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><strong>Complement</strong>$(X)$</td>
<td>returns $U \setminus X$.</td>
</tr>
<tr>
<td><strong>Intersection</strong>$(X, Y)$</td>
<td>returns $X \cap Y$.</td>
</tr>
<tr>
<td><strong>Union</strong>$(X, Y)$</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><strong>Member</strong>$(x, X)$</td>
<td>returns <strong>true</strong> if $x \in X$, <strong>false</strong> otherwise.</td>
</tr>
<tr>
<td><strong>Empty</strong>$(X)$</td>
<td>returns <strong>true</strong> if $X = \emptyset$, <strong>false</strong> otherwise.</td>
</tr>
<tr>
<td><strong>Universal</strong>$(X)$</td>
<td>returns <strong>true</strong> if $X = U$, <strong>false</strong> otherwise.</td>
</tr>
<tr>
<td><strong>Included</strong>$(X, Y)$</td>
<td>returns <strong>true</strong> if $X \subseteq Y$, <strong>false</strong> otherwise.</td>
</tr>
<tr>
<td><strong>Equal</strong>$(X, Y)$</td>
<td>returns <strong>true</strong> if $X = Y$, <strong>false</strong> otherwise.</td>
</tr>
</tbody>
</table>
Implementation on DFAs

• Assumption: each object encoded by one word, and vice versa.
Implementation on DFAs

• Assumption: each object encoded by one word, and vice versa.

• Membership: trivial algorithm, linear in the length of the word.
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.
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:

\[
L_1 \widehat{\odot} L_2 = \{ w \in \Sigma^* \mid (w \in L_1) \odot (w \in L_2) \}
\]
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>$b_1 \land \neg b_2$</td>
</tr>
<tr>
<td>Symmetric difference ($L_1 \setminus L_2 \cup L_2 \setminus L_1$)</td>
<td>$b_1 \leftrightarrow \neg b_2$</td>
</tr>
</tbody>
</table>
Generic algorithm for binary boolean operations

\[ \text{BinOp}[\odot](A_1, A_2) \]

**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

\textit{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. \quad \textbf{pick} \([q_1, q_2]\) \textbf{from} \(W\)
5. \quad \textbf{add} \([q_1, q_2]\) \textbf{to} \(Q\)
6. \quad \textbf{if} \((q_1 \in F_1) \text{ and } (q_2 \notin F_2)\) \textbf{then} \textbf{return} \textbf{false}\n7. \quad \textbf{for all} \(a \in \Sigma\) \textbf{do}
8. \quad \quad \(q'_1 \leftarrow \delta_1(q_1, a); q'_2 \leftarrow \delta_2(q_2, a)\)
9. \quad \quad \textbf{if} \([q'_1, q'_2]\) \notin Q \textbf{then add} \([q'_1, q'_2]\) \textbf{to} \(W\)
10. \quad \textbf{return} \textbf{true}
Operations and tests on sets: Implementation on NFAs
Membership

Prefix read | $W$
---|---
$\varepsilon$ | $\{1\}$
$a$ | $\{2\}$
$aa$ | $\{2, 3\}$
$aaa$ | $\{1, 2, 3\}$
$aaab$ | $\{2, 3, 4\}$
$aaabb$ | $\{2, 3, 4\}$
$aaabba$ | $\{1, 2, 3, 4\}$
Membership

\[ \text{MemNFA}[A](w) \]

**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. \textbf{while} \( w \neq \varepsilon \) \textbf{do}
3. \hspace{1em} \( U \leftarrow \emptyset \)
4. \hspace{1em} \textbf{for all} \( q \in W \) \textbf{do}
5. \hspace{2em} \textbf{add} \( \delta(q, \text{head}(w)) \) \textbf{to} \( U \)
6. \hspace{1em} \( W \leftarrow U \)
7. \hspace{1em} \( w \leftarrow \text{tail}(w) \)
8. \hspace{1em} \textbf{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.
Let $\Sigma = \{a, b\}$. For every $n \geq 1$, let $L_n$ be the language of the regular expression

$$\Sigma^* (a\Sigma^{n-1}b + b\Sigma^{n-1}a)\Sigma^*$$

**Proposition:** For every $n \geq 1$, there exists a NFA for $L_n$ with $2n + 2$ states.

**Proposition:** For every $n \geq 1$, every NFA for $\overline{L_n}$ has at least $2^n$ states.
\( L_n \quad "=\quad \Sigma^* (a\Sigma^{n-1} b + b\Sigma^{n-1} a)\Sigma^* \)

Proposition: For every \( n \geq 1 \), there exists a NFA for \( L_n \) with \( 2n + 2 \) states.
Proposition: For every \( n \geq 1 \), every NFA for \( \overline{L_n} \) has at least \( 2^n \) states.
\[ L_n \ = \ \Sigma^* (a\Sigma^{n-1}b + b\Sigma^{n-1}a)\Sigma^* \]

**Proposition:** For every \( n \geq 1 \), every NFA for \( \overline{L_n} \) has at least \( 2^n \) states.

**Proof.** Observe: \( ww \in \overline{L_n} \) for every \( w \in \Sigma^n \).

Take an arbitrary NFA for \( \overline{L_n} \).

For every \( w \in \Sigma^n \) let \( q_w \) be the state reached after reading \( w \) in an accepting run of \( ww \).

