $O(n_1 n_2)$ states.

Hence: for DFAs of size k_1, k_2 , union, intersection, etc. can be carried out in time $O(n_1 n_2)$

Language tests

Emptiness: no final states

Universality: only final states

Inclusion: $L1$ included in $L2$ iff $L1 \setminus L2$ is empty

Equality: $L1 = L2$ iff $(L1 \setminus L2) \cup (L2 \setminus L1)$ is empty

InclDFA(A_1, A_2)

Input: DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

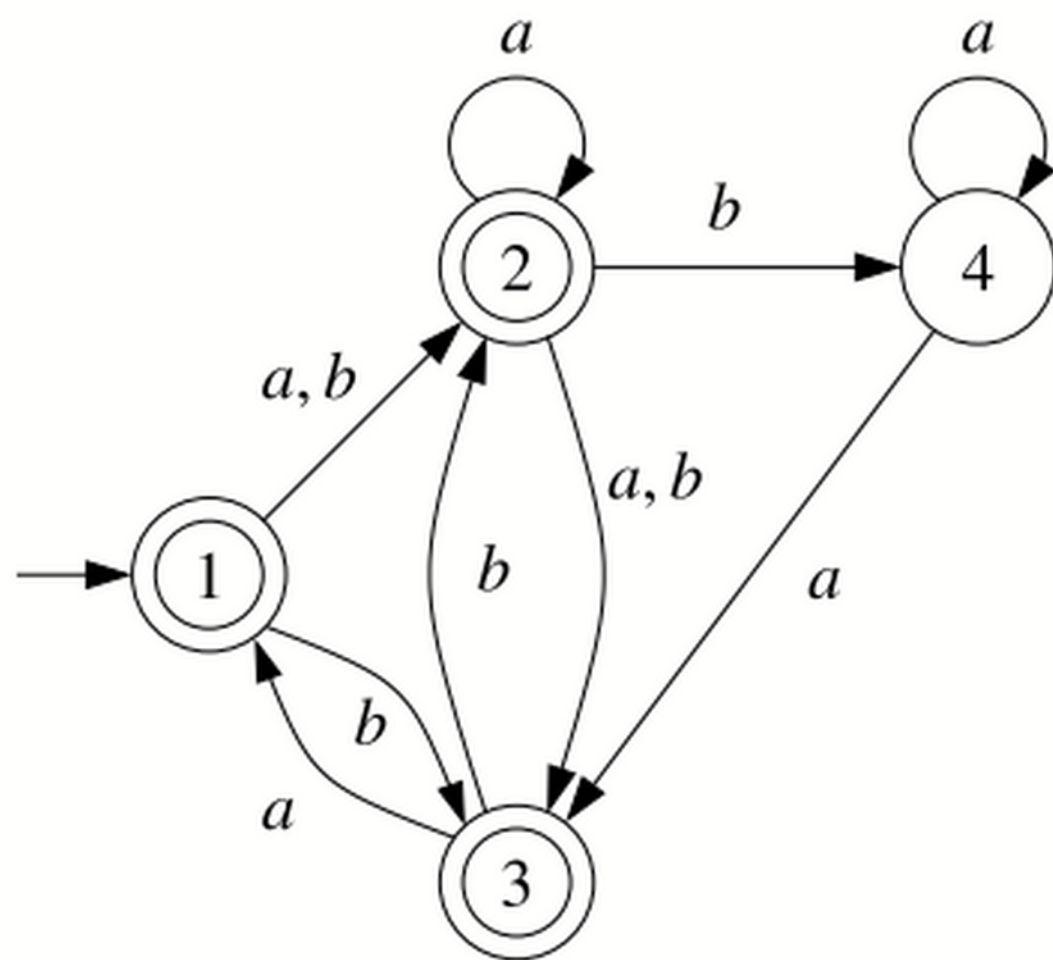
Output: **true** if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, **false** otherwise

```
1   $Q \leftarrow \emptyset$ ;  
2   $W \leftarrow \{[q_{01}, q_{02}]\}$   
3  while  $W \neq \emptyset$  do  
4      pick  $[q_1, q_2]$  from  $W$   
5      add  $[q_1, q_2]$  to  $Q$   
6      if  $(q_1 \in F_1)$  and  $(q_2 \notin F_2)$  then return false  
7      for all  $a \in \Sigma$  do  
8           $q'_1 \leftarrow \delta_1(q_1, a)$ ;  $q'_2 \leftarrow \delta_2(q_2, a)$   
9          if  $[q'_1, q'_2] \notin Q$  then add  $[q'_1, q'_2]$  to  $W$   
10 return true
```

Operations on sets: Implementation on NFAs

Member (x, X)	:	returns true if $x \in X$, false otherwise.
Complement (X)	:	returns $U \setminus X$.
Intersection (X, Y)	:	returns $X \cap Y$.
Union (X, Y)	:	returns $X \cup Y$.
Empty (X)	:	returns true if $X = \emptyset$, false otherwise.
Universal (X)	:	returns true if $X = U$, false otherwise.
Included (X, Y)	:	returns true if $X \subseteq Y$, false otherwise.
Equal (X, Y)	:	returns true if $X = Y$, false otherwise.

