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\Y
Symmetric set difference: returns (X\Y) U (Y\X)

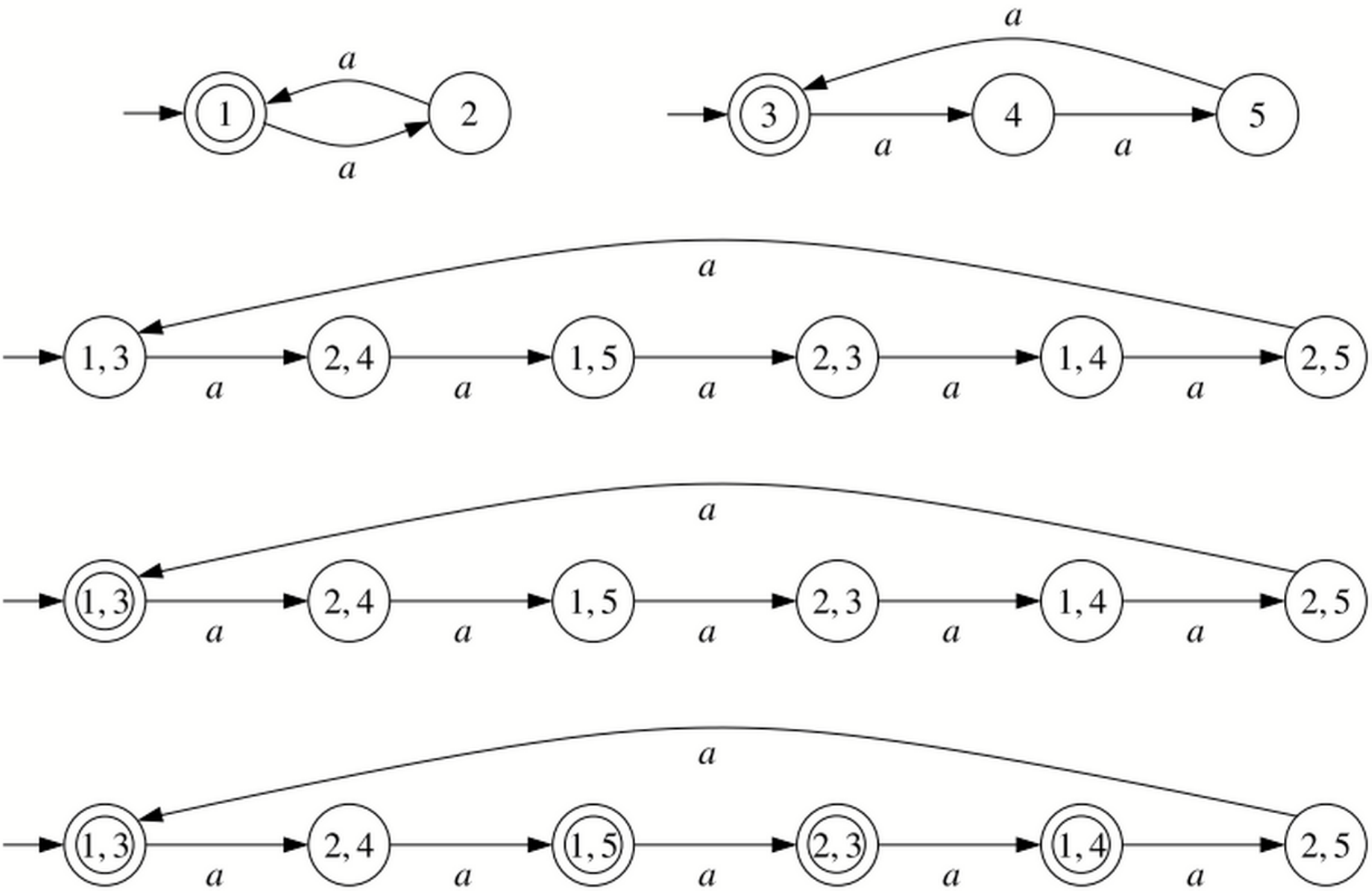
Op(X,Y,Z): returns (X U Y) \ Z

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.*


a
1
2
a
a
3
4
5
a
a
a
1, 3
2, 4
1, 5
2, 3
1, 4
2, 5
a
a
a
a
a
a
a
1, 3
2, 4
1, 5
2, 3
1, 4
2, 5
a
a
a
a
a
a
a
1, 3
2, 4
1, 5
2, 3
1, 4
2, 5
a
a
a
a
a
a

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

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

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 $\mathcal{L}(A) = \mathcal{L}(A_1) \mathbin{\widehat{\odot}} \mathcal{L}(A_2)$

```
1   Q ← ∅; F ← ∅
2   q0 ← [q01, q02]
3   W ← {q0}
4   while W ≠ ∅ do
5       pick [q1, q2] from W
6       add [q1, q2] to Q
7       if (q1 ∈ F1) ⊙ (q2 ∈ F2) then add [q1, q2] to F
8       for all a ∈ Σ do
9           q1' ← δ1(q1, a); q2' ← δ2(q2, a)
10          if [q1', q2'] ∉ Q then add [q1', q2'] to W
11          add ([q1, q2], a, [q1', q2']) to δ
12  return (Q, Σ, δ, q0, F)
```

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

- Hence: for DFAs with n_1, n_2 states and an alphabet with k letters, union, intersection, etc. can be carried out in $O(k \cdot n_1 \cdot n_2)$ time.

