Operations and tests on sets: Implementation on DFAs

Operations and tests

Universe of objects U, sets of objects X, Y, object x.

Operations on sets		
Complement(X)	:	returns $U \setminus X$.
Intersection(X, Y)	:	returns $X \cap Y$.
Union(X, Y)	:	returns $X \cup Y$.

Tests on sets		
Member(x, X)	:	returns **true** if $x \in X$, **false** otherwise.
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.

Implementation on DFAs

- Assumption: each object encoded by one word, and vice versa.
- Membership: trivial algorithm, linear in the length of the word.
- Complement: exchange final and non-final states. Linear (or even constant) time.
- Generic implementation of binary boolean operations based on pairing.

Pairing

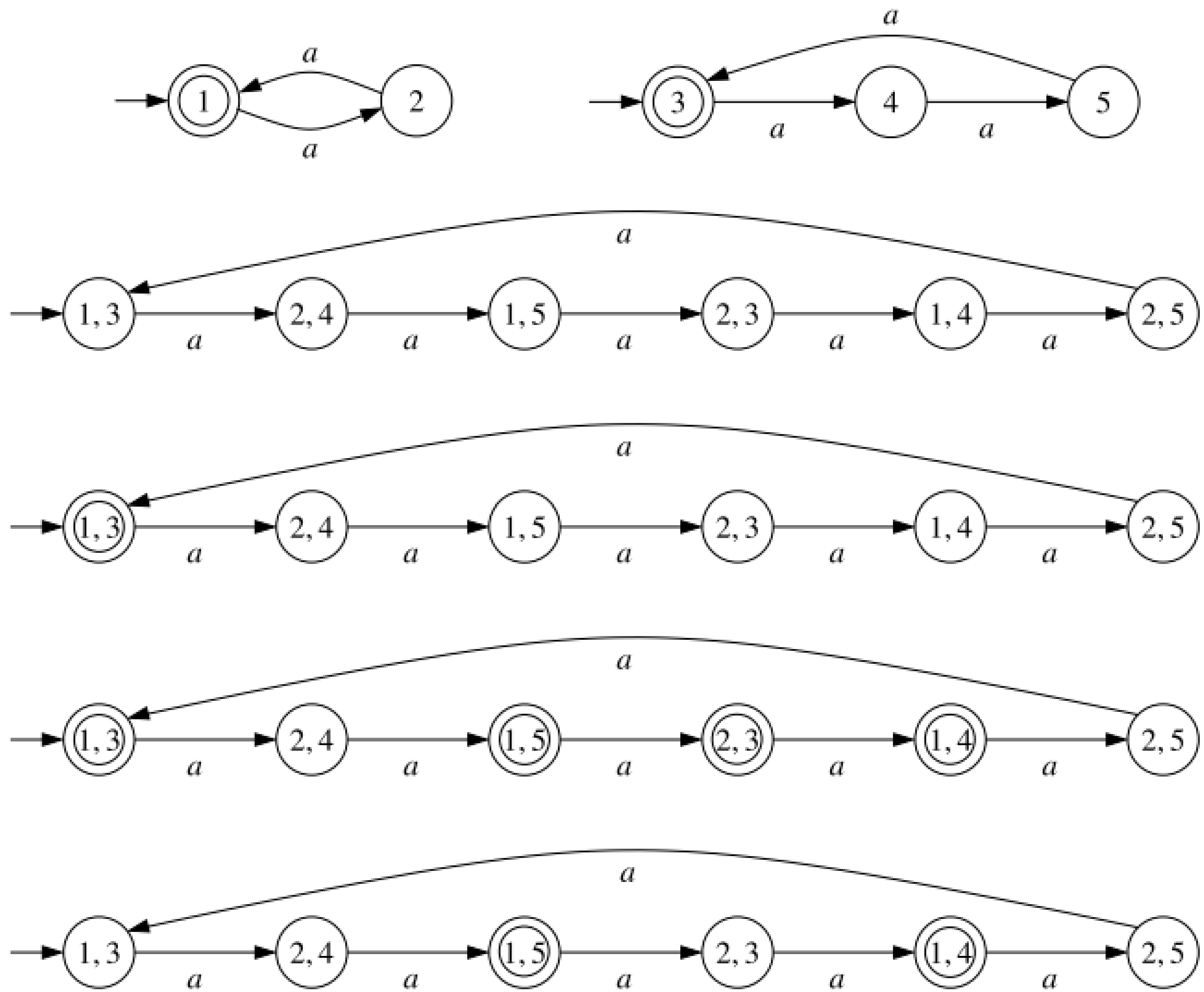
Definition. Let $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be DFAs.