Membership



Prefix read	W
ϵ	$\{q_0\}$
a	$\{q_2\}$
aa	$\{q_2, q_3\}$
aaa	$\{q_1, q_2, q_3\}$
$aaab$	$\{q_2, q_3\}$
$aaabb$	$\{q_2, q_3, q_4\}$
$aaabba$	$\{q_1, q_2, q_3, q_4\}$

$Mem[A](w)$

Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$, word $w \in \Sigma^*$,

Output: **true** if $w \in \mathcal{L}(A)$, **false** otherwise

```

1   $W \leftarrow \{q_0\};$ 
2  while  $w \neq \varepsilon$  do
3       $U \leftarrow \emptyset$ 
4      for all  $q \in W$  do
5          add  $\delta(q, head(w))$  to  $U$ 
6       $W \leftarrow U$ 
7       $w \leftarrow tail(w)$ 
8  return  $(W \cap F \neq \emptyset)$ 
```

Complexity:

while loop executed $|w|$ times
 for loop executed at most $|Q|$ times
 each execution takes $O(|Q|)$ time

Overall: $O(|w||Q|^2)$ time

Complement

Swapping final and non-final states doesn't work:

Solution: determinize and then swap states.

Problem: Exponential blow-up in size!!

To be avoided whenever possible!!

No better way: there are NFAs with $O(n)$ states such that the smallest NFA for the complement has $O(2^n)$ states

(see the next exercise sheet)

Union and intersection

The pairing construction still works for union and intersection, with the same complexity, but not for set difference, or other non-monotonic operations.

There is a better construction for union, but not for intersection.

IntersNFA(A_1, A_2)

Input: NFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

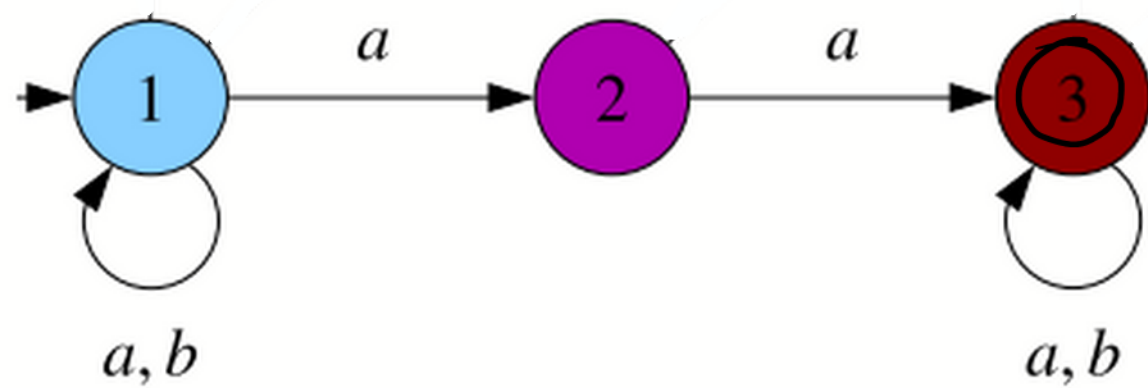
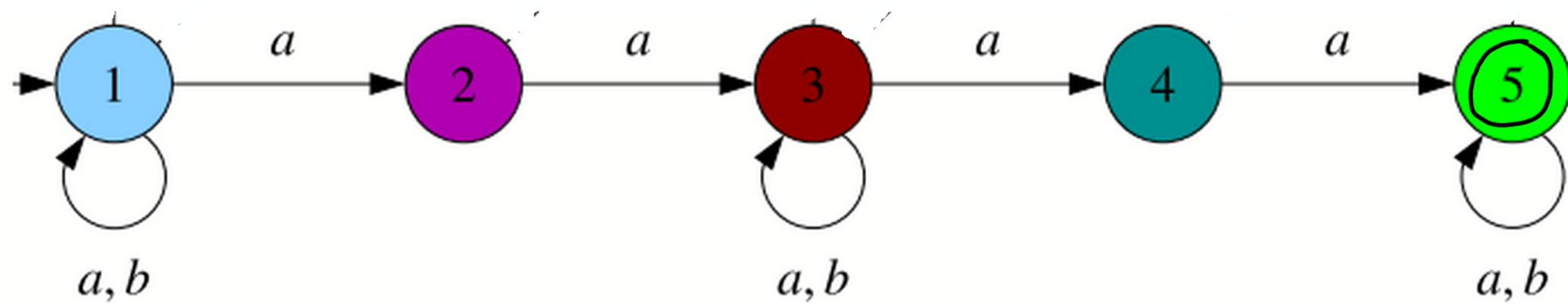
Output: NFA $A_1 \cap A_2 = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}(A_1 \cap A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$