Language Tests

- Emptiness: a DFA is empty iff
  it has no final states
- Universality: a DFA is universal iff
  it has only final states
- Inclusion: $L_1 \subseteq L_2$ iff $L_1 \setminus L_2 = \emptyset$

- Equality: $L_1 = L_2$ iff $(L_1 \setminus L_2) \cup (L_2 \setminus L_1) = \emptyset$

InclDFA(A_1, A_2)

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

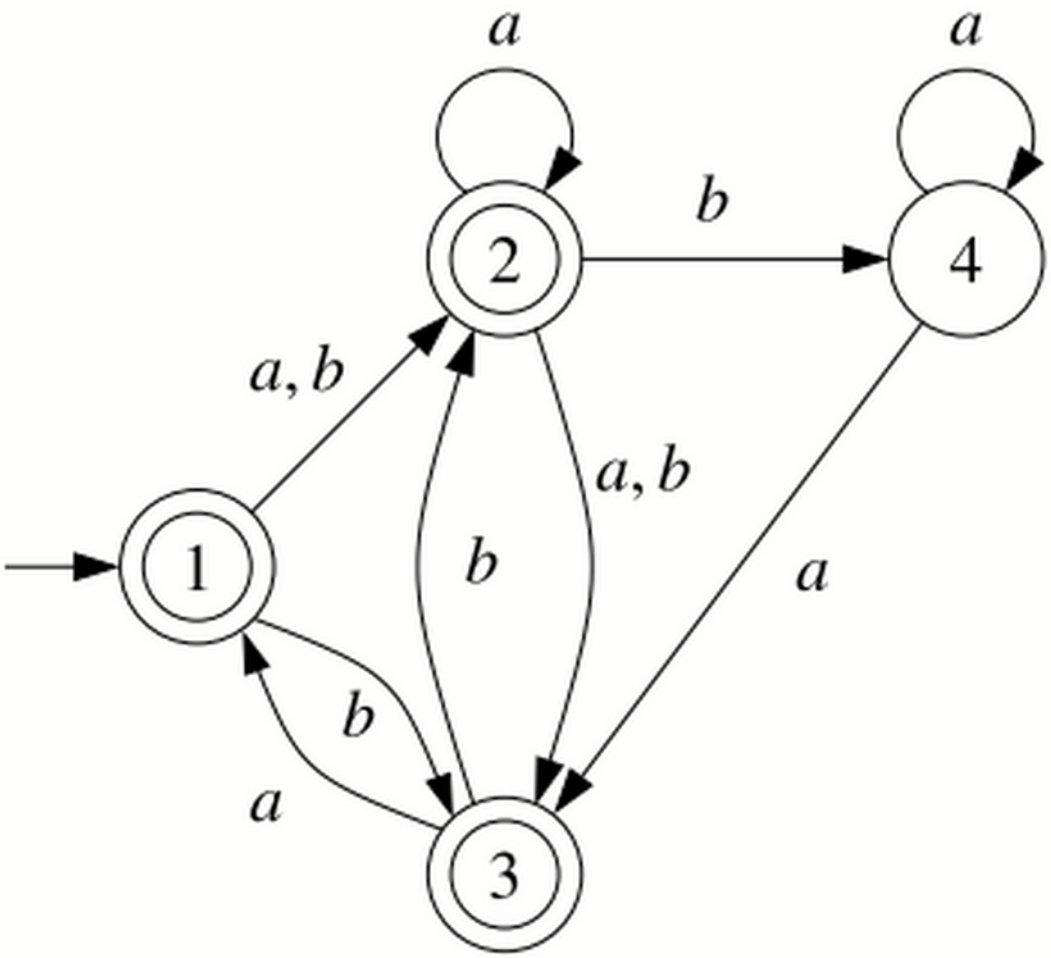
Output: **true** if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, **false** otherwise

```
1   Q ← ∅;
2   W ← {[q01, q02]}
3   while W ≠ ∅ do
4       pick [q1, q2] from W
5       add [q1, q2] to Q
6       if (q1 ∈ F1) and (q2 ∉ F2) then return false
7       for all a ∈ Σ do
8           q1' ← δ1(q1, a); q2' ← δ2(q2, a)
9           if [q1', q2'] ∉ Q then add [q1', q2'] 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 ← {q0};
2  while w ≠ ε do
3     U ← ∅
4     for all q ∈ W do
5        add δ(q, head(w)) to U
6     W ← U
7     w ← tail(w)
8  return (W ∩ F ≠ ∅)
```

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 does not 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 n states such that the smallest NFA for their complement has $\Theta(2^n)$ states.