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

Pairing



Pairing

- Another example: DFA for the language of words with an even number of as and even number of bs (and even number of cs ...).

Generic algorithm for binary boolean operations

- We assign to a binary boolean operator $\odot$ an operation on languages $\widehat{\odot}$ as follows:

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

- For example:

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 ($L_1 \setminus L_2 \cup L_2 \setminus L_1$)	$b_1 \Leftrightarrow \neg b_2$

Generic algorithm for binary boolean operations

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

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

$Q, \delta, F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, q_{02}]$
$W \leftarrow \{q_0\}$
while $W \neq \emptyset$ **do**
    pick $[q_1, q_2]$ **from** W
    add $[q_1, q_2]$ **to** Q
    if $(q_1 \in F_1) \odot (q_2 \in F_2)$ **then add** $[q_1, q_2]$ **to** F
    for all $a \in \Sigma$ **do**
        $q'_1 \leftarrow \delta_1(q_1, a)$; $q'_2 \leftarrow \delta_2(q_2, a)$
        if $[q'_1, q'_2] \notin Q$ **then add** $[q'_1, q'_2]$ **to** W
        add $([q_1, q_2], a, [q'_1, q'_2])$ **to** δ

Generic algorithm for binary boolean operations

- 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 and n_2 states over an alphabet with k letters, binary operations can be computed in $O(k \cdot n_1 \cdot n_2)$ time.
- Further: there is a family of languages for which the computation of intersection takes $\Theta(k \cdot n_1 \cdot n_2)$ 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 $(L_1 \setminus L_2) \cup (L_2 \setminus L_1) = \emptyset$

Inclusion test

InclDFA(A_1, A_2)

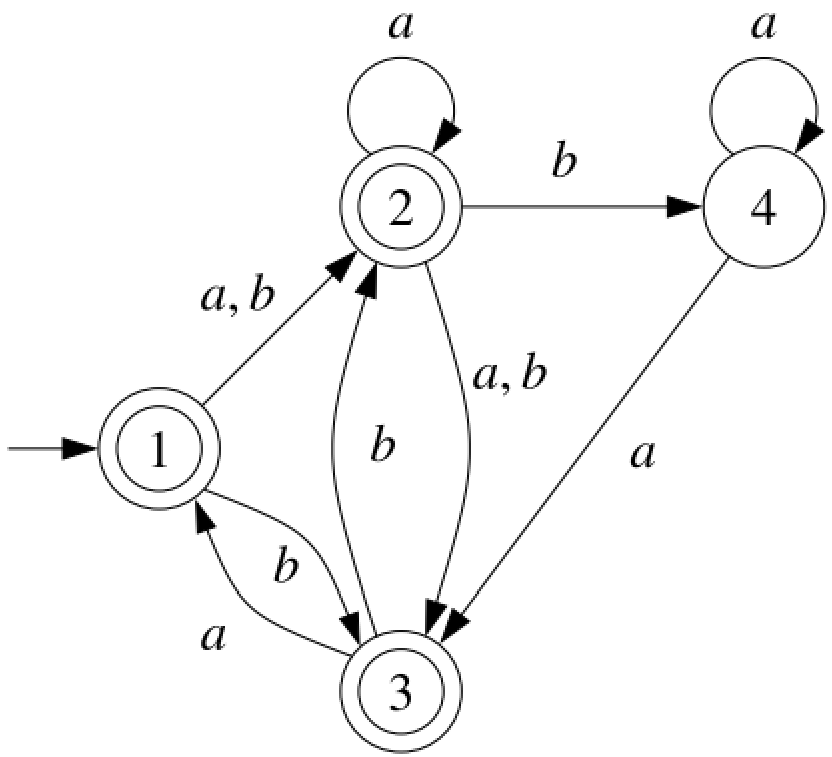
Input: DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

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

$Q \leftarrow \emptyset$;
$W \leftarrow \{[q_{01}, q_{02}]\}$
while $W \neq \emptyset$ **do**
    pick $[q_1, q_2]$ **from** W
    add $[q_1, q_2]$ **to** Q
    if $(q_1 \in F_1)$ **and** $(q_2 \notin F_2)$ **then return false**
    for all $a \in \Sigma$ **do**
        $q'_1 \leftarrow \delta_1(q_1, a)$; $q'_2 \leftarrow \delta_2(q_2, a)$
        if $[q'_1, q'_2] \notin Q$ **then add** $[q'_1, q'_2]$ **to** W
return true