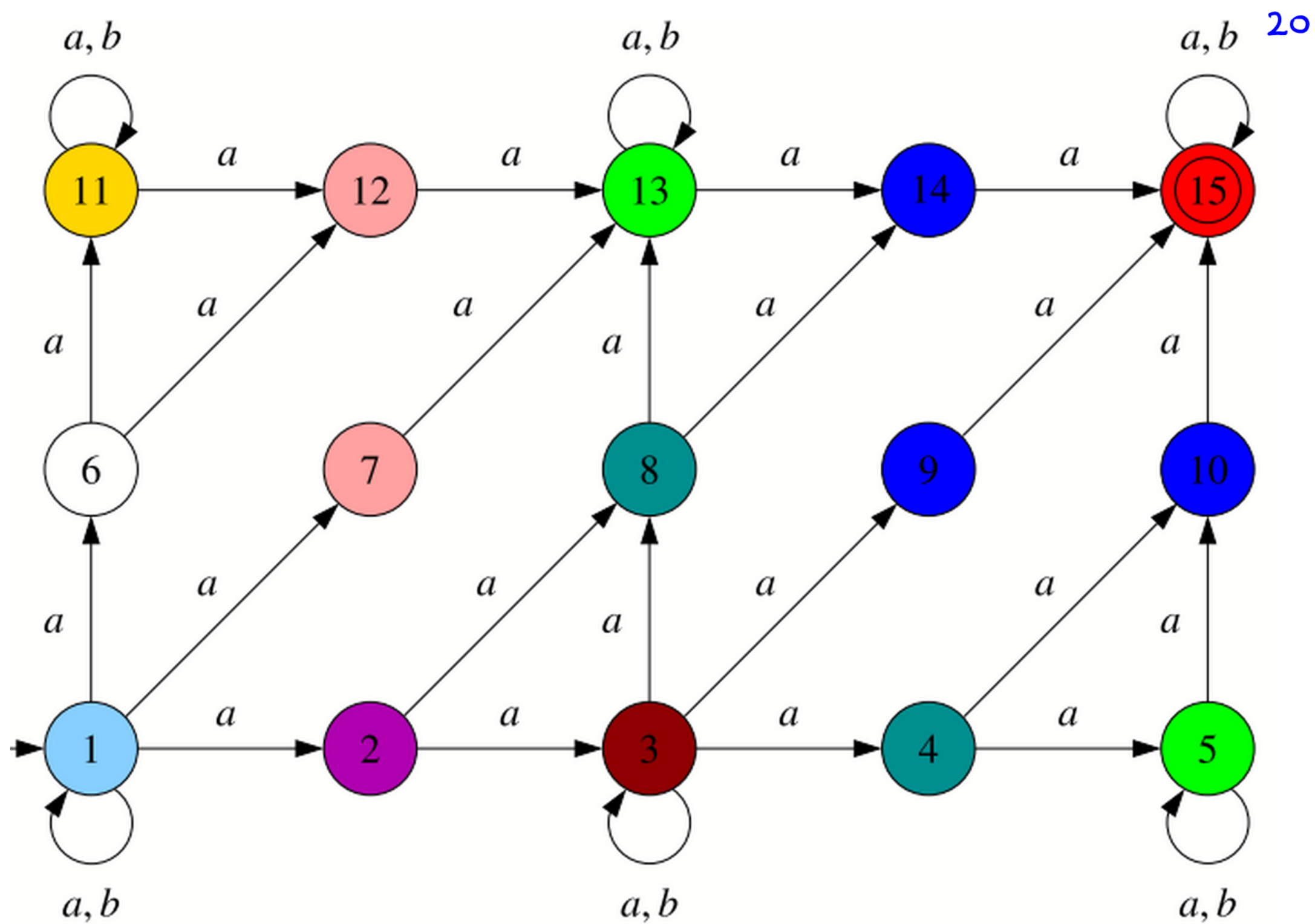
```

1   $Q \leftarrow \emptyset; F \leftarrow \emptyset$ 
2   $q_0 \leftarrow [q_{01}, q_{02}]$ 
3   $W \leftarrow \{ [q_{01}, q_{02}] \}$ 
4  while  $W \neq \emptyset$  do
5      pick  $[q_1, q_2]$  from  $W$ 
6      add  $[q_1, q_2]$  to  $Q$ 
7      if  $q_1 \in F_1$  and  $q_2 \in F_2$  then add  $[q_1, q_2]$  to  $F$ 
8      for all  $a \in \Sigma$  do
9          for all  $q'_1 \in \delta_1(q_1, a), q'_2 \in \delta_2(q_2, a)$  do
10             if  $[q'_1, q'_2] \notin Q$  then add  $[q'_1, q'_2]$  to  $W$ 
11             add  $([q_1, q_2], a, [q'_1, q'_2])$  to  $\delta$ 
12  return  $(Q, \Sigma, \delta, q_0, F)$ 

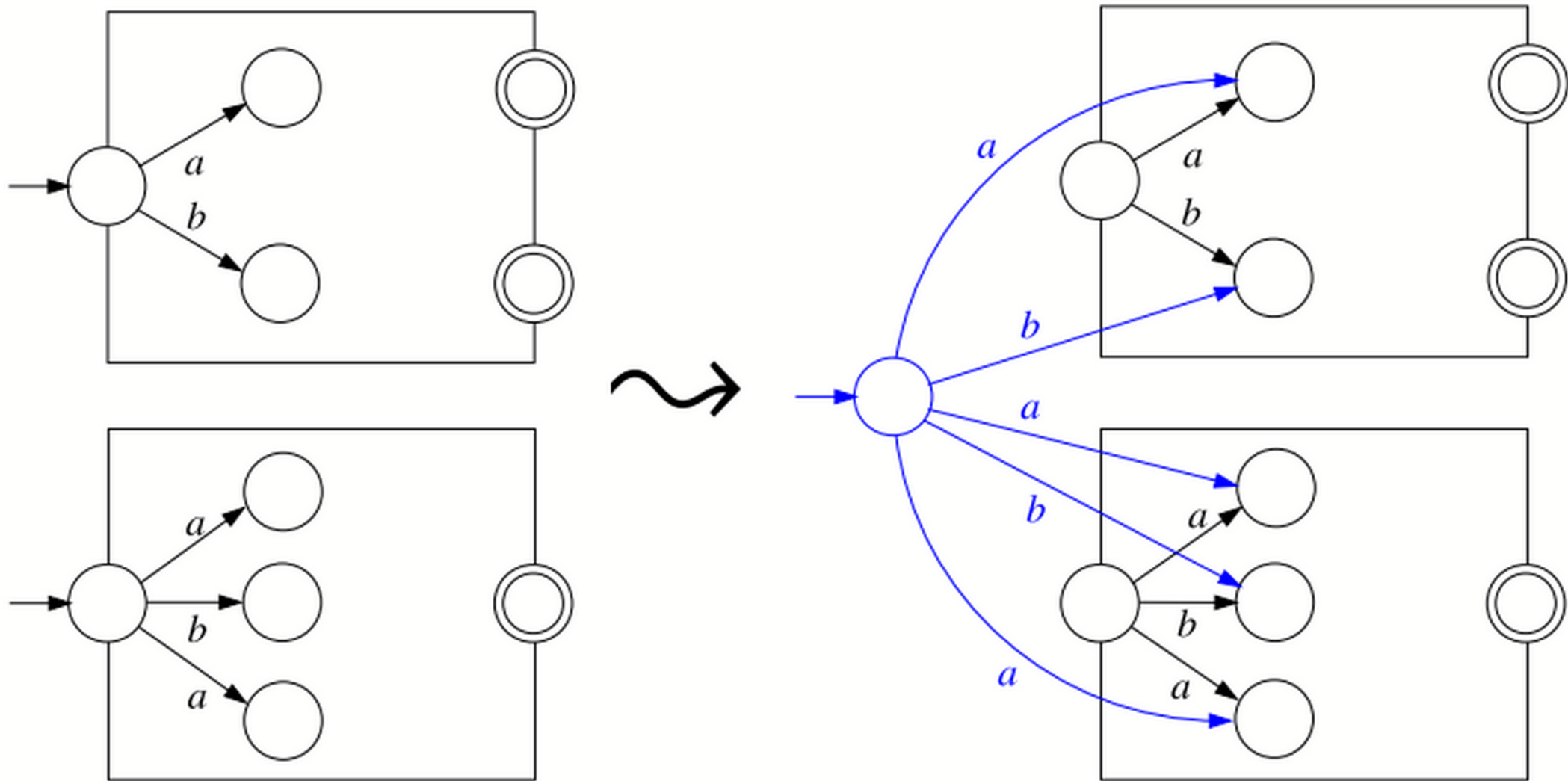
```