Union and intersection

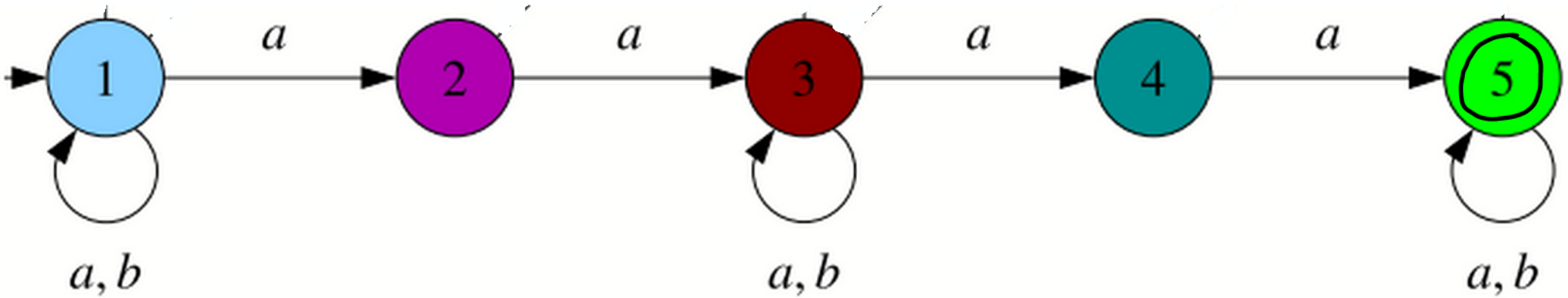
The pairing construction still works for union and intersection, with the same complexity, but not for set diference, or other non-monotonic operations.

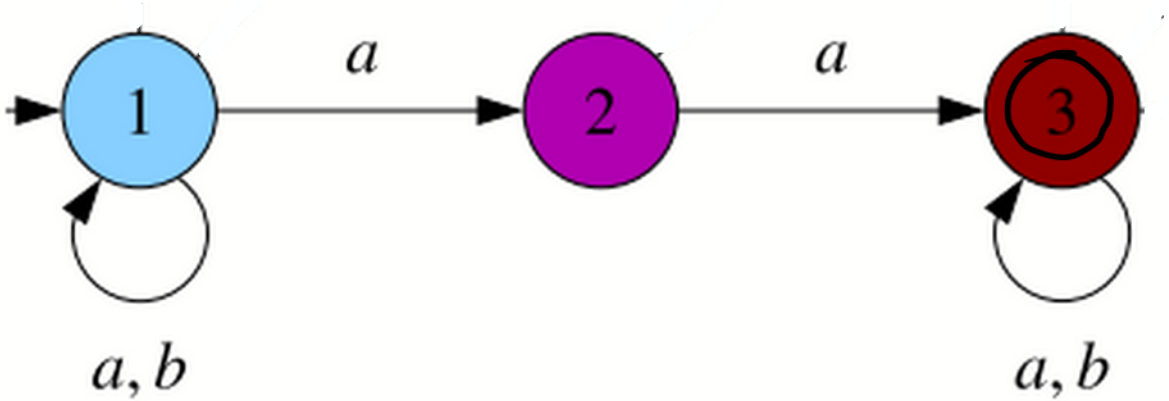
There is a better construction for union, but not for intersection.