Operations and tests on sets: Implementation on NFAs

Membership



Prefix read	W
ϵ	$\{1\}$
a	$\{2\}$
aa	$\{2, 3\}$
aaa	$\{1, 2, 3\}$
$aaab$	$\{2, 3, 4\}$
$aaabb$	$\{2, 3, 4\}$
$aaabba$	$\{1, 2, 3, 4\}$

Membership

$MemNFA[A](w)$

Input: NFA $A = (Q, \Sigma, \delta, Q_0, F)$, word $w \in \Sigma^*$,

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

$W \leftarrow Q_0$;
while $w \neq \varepsilon$ **do**
    $U \leftarrow \emptyset$
    for all $q \in W$ **do**
        add $\delta(q, head(w))$ **to** U
    $W \leftarrow U$
    $w \leftarrow tail(w)$
return $(W \cap F \neq \emptyset)$

Complexity:

- While loop executed $|w|$ times
- For loop executed at most $|Q|$ times
- Each execution of the loop body takes $O(|Q|)$ time

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

Complement

- Swapping final and non-final states does not work
- Solution: determinize <u>and then</u> 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

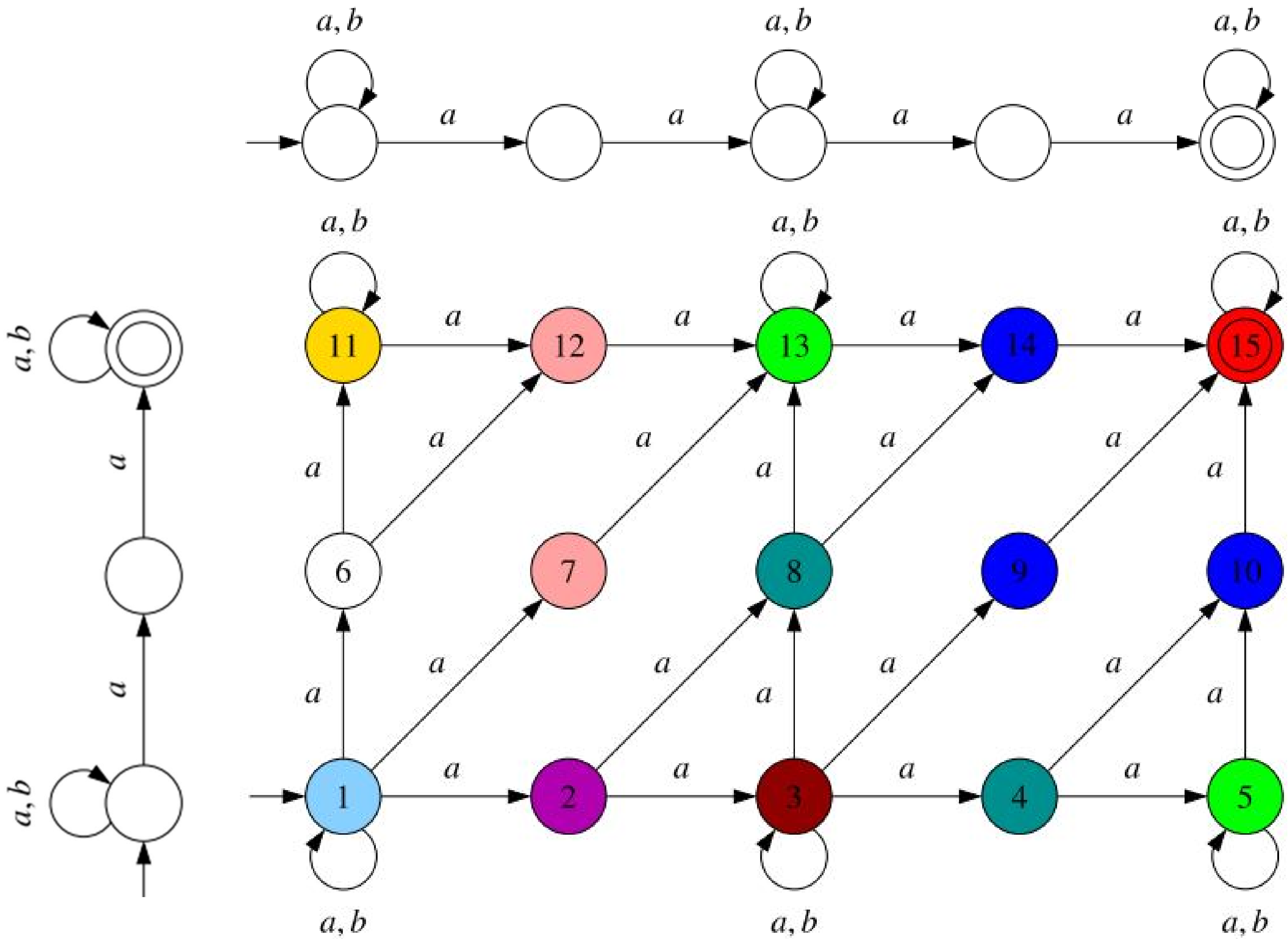
- The pairing construction still works for intersection, with the same complexity.
- It also works for union, but only if the NFAs are complete, i.e., they have at least one run for each word.
- Optimal construction for intersection (same example as for DFAs).
- Non-optimal construction for union. There is another construction which produces an NFA with $|Q_1| + |Q_2|$ states, instead of $|Q_1| \cdot |Q_2|$: just put the automata side by side!