For the complexity, observe that in the worst case the algorithm must examine all pairs $[t_1, t_2]$ of transitions of $\delta_1 \times \delta_2$, but every pair is examined at most once. So the runtime is $\mathcal{O}(|\delta_1||\delta_2|)$.





Union



UnionNFA(A_1, A_2)

Input: NFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: NFA $A_1 \cup A_2$ with $\mathcal{L}(A_1 \cup A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$

```
1   $Q \leftarrow Q_1 \cup Q_2 \cup \{q_0\}; F \leftarrow F_1 \cup F_2$   
2   $\delta \leftarrow \delta_1 \cup \delta_2$   
3  for all  $(q_{01}, a, q) \in \delta_1$  do  
4      add  $(q_0, a, q_1)$  to  $\delta$   
5  for all  $(q_{02}, a, q) \in \delta_2$  do  
6      add  $(q_0, a, q_2)$  to  $\delta$   
7  return  $(Q, \Sigma, \delta, q_0, F)$ 
```

$\mathcal{O}(m_1 + m_2)$, where m_i is the number of transitions of A_i starting at q_{0i} .

Example showing that the pairing construction does not work for set difference:

- $\text{SetDiff}(A, A)$ should always produce an NFA recognizing the empty language, but this is not the case.

Emptiness and universality

Exactly one of these two sentences is true:

NFA is empty iff every state is non-final

NFA is universal iff every state is final

Emptiness and universality

Exactly one of these two sentences is true:

NFA is empty iff every state is non-final

NFA is universal iff every state is final

Emptiness decidable in linear time.

Universality is PSPACE-complete.

Theorem 4.6 *The universality problem for NFAs is PSPACE-complete*

Proof: We only sketch the proof. To prove that the problem is in PSPACE, we show that it belongs to NPSPACE and apply Savitch's theorem. The polynomial-space nondeterministic algorithm for universality looks as follows. Given an NFA $A = (Q, \Sigma, \delta, q_0, F)$, the algorithm guesses a run of $B = \text{NFAtoDFA}(A)$ leading from $\{q_0\}$ to a non-final state, i.e., to a set of states of A containing no final state (if such a run exists). The algorithm only does not store the whole run, only the current state, and so it only needs linear space in the size of A .

We prove PSPACE-hardness by reduction from the acceptance problem for linearly bounded automata. A linearly bounded automaton is a deterministic Turing machine that always halts and only uses the part of the tape containing the input. A configuration of the Turing machine on an input of length k is coded as a word of length k . The run of the machine on an input can be encoded as a word $c_0\#c_1\ldots\#c_n$, where the c_i 's are the encodings of the configurations.

Let Σ be the alphabet used to encode the run of the machine. Given an input x , M accepts if there exists a word w of Σ^* satisfying the following properties:

- (a) w has the form $c_0\#c_1\ldots\#c_n$, where the c_i 's are configurations;
- (b) c_0 is the initial configuration;
- (c) c_n is an accepting configuration; and
- (d) for every $0 \leq i \leq n-1$: c_{i+1} is the successor configuration of c_i according to the transition relation of M .