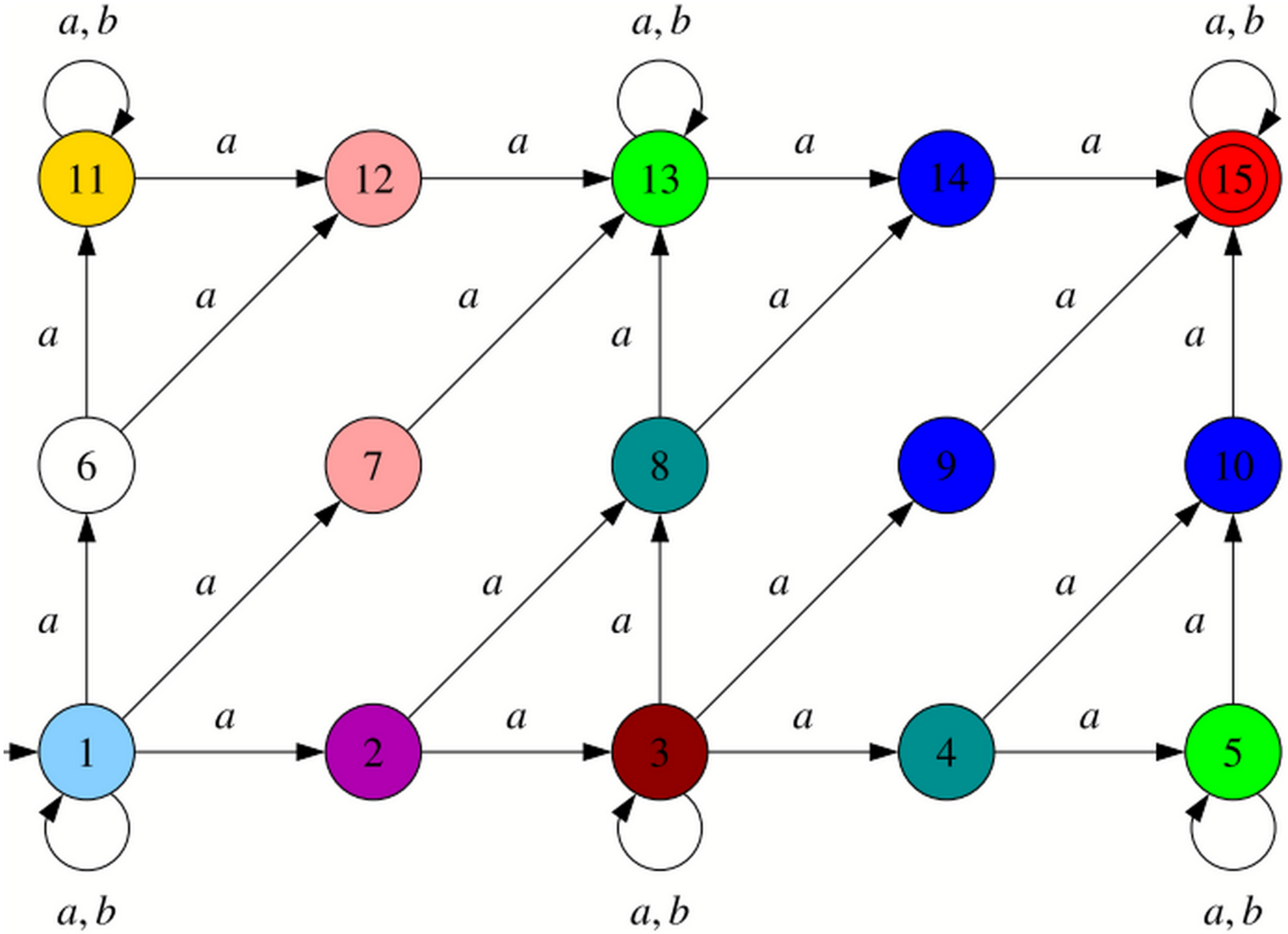
IntersNFA(A_1, A_2)
Input: NFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
Output: NFA $A_1 \cap A_2 = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}(A_1 \cap A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$

```
1   Q ← ∅; F ← ∅
2   q0 ← [q01, q02]
3   W ← { [q01, q02] }
4   while W ≠ ∅ do
5      pick [q1, q2] from W
6      add [q1, q2] to Q
7      if q1 ∈ F1 and q2 ∈ F2) then add [q1, q2] to F
8      for all a ∈ Σ do
