# 4. Implementing operations on sets using finite automata

## 4.1 Implementation using DFAs

**Recall:**

| | | |
|---|---|---|
| **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 |

We assume that each object (input, automaton, etc.) is encoded by one word.

We observe:

| | | |
|---|---|---|
| Membership | : | trivial, linear for fixed automaton |
| | | uniform word problem: low polynomial |
| Complement | : | trivial, swap final and non-final states |
| | | linear (or even constant) time |

Also consider these set operations:

| | | |
|---|---|---|
| **Intersection**$(X, Y)$ | : | returns $X \cap Y$ |
| **Union**$(X, Y)$ | : | returns $X \cup Y$ |
| **SetDifference**$(X, Y)$ | : | returns $X \setminus Y$ |
| **SymmetricSetDifference**$(X, Y)$ | : | returns $X \vartriangle Y$ |
| **Op**$(X, Y, Z)$ | : | returns $(X \cup Y) \setminus Z$ |

## The **product construction** or **pairing** for DFAs

Two DFAs run synchronously in parallel, an input word is accepted iff both automata accept it.

### Theorem 27

*Let* $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ *and* $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ *be two DFAs. Then the product automaton or pairing* $M = [M_1, M_2]$ *of* $M_1$ *and* $M_2$*, defined by*

$$M := (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$

*with* $\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$ *for all* $q_1 \in Q_1, q_2 \in Q_2$ *and* $a \in \Sigma$*, is a DFA recognizing* $L(M_1) \cap L(M_2)$*.*

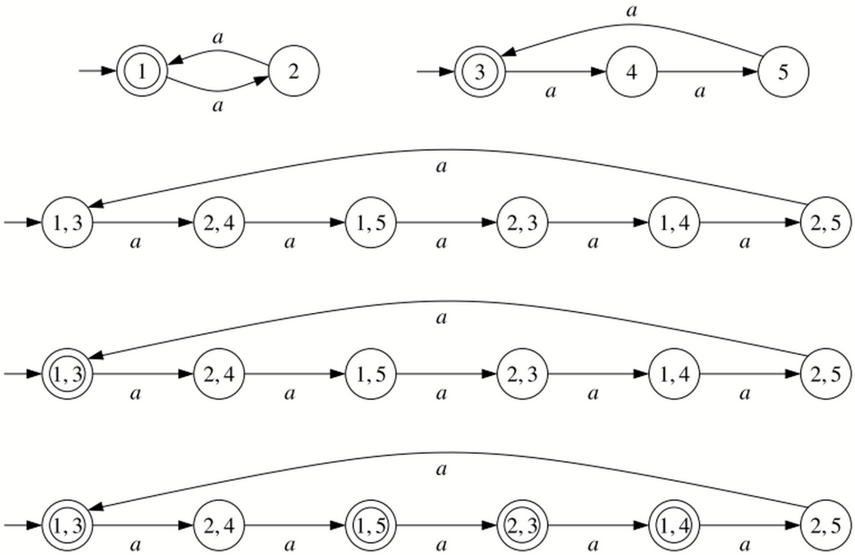
## Proof.

Induction on $|w|$. We have:

$$\begin{aligned}
w \in L(M) &\Leftrightarrow \hat{\delta}((s_1, s_2), w) \in F_1 \times F_2 \\
&\Leftrightarrow (\hat{\delta}_1(s_1, w), \hat{\delta}_2(s_2, w)) \in F_1 \times F_2 \\
&\Leftrightarrow \hat{\delta}_1(s_1, w) \in F_1 \wedge \hat{\delta}_2(s_2, w) \in F_2 \\
&\Leftrightarrow w \in L(M_1) \wedge w \in L(M_2) \\
&\Leftrightarrow w \in L(M_1) \cap L(M_2)\,.
\end{aligned}$$

□

**Question:** Does the pairing construction (for intersection) also work for NFAs?


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

## Definition 28

The reversal(mirror) of a word $w = a_1 \cdots a_n$ is

$$w^R := a_n \cdots a_1 \,.$$

The reversal of a language $L$ is

$$L^R := \{w^R ; w \in L\} \,.$$

## Theorem 29

*If $L$ is a regular language, so is $L^R$.*

## Proof.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $L = L(M)$. We construct an $\epsilon$-NFA $N = (Q \uplus \{q_0'\}, \Sigma, \delta', q_0', \{q_0\})$ as follows:

- we reverse all state transitions, *i.e.*, $\delta(q, a) = p$ iff $q \in \delta'(p)$;
- we create the new start state $q_0'$ of $N$, with $\epsilon$-transitions to all $f \in F$;
- $q_0$ becomes the (only) final state of $N$.

Following the state transitions of $M$ on some arbitrary input $w \in \Sigma^*$ backwards, we easily see that

$$L(N) = L^R\,.$$

□

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

$Q \leftarrow \emptyset$; $F \leftarrow \emptyset$

$q_0 \leftarrow [q_{01}, q_{02}]$

$W \leftarrow \{q_0\}$

**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** $\delta$

**return** $(Q, \Sigma, \delta, q_0, F)$

**Observation:**

- The product automaton/pairing of two DFAs with $n_1$ resp. $n_2$ states has (in normal form) $O(n_1 \cdot n_2)$ states.
- Hence, for DFAs with $n_1$ resp. $n_2$ states and an alphabet $\Sigma$ with $k$ letters, the operations union, intersection, etc. can be carried out in $O(k \cdot n_1 \cdot n_2)$ time.