The reduction shows how to construct in polynomial time, given a linearly bounded automaton M and an input x , an NFA $A(M, x)$ accepting all the words of Σ^* that do *not* satisfy at least one of the conditions (a)-(d) above. We then have

- If M accepts x , then there is a word $w(M, x)$ encoding the accepting run of M on x , and so $\mathcal{L}(A(M, x)) = \Sigma^* \setminus \{w(M, x)\}$.
- If M rejects x , then no word encodes an accepting run of M on x , and so $\mathcal{L}(A(M, x)) = \Sigma^*$.

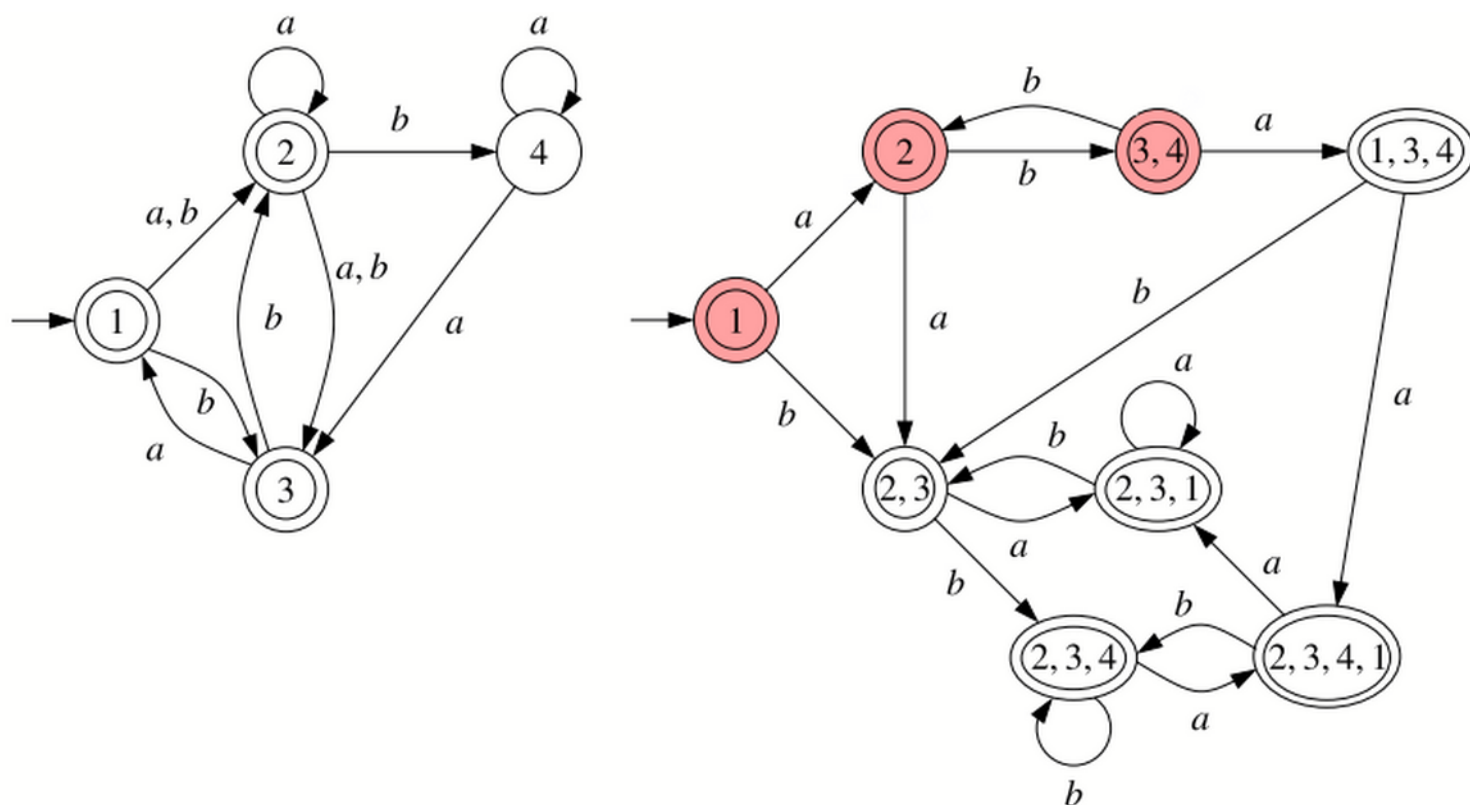
So M accepts x if and only if $\mathcal{L}(A(M, x)) \neq \Sigma^*$, and we are done. □

- Complement and check for emptiness
Exponential complexity
- Improvements:
 - (1) check for emptiness while complementing
(on the fly check)
 - (2) subsumption test

Definition 4.7 Let A be a NFA, and let $B = \text{NFAtoDFA}(A)$. A state Q' of B is minimal if no other state Q'' satisfies $Q'' \subset Q'$.

Proposition 4.8 Let A be a NFA, and let $B = \text{NFAtoDFA}(A)$. A is universal iff every minimal state of B is final.