9         for all q'1 ∈ δ1(q1, a), q'2 ∈ δ2(q2, a) do
10        if [q'1, q'2] ∉ Q then add [q'1, q'2] to W
11        add ([q1, q2], a, [q'1, q'2]) to δ
12  return (Q, Σ, δ, q0, F)
```

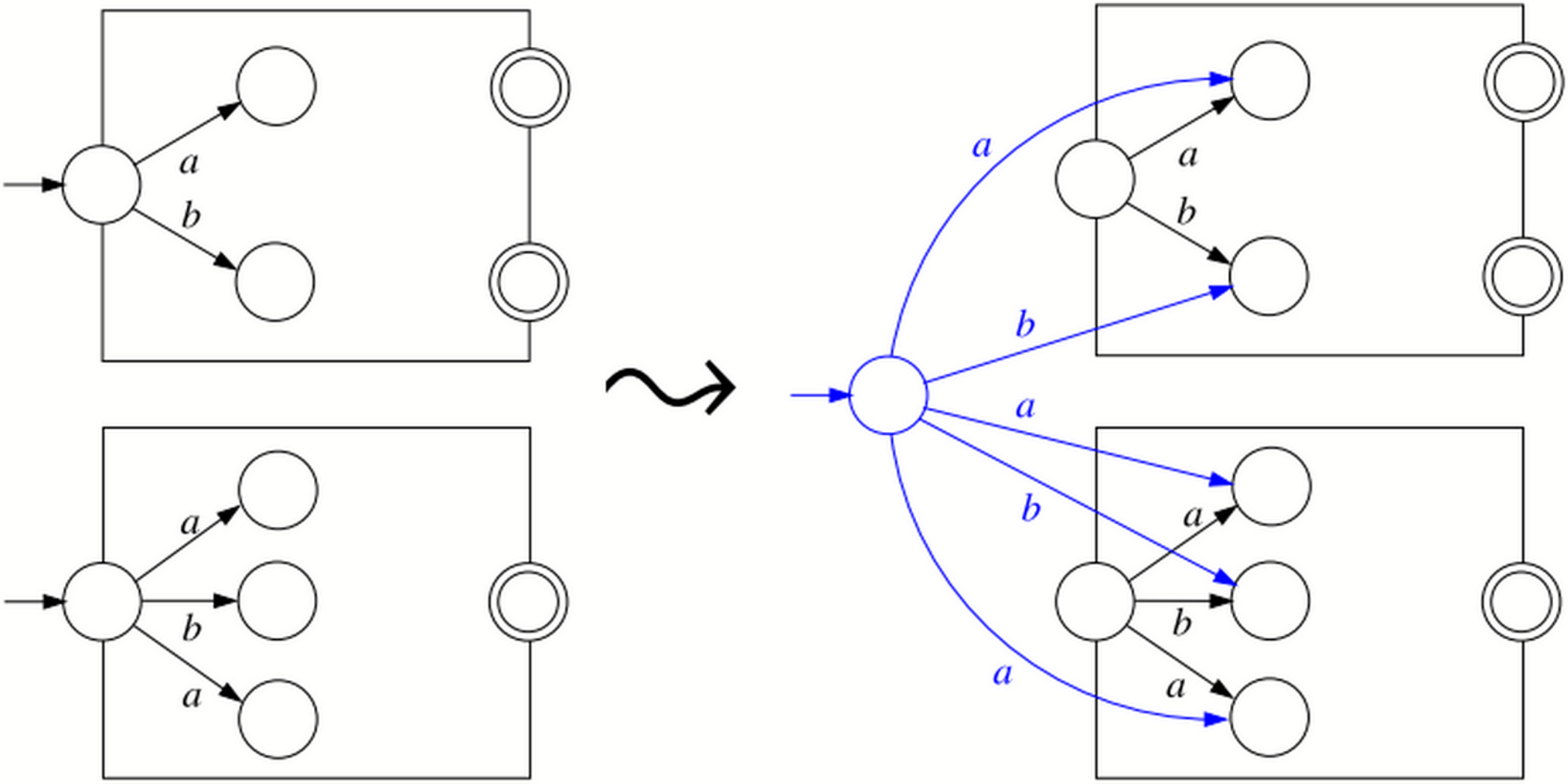
For the complexity, observe that in the worst case the algorithm must examine all pairs $[t_1, t_2]$ of transitions of $\delta_1 \times \delta_2$, but every pair is examined at most once. So the runtime is $\mathcal{O}(|\delta_1 \| \delta_2|)$.


