Operations and tests on sets: Implementation on DFAs
## Operations and tests

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

<table>
<thead>
<tr>
<th>Operations on sets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Complement}(X)$</td>
<td>returns $U \setminus X$.</td>
</tr>
<tr>
<td>$\text{Intersection}(X, Y)$</td>
<td>returns $X \cap Y$.</td>
</tr>
<tr>
<td>$\text{Union}(X, Y)$</td>
<td>returns $X \cup Y$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tests on sets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Member}(x, X)$</td>
<td>returns $\text{true}$ if $x \in X$, $\text{false}$ otherwise.</td>
</tr>
<tr>
<td>$\text{Empty}(X)$</td>
<td>returns $\text{true}$ if $X = \emptyset$, $\text{false}$ otherwise.</td>
</tr>
<tr>
<td>$\text{Universal}(X)$</td>
<td>returns $\text{true}$ if $X = U$, $\text{false}$ otherwise.</td>
</tr>
<tr>
<td>$\text{Included}(X, Y)$</td>
<td>returns $\text{true}$ if $X \subseteq Y$, $\text{false}$ otherwise.</td>
</tr>
<tr>
<td>$\text{Equal}(X, Y)$</td>
<td>returns $\text{true}$ if $X = Y$, $\text{false}$ otherwise.</td>
</tr>
</tbody>
</table>
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 $(Q, \Sigma, \delta, q_0)$ 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 $\Sigma^*$ is defined as for DFAs.
Pairing

• Another example: DFA for the language of words with an even number of $a$s and even number of $b$s (and even number of $c$s ...).
Generic algorithm for binary boolean operations

- We assign to a binary boolean operator $\odot$ an operation on languages $\widehat{\odot}$ as follows:
  \[ L_1 \widehat{\odot} L_2 = \{ w \in \Sigma^* \mid (w \in L_1) \odot (w \in L_2) \} \]

- For example:

<table>
<thead>
<tr>
<th>Language operation</th>
<th>$b_1 \odot b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>$b_1 \lor b_2$</td>
</tr>
<tr>
<td>Intersection</td>
<td>$b_1 \land b_2$</td>
</tr>
<tr>
<td>Set difference ($L_1 \setminus L_2$)</td>
<td>$b_1 \land \overline{b_2}$</td>
</tr>
<tr>
<td>Symmetric difference ($L_1 \setminus L_2 \cup L_2 \setminus L_1$)</td>
<td>$b_1 \iff \overline{b_2}$</td>
</tr>
</tbody>
</table>
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 \( L(A) = L(A_1) \odot L(A_2) \)

1. \( Q, \delta, F \leftarrow \emptyset \)
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** \( \delta \)
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

\textbf{InclDFA}(A_1, A_2)

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

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

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

<table>
<thead>
<tr>
<th>Prefix read</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>${1}$</td>
</tr>
<tr>
<td>$a$</td>
<td>${2}$</td>
</tr>
<tr>
<td>$aa$</td>
<td>${2, 3}$</td>
</tr>
<tr>
<td>$aaa$</td>
<td>${1, 2, 3}$</td>
</tr>
<tr>
<td>$aaab$</td>
<td>${2, 3, 4}$</td>
</tr>
<tr>
<td>$aaabb$</td>
<td>${2, 3, 4}$</td>
</tr>
<tr>
<td>$aaabba$</td>
<td>${1, 2, 3, 4}$</td>
</tr>
</tbody>
</table>
Membership

\[ MemNFA[A](w) \]

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

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

1. \( W \leftarrow Q_0; \)
2. while \( w \neq \varepsilon \) do
3. \( U \leftarrow \emptyset \)
4. for all \( q \in W \) do
5. \( \text{add } \delta(q, \text{head}(w)) \text{ to } U \)
6. \( W \leftarrow U \)
7. \( w \leftarrow \text{tail}(w) \)
8. 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 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.
• 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

\[ \text{IntersNFA}(A_1, A_2) \]