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

$Q, \delta, F \leftarrow \emptyset$; $Q_0 \leftarrow Q_{01} \times Q_{02}$
$W \leftarrow Q_0$
while $W \neq \emptyset$ **do**
    pick $[q_1, q_2]$ **from** W
    add $[q_1, q_2]$ **to** Q
    if $(q_1 \in F_1)$ **and** $(q_2 \in F_2)$ **then add** $[q_1, q_2]$ **to** F
    for all $a \in \Sigma$ **do**
        for all $q_1' \in \delta_1(q_1, a), q_2' \in \delta_2(q_2, a)$ **do**
            if $[q_1', q_2'] \notin Q$ **then add** $[q_1', q_2']$ **to** W
            add $([q_1, q_2], a, [q_1', q_2'])$ **to** δ

Intersection



Emptiness and Universality

- Like DFAs, an NFA is empty iff every state is non-final.
- However, contrary to DFAs, it does not hold that an NFA is universal iff every state is final. Both directions fail!
- Emptiness is decidable in linear time.
- Universality is PSPACE-complete.

Crash course on PSPACE

- PSPACE: Class of decision problems for which there is an algorithm that
  - always terminates and returns the correct answer, and
  - only uses polynomial memory in the size of the input.
- P ⊆ NP ⊆ PSPACE. It is unknown if the inclusions are strict.
- NPSPACE: Class of decision problems for which there is a nondeterministic algorithm that
  - does not terminate or terminates and answers„no" for no-inputs,
  - has at least one terminating execution answering „yes" for yes-inputs, and
  - only uses polynomial memory in the size of the input.
- Savitch´s theorem: PSPACE=NPSPACE

Crash course on PSPACE

- PSPACE-complete: A problem Π is PSPACE-complete if
  - it belongs to PSPACE, and
  - every PSPACE-problem can be reduced in polynomial time to Π.
- PSPACE-complete problems:
  - Given a deterministic Turing machine M that only visits the cell tapes occupied by the input, and an input x, does M accept x ?
  - Is a given quantified boolean formula true?

Universality is PSPACE complete

Theorem 4.7 *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 $B = NFAtoDFA(A)$ 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 does not store the whole run, only the current state of B, and so it only needs linear space in the size of A.

Universality is PSPACE complete

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.

Universality is PSPACE complete

Let Σ be the alphabet used to encode the run of the machine. Given an input x, the machine accepts if there exists a word w of $(\Sigma \cup \{\#\})^*$ (we assume $\# \notin \Sigma$) 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 the machine.

Universality is PSPACE complete

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 $L(A_{M,x}) = \Sigma^* \setminus \{w_{M,x}\}$.
- If M rejects x, then no word encodes an accepting run of M on x, and so $L(A_{M,x}) = \Sigma^*$.

So M accepts x if and only if $L(A_{M,x}) = \Sigma^*$, and we are done. $\square$

Deciding universality of NFAs

- Complement and check for emptiness
  - Needs exponential time and space.
- Improvements:
  - Check for emptiness while complementing (on-the-fly check).
  - Subsumption test.