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


1
a
2
a
3
a, b
a, b


a, b
a, b
a, b
11
12
13
14
15
a
a
a
a
a
a
a
a
a
a
6
7
8
9
10
a
a
a
a
a
a
a
a
a
a
1
2
3
4
5
a, b
a, b
a, b

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** δ
5 **for all** $(q_{02}, a, q) \in \delta_2$ **do**
6 **add** (q_0, a, q_2) **to** δ
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}.

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 $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$

```
1   Q ← Q_1 ∪ Q_2 ∪ {q_0}
2   δ ← δ_1 ∪ δ_2
3   F ← F_1 ∪ F_2
4   for all i = 1, 2 do
5       if q_{0i} ∈ F_i then add q_0 to F
6       for all (q_{0i}, a, q) ∈ δ_i do
7           add (q_0, a, q) to δ
8       if δ_i^{-1}(q_{0i}) = ∅ then
9           remove q_{0i} from Q
10          for all a ∈ Σ, q ∈ δ_i(q_{0i}, a) do
11              remove (q_{0i}, a, q) from δ_i
12  return (Q, Σ, δ, q_0, F)
```

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

- 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 $B = \mathit{NFAtoDFA}(A)$ 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 $\mathcal{L}(A(M, x)) = \Sigma^*$, and we are done. □

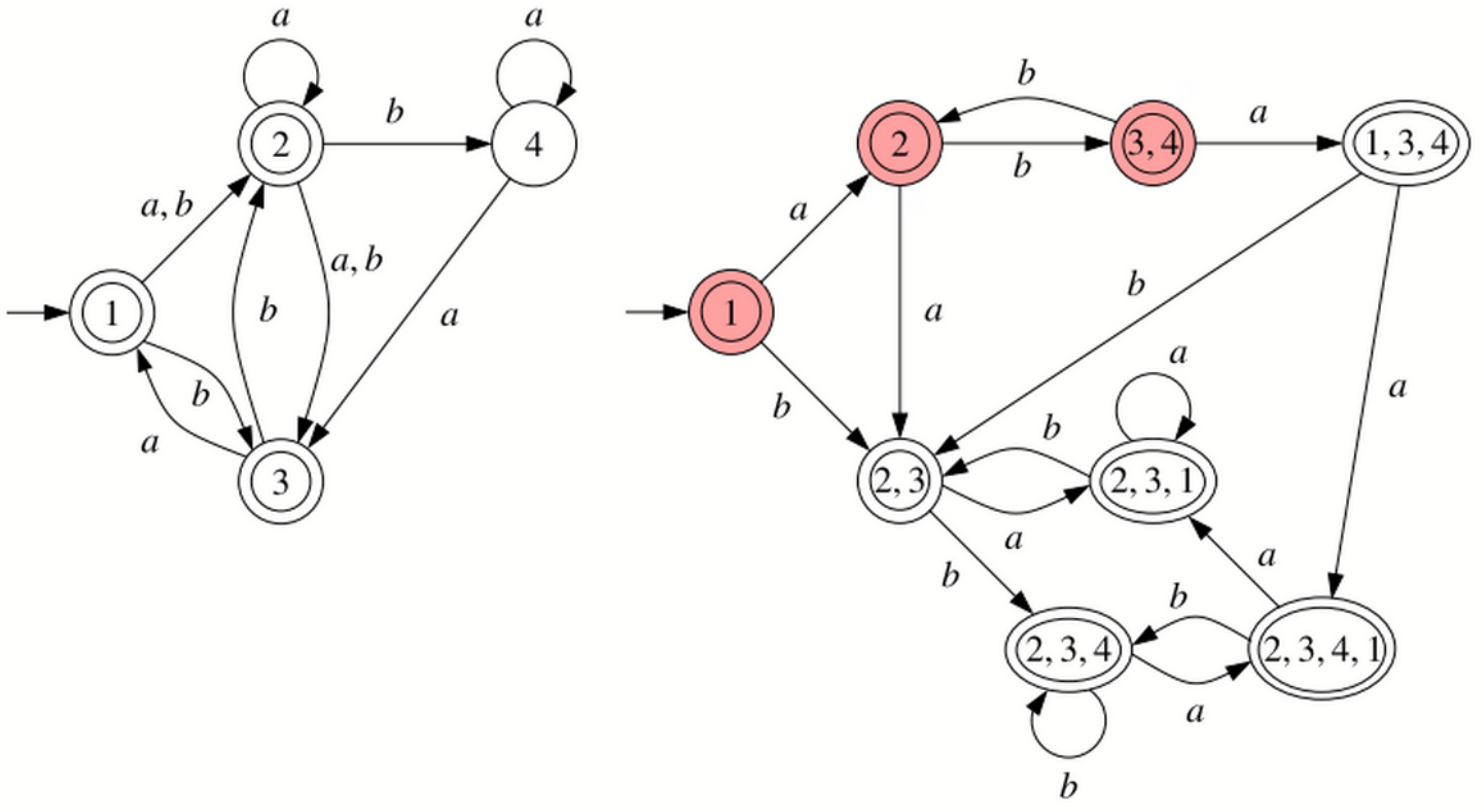
- Complement and check for emptiness
  - Exponential complexity