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

1. \( Q, \delta, F \leftarrow \emptyset; Q_0 \leftarrow Q_{01} \times Q_{02} \)
2. \( W \leftarrow Q_0 \)
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 \((q_2 \in F_2)\) then add \([q_1, q_2]\) to \( F \)
7.   for all \( a \in \Sigma \) do
8.     for all \( q'_1 \in \delta_1(q_1, a), q'_2 \in \delta_2(q_2, a) \) do
9.       if \([q'_1, q'_2] \notin Q\) then add \([q'_1, q'_2]\) to \( W \)
10.    add \(([q_1, q_2], a, [q'_1, q'_2])\) to \( \delta \)
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 \( \Pi \) is PSPACE-complete if
  • it belongs to PSPACE, and
  • every PSPACE-problem can be reduced in polynomial time to \( \Pi \).

• 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 only does not store the whole run, only the current state, 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 $\Sigma$ be the alphabet used to encode the run of the machine. Given an input $x$, $M$ accepts if there exists a word $w$ of $\Sigma^*$ 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 $M$. 
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 $\Sigma^*$ 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.
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 $\Sigma^*$ 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.
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 $B = \text{NFAtoDFA}(A)$. A state $Q'$ of $B$ is minimal if no other state $Q''$ satisfies $Q'' \subseteq Q'$.
- **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

1 \( \mathcal{Q} \leftarrow \emptyset; \)
2 \( \mathcal{W} \leftarrow \{ \{q_0\} \} \)
3 \( \text{while } \mathcal{W} \neq \emptyset \text{ do} \)
4 \( \text{pick } Q' \text{ from } \mathcal{W} \)
5 \( \text{if } Q' \cap F = \emptyset \text{ then return false} \)
6 \( \text{add } Q' \text{ to } \mathcal{Q} \)
7 \( \text{for all } a \in \Sigma \text{ do} \)
8 \( \text{if } \mathcal{W} \cup \mathcal{Q} \text{ contains no } Q'' \subseteq \delta(Q', a) \text{ then add } \delta(Q', a) \text{ to } \mathcal{W} \)
9 \( \text{return true} \)
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!
Proposition: Let $A$ be an NFA and let $B = NFAtoDFA(A)$. After termination of $UnivNFA(A)$ the set $Q$ 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
- $Q_1 \in Q$ (after termination), and
- $Q_n \notin Q$ and $Q_n$ is minimal.

Since the path is shortest, $Q_2 \notin Q$ 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
- $Q_2 \in Q$ (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 $\Sigma^*$. 
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

```plaintext
1 \( Q \leftarrow \emptyset; \)
2 \( W \leftarrow \{ [q_{01}, Q_{02}] \mid q_{01} \in Q_{01} \} \)
3 \text{while } W \neq \emptyset \text{ do}
4 \hspace{1em} \text{pick } [q_1, Q_2] \text{ from } W
5 \hspace{1em} \text{if } (q_1 \in F_1) \text{ and } (Q_2 \cap F_2 = \emptyset) \text{ then return false}
6 \hspace{1em} \text{add } [q_1, Q_2] \text{ to } Q
7 \hspace{1em} \text{for all } a \in \Sigma, q'_1 \in \delta_1(q_1, a) \text{ do}
8 \hspace{1em} \hspace{1em} Q'_2 \leftarrow \delta_2(Q_2, a)
9 \hspace{1em} \hspace{1em} \text{if } W \cup Q \text{ contains no } [q''_1, Q''_2] \text{ s.t. } q''_1 = q'_1 \text{ and } Q''_2 \subseteq Q'_2 \text{ then}
10 \hspace{1em} \hspace{1em} \hspace{1em} \text{add } [q'_1, Q'_2] \text{ to } W
11 \hspace{1em} \text{return } \text{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: \( A_1 \) 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.