Subsumption test

- Let A be an NFA and let $B = NFAtoDFA(A)$. A state Q' of B is minimal if no other state Q'' satisfies $Q'' \subset Q'$.
- Proposition: A is universal iff every minimal state of B is final.

  Proof:

  A is universal

  iff B is universal

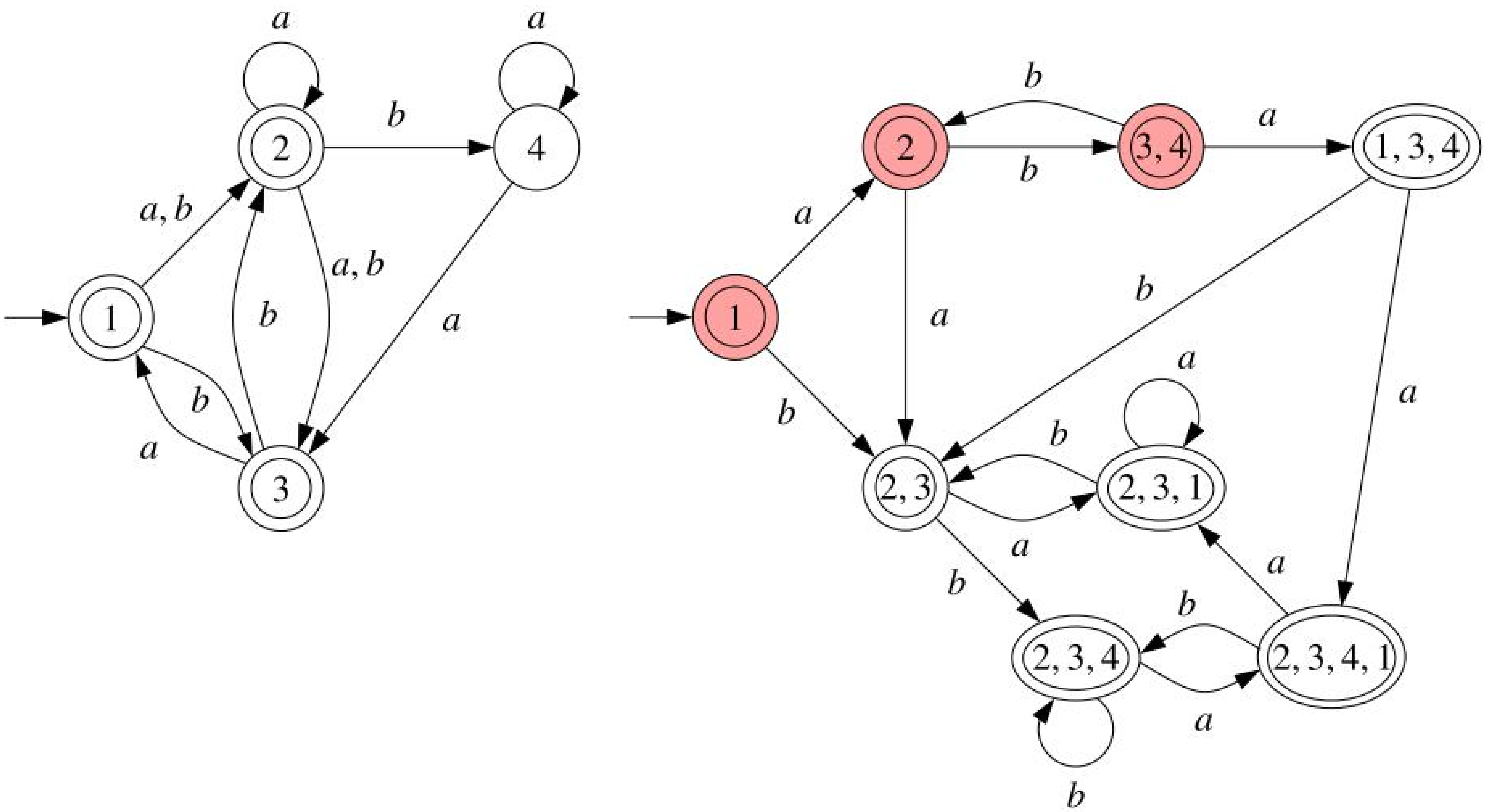
  iff every state of B is final

  iff every state of B contains a final state of A

  iff every minimal state of B contains a final state of A

  iff every minimal state of B is final

Subsumption test



Subsumption test

UnivNFA(A)