- Improvements:
  - Check for emptiness while complementing (on-the-fly check).
  - Subsumption test.

Definition 4.7 *Let A be a NFA, and let $B = NFAtoDFA(A)$. 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 $B = NFAtoDFA(A)$. 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. □

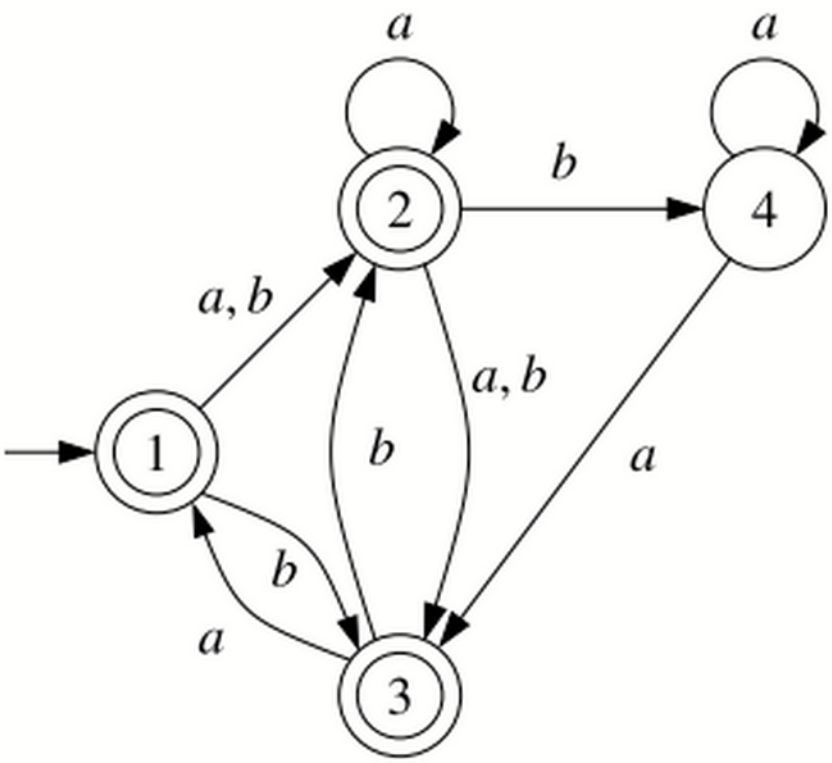


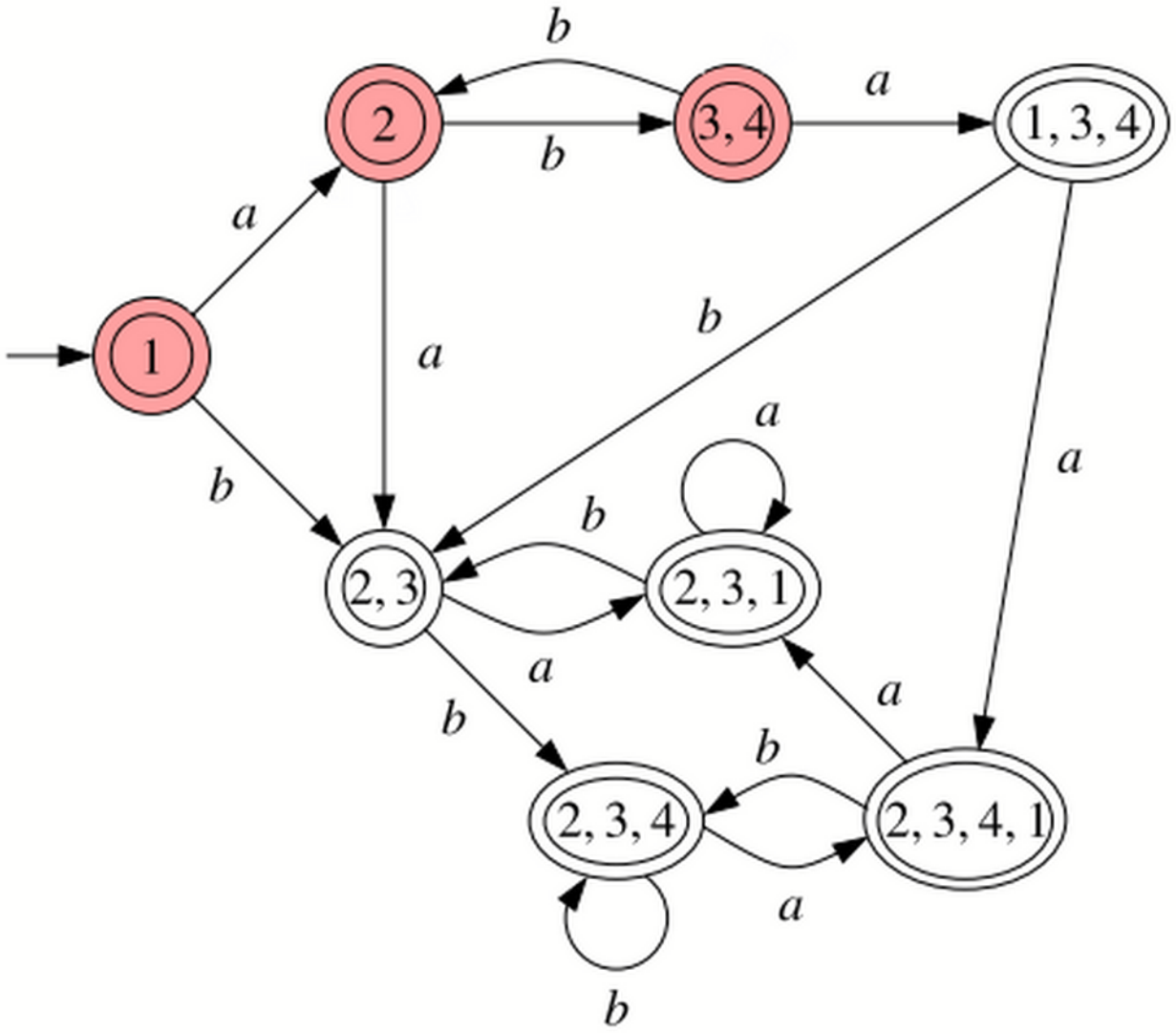
UnivNFA(A)

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

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

```
1  Q ← ∅;
2  W ← { {q0} }
3  while W ≠ ∅ do
4      pick Q' from W
5      if Q' ∩ F = ∅ then return false
6      add Q' to Q
7      for all a ∈ Σ do
8          if W ∪ Q contains no Q'' ⊆ δ(Q', a) then add δ(Q', a) to W
9  return true
```


a
a
b
2
4
a, b
a, b
1
b
a
b
a
3


b
2
3, 4
a
1, 3, 4
b
a
b
1
a
a
b
b
a
2, 3
2, 3, 1
a
a
b
b
2, 3, 4
2, 3, 4, 1
a
b

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* $B = NFAtoDFA(A)$. *After termination of* $UnivNFA(A)$, *the set* $\mathcal{Q}$ *contains all minimal states of* B.

- Proof: Assume the contrary.