Proof: Since A and B recognize the same language, A is universal iff B is universal. So A is universal iff every state of B is final. But a state of B is final iff it contains some final state of A , and so every state of B is final iff every minimal state of B is final. $\square$

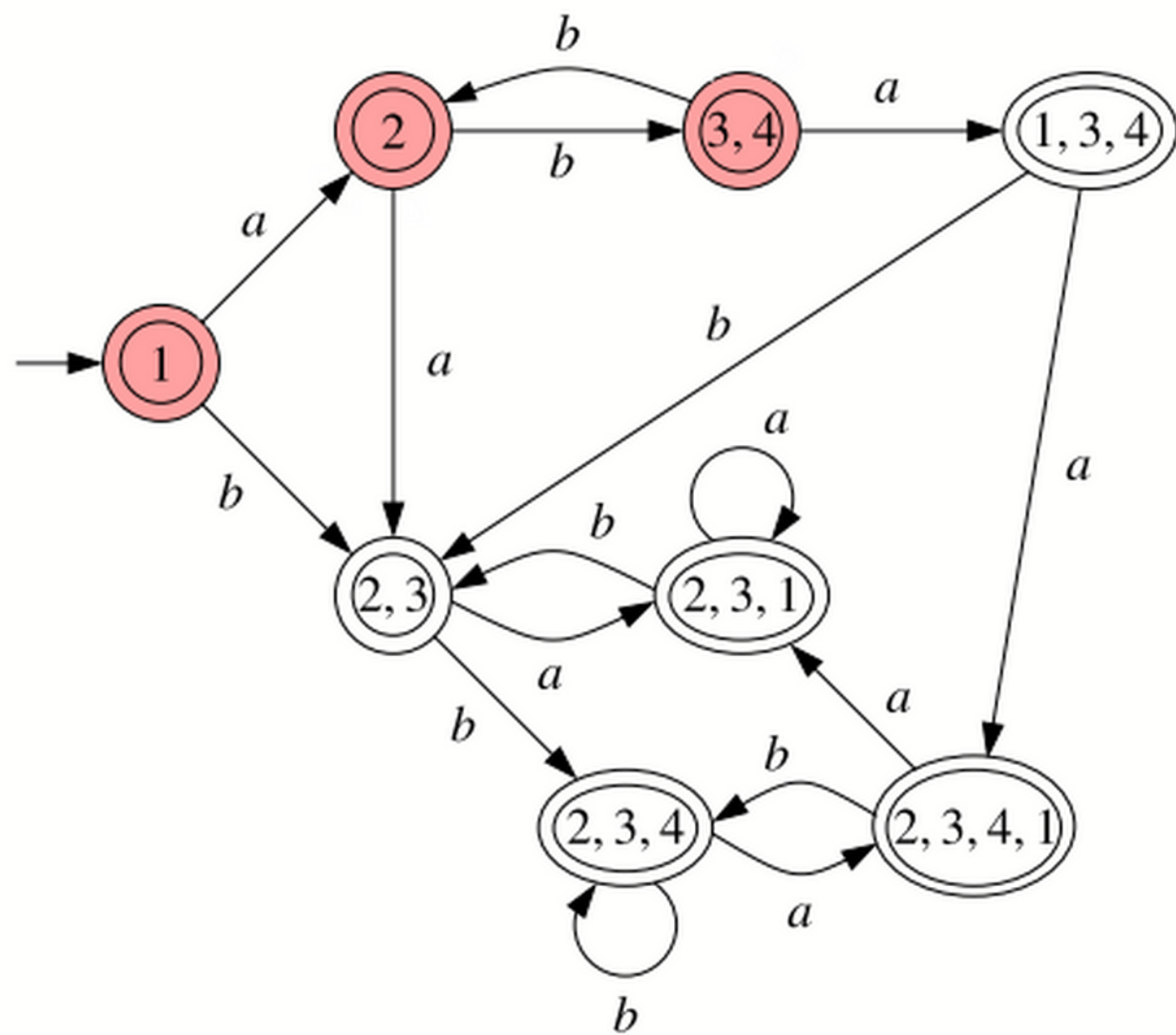
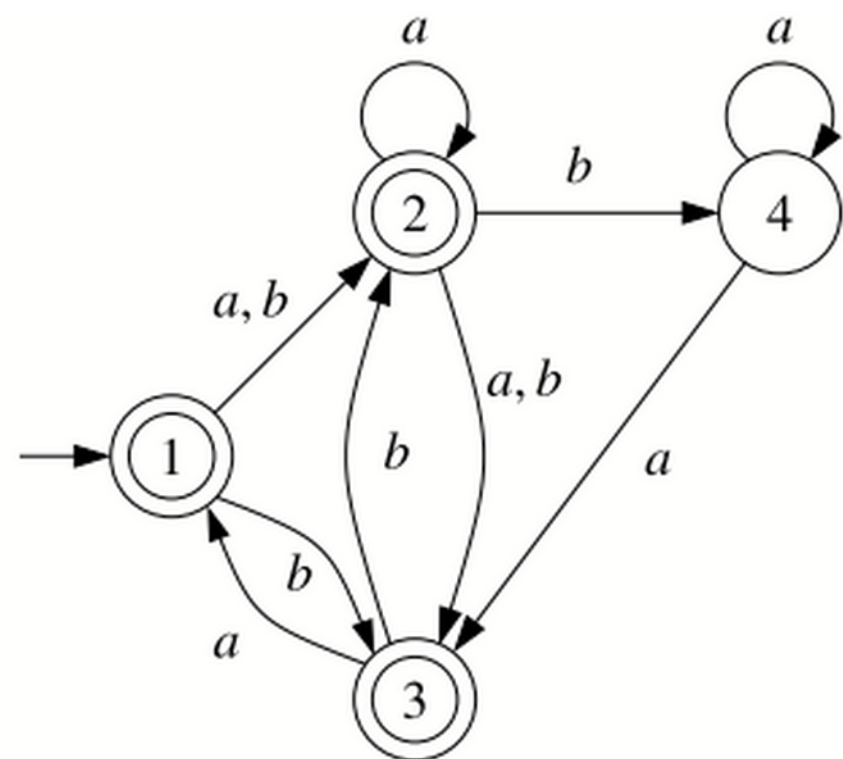