Input: NFA $A = (Q, \Sigma, \delta, Q_0, F)$

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

$\mathcal{Q} \leftarrow \emptyset$;
$\mathcal{W} \leftarrow \{ \{q_0\} \}$
while $\mathcal{W} \neq \emptyset$ **do**
    pick Q' **from** $\mathcal{W}$
    if $Q' \cap F = \emptyset$ **then return false**
    add Q' **to** $\mathcal{Q}$
    for all $a \in \Sigma$ **do**
        if $\mathcal{W} \cup \mathcal{Q}$ contains no $Q'' \subseteq \delta(Q', a)$ **then add** $\delta(Q', a)$ **to** $\mathcal{W}$
return true

Subsumption test

- But is it correct ?

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

Subsumption test

Proposition: Let A be an 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
- $Q_n \notin \mathcal{Q}$ and Q_n is minimal.

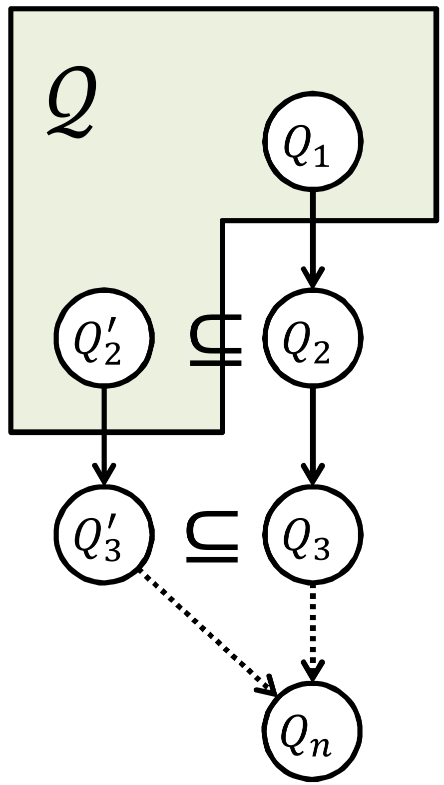
Since the path is shortest, $Q_2 \notin \mathcal{Q}$ and so when *UnivNFA* processes Q_1, it does not add Q_2. This can only be because UnivNFA already added some $Q_2' \subset Q_2$.

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.



Inclusion

- Proposition: The inclusion problem is PSPACE-complete.
- Proof:

Membership in PSPACE. By Savitch´s theorem it suffices to give a nondeterministic algorithm for non-inclusion. For this, guess letter by letter a word, storing the sets of states Q_1', Q_2' reached by both NFAs on the word guessed so far. Stop when Q_1'contains a final state, but Q_2' does not.

PSPACE-hardness. A is universal iff $L(A) \subseteq L(B)$, where B is the one-state DFA for Σ^*.

Deciding inclusion

- 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).

Deciding inclusion

- Further optimization: subsumption test.

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 $L(A_1) \subseteq L(A_2)$, **false** otherwise

$Q \leftarrow \emptyset$;
$W \leftarrow \{\ [q_{01}, Q_{02}] \mid q_{01} \in Q_{01}\ \}$
while $W \neq \emptyset$ **do**
    pick $[q_1, Q_2]$ **from** W
    if $(q_1 \in F_1)$ **and** $(Q_2 \cap F_2 = \emptyset)$ **then return false**
    add $[q_1, Q_2]$ **to** Q
    for all $a \in \Sigma, q'_1 \in \delta_1(q_1, a)$ **do**
        $Q'_2 \leftarrow \delta_2(Q_2, a)$
        if $\mathcal{W} \cup Q$ contains no $[q''_1, Q''_2]$ s.t. $q''_1 = q'_1$ and $Q''_2 \subseteq Q'_2$ **then**
            add $[q'_1, Q'_2]$ **to** $\mathcal{W}$
return true

Deciding inclusion

- 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.

Deciding inclusion

- Important special case: A_1 is an NFA, A_2 is a DFA.
  - Complementing A_2 is now easy.
  - The while-loop is executed $O(n_1 \cdot n_2)$ times.
  - The for-loop is executed k times.
  - An execution of the for-loop takes constant time.
  - Overall: $O(k \cdot n_1 \cdot n_2)$ time.

- Checking equality: check inclusion in both directions.