Then B has a shortest path $Q_1 \to Q_2 \to \ \ldots \ \to Q_n$ such that

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

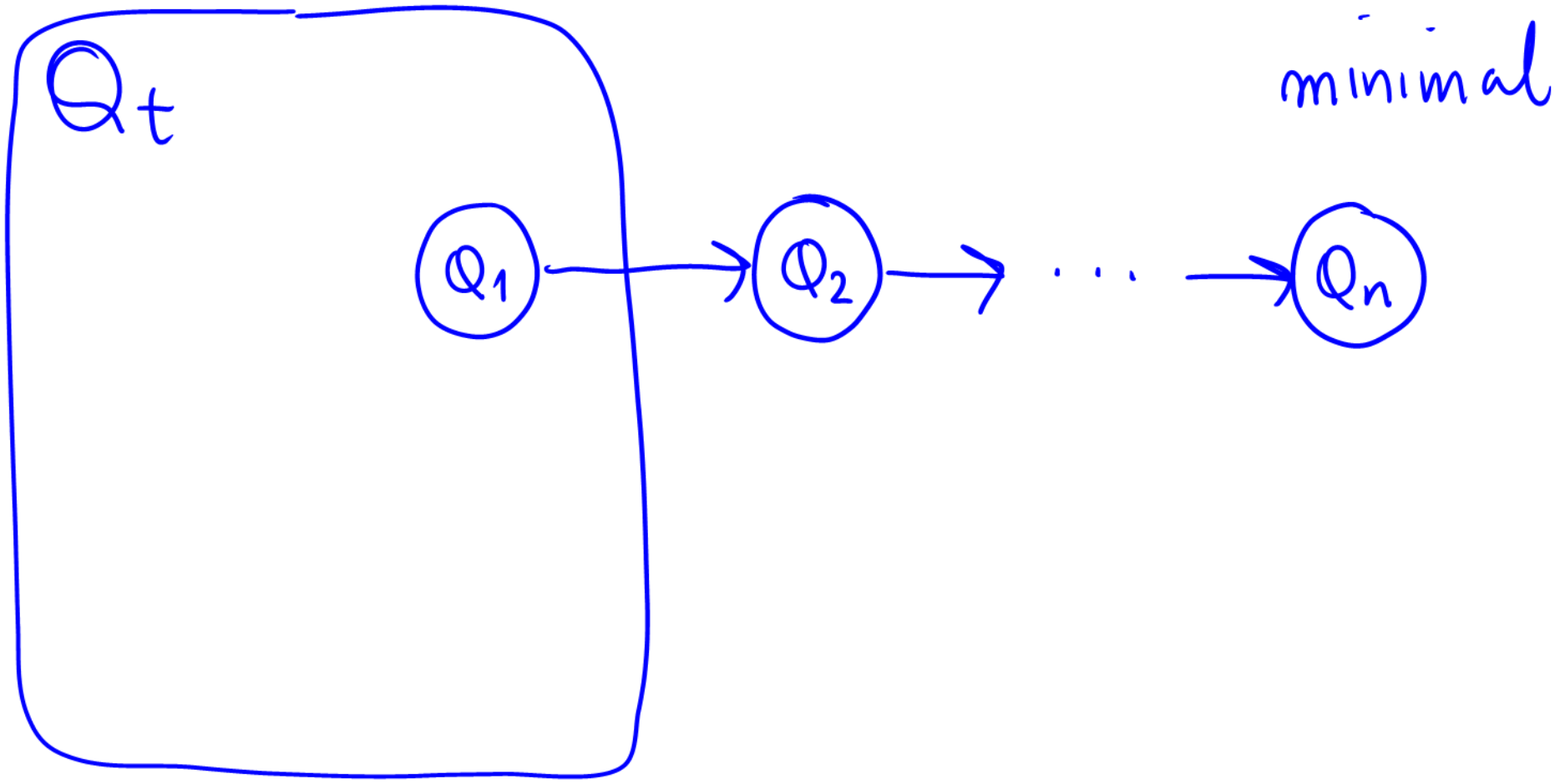
Since the path is shortest, $Q_2 \notin \mathcal{Q}$ and so when UnivNFA processes Q_1, it does not add Q_2. This can only be because UnivNFA already added some $Q_2' \subset Q_2$.

But then B has a path $Q_2' \to \ \ldots \ \to Q_n'$ with $Q_n' \subseteq Q_n$. Since Q_n is minimal, Q_n' is minimal (actually $Q_n' = Q_n$).

So the path $Q_2' \to \ \ldots \ \to Q_n'$ satisfies

- $Q_2 \in \mathcal{Q}$ (after termination), and
- Q_n' is minimal.

contradicting that $Q_1 \to Q_2 \to \ \ldots \ \to Q_n$ is shortest.


Q_t
Q_1
Q_2
Q_n
minimal

Inclusion and equality

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

Proof: We first prove membership in PSPACE. Since PSPACE=co-PSPACE=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 reached 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. □

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

- Concatenate four algorithms:
  (1) determinize A_2,
  (2) complement the result,
  (3) intersect it with A_1, and
  (4) check for emptiness.

- State of (3): pair (q, Q) , where $q \in Q_1$ and $Q \subseteq Q_2$

- Easy optimizations:
  - store only the states of (3), not its transitions;
  - do not perform (1), then (2), then (3): instead, 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 $B_2 = NFAtoDFA(A_2)$. 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 $Q_2' \subset Q'$.*

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 ← ∅;
2   W ← { [q01, {q02}] }
3   while W ≠ ∅ do
4       pick [q1, Q2] from W
5       if (q1 ∈ F1) and (Q2 ∩ F2 = ∅) then return false
6       add [q1, Q2] to Q
7       for all a ∈ Σ, q1' ∈ δ1(q1, a) do
8           Q2' ← δ2(Q2, a)
9           if W ∪ Q contains no [q1'', Q2''] s.t. q1'' = q1' and Q2'' ⊆ Q2' then
10              add [q1', Q2'] to W
11  return true
```

- Complexity:
  - Let A_1, A_2 be NFAs with n_1, n_2 states over an alphabet with k letters.
  - Without the subsumption test:
    - The while-loop is executed at most $n_1 \cdot 2^{n_2}$ times.
    - The for-loop is executed at most $O(k \cdot n_1)$ times.
    - An execution of the for-loop takes $O(n_2^2)$ time.
    - Overall: $O(k \cdot n_1^2 \cdot n_2^2 \cdot 2^{n_2})$ time.
  - With the subsumption case 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(n1^2 n2) time

Equality: check inclusion in both directions.