UnivNFA(A)

Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$

Output: true if $\mathcal{L}(A) = \Sigma^*$, false otherwise

```
1   $\mathcal{Q} \leftarrow \emptyset;$ 
2   $\mathcal{W} \leftarrow \{ \{q_0\} \}$ 
3  while  $\mathcal{W} \neq \emptyset$  do
4      pick  $Q'$  from  $\mathcal{W}$ 
5      if  $Q' \cap F = \emptyset$  then return false
6      add  $Q'$  to  $\mathcal{Q}$ 
7      for all  $a \in \Sigma$  do
8          if  $\mathcal{W} \cup \mathcal{Q}$  contains no  $Q'' \subseteq \delta(Q', a)$  then add  $\delta(Q', a)$  to  $\mathcal{W}$ 
9  return true
```



Is it correct?

By removing a non-minimal state we might be preventing the discovery of minimal states in the future!

Proposition 4.11 *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a NFA, and let $B = \text{NFAtoDFA}(A)$. After termination of $\text{UnivNFA}(A)$, the set $\mathcal{Q}$ contains all minimal states of B .*

Proof: Assume the contrary.

Then B has a shortest path $Q_1 \rightarrow Q_2 \rightarrow \dots \rightarrow Q_n$ such that

- Q_1 belongs to $\mathcal{Q}$ (after termination), and
- Q_n is minimal.

Since the path is shortest, Q_2 does not belong to $\mathcal{Q}$

When UnivNFA processes Q_1 it does not add Q_2 . Why?

Because UnivNFA computes a state Q_2' strictly contained in Q_2 .

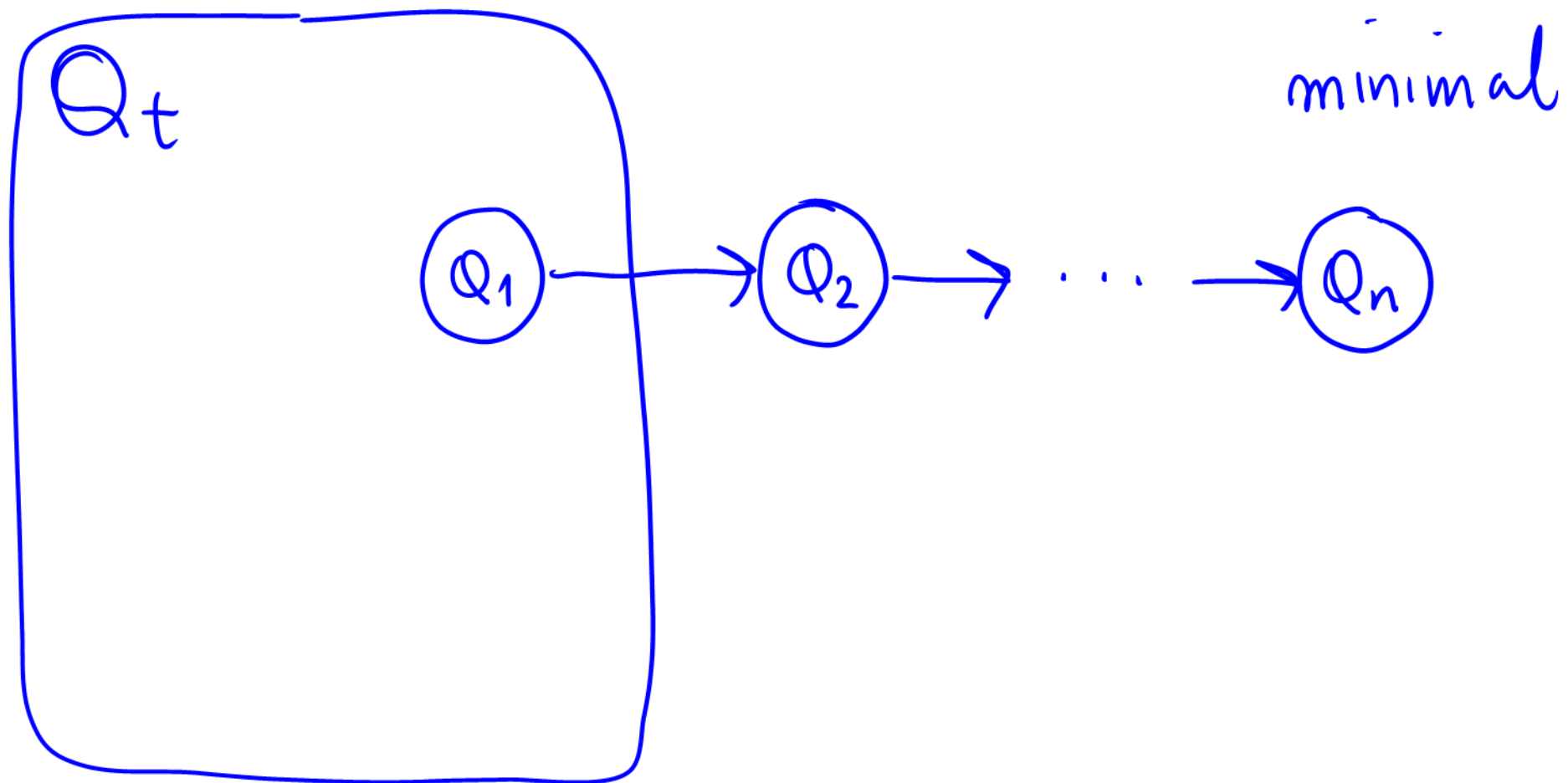
But then B has a path $Q_2' \rightarrow \dots \rightarrow Q_{n'}$ with $Q_{n'}$ contained in Q_n .