## Language tests

Let $A, A_1$, and $A_2$ be DFAs, with $L = L(A)$, $L_1 = L(A_1)$, and $L_2 = L(A_2)$ the languages recognized by them, respectively. Note that we assume that all these automata are in normal form!

Then we have

- Emptiness: $L$ is empty iff $A$ has no final states.
- Universality: $L = \Sigma^*$ iff $A$ has only final states.
- Inclusion: $L_1 \subseteq L_2$ iff $L_1 \setminus L_2 = \emptyset$.
- Equality: $L_1 = L_2$ iff $L_1 \triangle L_2 = \emptyset$.

*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

$Q \leftarrow \emptyset$;

$W \leftarrow \{[q_{01}, q_{02}]\}$

**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** $W$

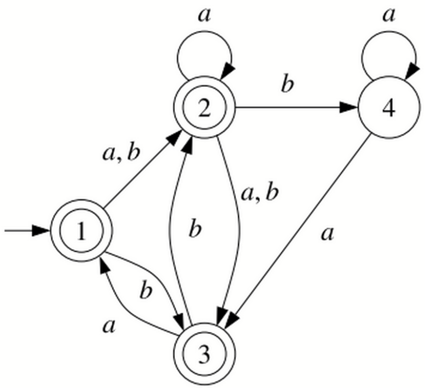
**return true**

# 4.2 Implementation using NFAs

**Recall:**

| | | |
|---|---|---|
| **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 | $W$ |
|---|---|
| $\epsilon$ | $\{q_0\}$ |
| $a$ | $\{q_2\}$ |
| $aa$ | $\{q_2, q_3\}$ |
| $aaa$ | $\{q_1, q_2, q_3\}$ |
| $aaab$ | $\{q_2, q_3\}$ |
| $aaabb$ | $\{q_2, q_3, q_4\}$ |
| $aaabba$ | $\{q_1, q_2, q_3, q_4\}$ |

$Mem[A](w)$

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

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

$W \leftarrow \{q_0\}$;
**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 takes O(|Q|) time

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

## Complement:

- Swapping final and non-final states does not work.
- Solution: convert to DFA and then swap states.
- Problem: exponential blow-up of size of automaton!
  Hence try to avoid this whenever possible!
- However, in the worst case there is no better way: There are NFAs with $n$ states such that any minimal NFA for their complement has $\Theta(2^n)$ states!

**Union and intersection:**

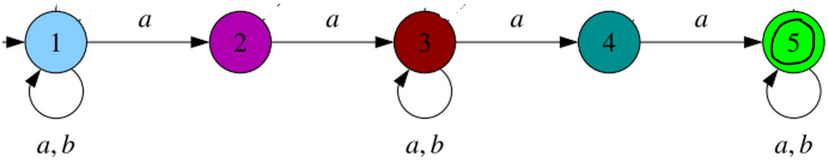
The product/pairing construction still works for union and intersection, with the same complexity, but (of course(!)) not for set difference or other non-monotonic operations.

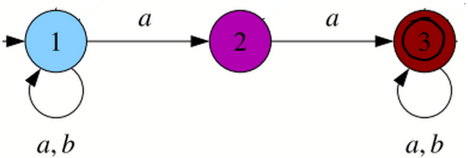
There is a better construction for union (see a few slides down), but not for intersection.

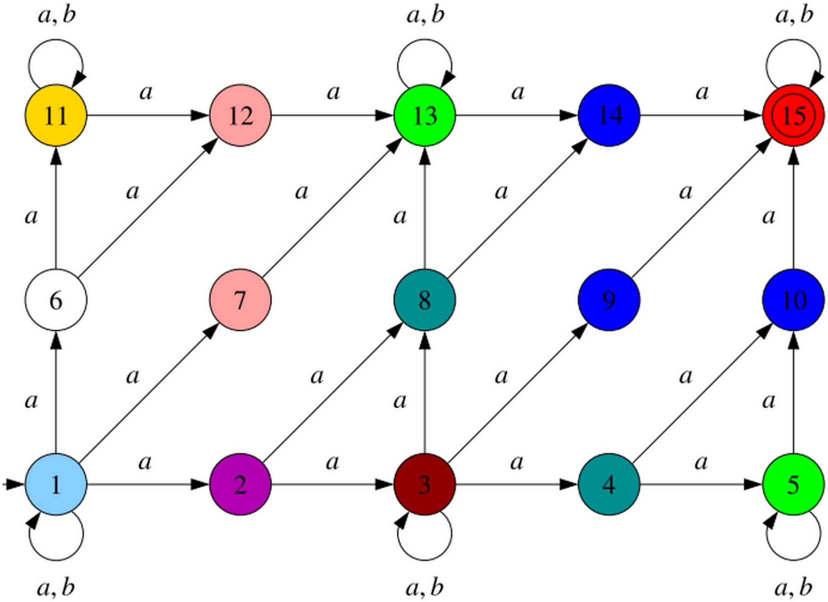
*IntersNFA*$(A_1, A_2)$
**Input:** NFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
**Output:** NFA $A_1 \cap A_2 = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}(A_1 \cap A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$

$Q \leftarrow \emptyset$; $F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, q_{02}]$
$W \leftarrow \{ [q_{01}, q_{02}] \}$
**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** $q_2 \in F_2)$ **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** $\delta$
**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 $\mathcal{O}(|\delta_1||\delta_2|)$.


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


a
a
1
2
3
a, b
a, b


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