Since Q_n is minimal, $Q_{n'}$ is minimal (actually $Q_n = Q_{n'}$)

So the path $Q_2' \rightarrow \dots \rightarrow Q_{n'}$ satisfies

- Q_2' belongs to $\mathcal{Q}$ (after termination), and
- $Q_{n'}$ is minimal.

contradicting that the original path is shortest.



Inclusion and equality

Proposition 4.14 *The inclusion problem for NFAs is PSPACE-complete*

Proof: We first prove membership in PSPACE. Since $\text{PSPACE} = \text{co-PSPACE} = \text{NPSPACE}$, it suffices to give a polynomial space nondeterministic algorithm that decides non-inclusion. Given NFAs A_1 and A_2 , the algorithm guesses a word $w \in \mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$ letter by letter, maintaining the sets Q'_1, Q'_2 of states that A_1 and A_2 can reach by the word guessed so far. When the guessing ends, the algorithm checks that Q'_1 contains some final state of A_1 , but Q'_2 does not.

Hardness follows from the fact that A is universal iff $\Sigma \subseteq \mathcal{L}(A)$, and so the universality problem, which is PSPACE-complete, is a subproblem of the inclusion problem. $\square$

Algorithm: use $L_1 \subseteq L_2$ iff $L_1 \cap \overline{L_2} = \emptyset$.

Concatenate four algorithms:

- (1) determinize A2
- (2) complement the result
- (3) intersect it with A1, and
- (4) check for emptiness

State of (3): pair (q, Q) , where q in Q_1 and Q subset of Q_2

Easy optimizations:

- we only need the states of (3), not its transitions
- do not perform (1), then (2), then (3): construct directly the states of (3)
- check (4) while constructing (3)

Further optimization: subsumption test

Definition 4.12 Let A_1, A_2 be NFAs, and let $B_2 = \text{NFAtoDFA}(A_2)$. A state $[q_1, Q_2]$ of $[A_1, B_2]$ is minimal if no other state $[q'_1, Q'_2]$ satisfies $q'_1 = q_1$ and $Q'_2 \subset Q_2$.

Proposition 4.13 $L(A_1) \subseteq L(A_2)$ iff every minimal state $[q_1, Q_2]$ of $[A_1, B_2]$ satisfying $q_1 \in F_1$ also satisfies $Q_2 \cap F_2 \neq \emptyset$.

Proof: Since A_2 and B_2 recognize the same language, $L(A_1) \subseteq L(A_2)$ iff $L(A_1) \cap \overline{L(A_2)} = \emptyset$ iff $L(A_1) \cap \overline{L(B_2)} = \emptyset$ iff $[A_1, B_2]$ has a state $[q_1, Q_2]$ such that $q_1 \in F_1$ and $Q_2 \cap F_2 = \emptyset$. But $[A_1, B_2]$ has some state satisfying this condition iff it has some minimal state satisfying it.

InclNFA(A_1, A_2)

Input: NFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: true if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, false otherwise

```

1   $Q \leftarrow \emptyset$ ;
2   $W \leftarrow \{ [q_{01}, \{q_{02}\}] \}$ 
3  while  $W \neq \emptyset$  do
4      pick  $[q_1, Q_2]$  from  $W$ 
5      if  $(q_1 \in F_1)$  and  $(Q_2 \cap F_2 = \emptyset)$  then return false
6      add  $[q_1, Q_2]$  to  $Q$ 
7      for all  $a \in \Sigma, q'_1 \in \delta_1(q_1, a)$  do
8           $Q'_2 \leftarrow \delta_2(Q_2, a)$ 
9          if  $W \cup Q$  contains no  $[q'_1, Q'_2]$  s.t.  $q'_1 = q'_1$  and  $Q'_2 \subseteq Q'_2$  then
10             add  $[q'_1, Q'_2]$  to  $W$ 
11 return true

```

Complexity:

- Let A_1, A_2 , with n_1, n_2 states over a k -letter alphabet
- Without the subsumption test:
 - The while loop is executed at most $n_1 2^{n_2}$ times
 - The for-loop is executed at most $O(n_1)$ times
 - An execution of the loop takes $O(n_2^2)$ time
 - Overall: $O(k n_1^2 n_2^2 2^{n_2})$ time
- With the subsumption test the worst-case complexity is higher. Exercise: give an upper bound.

Important special case:

- A1 is an NFA, A2 is a DFA
- Complementing A2 is now easy
- We get $O(n_1^2 n_2)$ time

Equality: check inclusion in both directions.