For every \( w, v \in \Sigma^n \) we have:
\[ w \neq v \implies q_w \neq q_v \]
Union and intersection

• The pairing construction still works for intersection, with the same complexity.
Union and intersection

• The pairing construction still works for intersection, with the same complexity.
• Does it also work for union?
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.
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).
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).
- Is the construction optimal for union?
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

\textit{IntersNFA}(A_1, A_2)

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

\textbf{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. \textbf{while } W \neq \emptyset \textbf{ do}
4. \hspace{1em} \textbf{pick } [q_1, q_2] \textbf{ from } W
5. \hspace{1em} \textbf{add } [q_1, q_2] \textbf{ to } Q
6. \hspace{2em} \textbf{if } (q_1 \in F_1) \textbf{ and } (q_2 \in F_2) \textbf{ then add } [q_1, q_2] \textbf{ to } F
7. \hspace{2em} \textbf{for all } a \in \Sigma \textbf{ do}
8. \hspace{3em} \textbf{for all } q'_1 \in \delta_1(q_1, a), q'_2 \in \delta_2(q_2, a) \textbf{ do}
9. \hspace{4em} \textbf{if } [q'_1, q'_2] \notin Q \textbf{ then add } [q'_1, q'_2] \textbf{ to } W
10. \hspace{4em} \textbf{add } ([q_1, q_2], a, [q'_1, q'_2]) \textbf{ 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.
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.

- **$P \subseteq NP \subseteq PSPACE$**. It is unknown if the inclusions are strict.
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.
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 is PSPACE-complete if
  - it belongs to PSPACE, and
  - It is PSPACE-hard, meaning: every problem in PSPACE can be reduced in polynomial time to it.
Crash course on PSPACE

• **PSPACE-complete**: A problem is PSPACE-complete if
  • it belongs to PSPACE, and
  • It is PSPACE-hard, meaning: every problem in PSPACE can be reduced in polynomial time to it.

• **PSPACE-complete problems**:
  • Acceptance of linearly bounded automata (LBA): Given a LBA, i.e., a deterministic Turing machine $M$ that only visits the cell tapes occupied by the input, and an input $x$, does $M$ accept $x$?
  • **QBF**: Is a given quantified boolean formula true?
Universality is PSPACE complete

Universality is in PSPACE
Universality is PSPACE complete

Universality is in PSPACE

By Savitch‘s theorem it suffices to show that (non)-universality is in NPSPACE.
Universality is PSPACE complete

Universality is in PSPACE

By Savitch‘s theorem it suffices to show that (non)-universality is in NPSPACE.

So it suffices to give a nondeterministic algorithm that, given an NFA $A$ as input:

• does not terminate if $A$ is universal,
• has at least one terminating execution answering „non-universal“ if $A$ is not universal, and
• only uses polynomial memory in the size of the input.
Universality is PSPACE complete

Universality is in PSPACE

By Savitch‘s theorem it suffices to show that (non)-universality is in NPSPACE.

So it suffices to give a nondeterministic algorithm that, given an NFA $A$ as input:

- does not terminate if $A$ is universal,
- has at least one terminating execution answering „non-universal“ if $A$ is not universal, and
- only uses polynomial memory in the size of the input.

The algorithm guesses a word letter by letter, simulating the run of the equivalent DFA on it, and stops if at some point the state of the DFA is non-final.
Universality is PSPACE complete

Universality is PSPACE-hard
Universality is PSPACE complete

Universality is PSPACE-hard

By reduction from the acceptance problem for LBA.

• Let $M$ be a LBA, let $x$ be an input for $M$. We construct in polynomial time a NFA $A$ such that

$$M \text{ accepts } x \iff A \text{ is not universal}$$
Universality is PSPACE complete

Universality is PSPACE-hard

By reduction from the acceptance problem for LBA.

• Let $M$ be a LBA, let $x$ be an input for $M$. We construct in polynomial time a NFA $A$ such that

$$M \text{ accepts } x \iff A \text{ is not universal}$$

• Configuration of $M$: sequence of the form $a_1a_2 \cdots a_i \; q \; a_{i+1} \cdots a_n$ where $a_1, a_2, \ldots, a_n \in \Sigma$, $n = |x|$, $q \in Q$. 
Universality is PSPACE complete

Universality is PSPACE-hard

By reduction from the acceptance problem for LBA.

- Let $M$ be a LBA, let $x$ be an input for $M$. We construct in polynomial time a NFA $A$ such that

  $M$ accepts $x$ iff $A$ is not universal

- Configuration of $M$: sequence of the form $a_1a_2\cdots a_i q a_{i+1}\cdots a_n$ where $a_1,a_2,\ldots,a_n \in \Sigma$, $n = |x|$, $q \in Q$.

- Encode the run of $M$ on $x$ as a word $w = c_0 \# c_1 \# \cdots \# c_n$ where each $c_i$ encodes a configuration of $M$ and $c_0$ is the initial configuration for $x$. 

Universality is PSPACE complete

- **Idea**: construct $A$ so that it accepts all words that are **not** the encoding of an accepting run of $M$ on $x$. Then

  - if $M$ accepts $x$ then $A$ accepts all words **but** $w$
    $\Rightarrow A$ is not universal
  - if $M$ rejects $x$ then $A$ accepts all words
    $\Rightarrow A$ is universal
Universality is PSPACE complete

• The run of $M$ on $x$ is the unique word satisfying the following three properties:
  1. $w$ is a sequence of configurations separated by #
  2. $w$ starts with the initial configuration of $M$ on $x$
  3. every configuration in $w$ is followed by the successor configuration of $M$

• Further, the run is accepting iff
  4. $w$ ends with a final configuration of $M$
Universality is PSPACE complete

- We construct NFAs $A_1, \ldots, A_4$ with polynomially many states recognizing
  1. All words that **do not** consist of a sequence of configurations separated by #
  2. All words that **do not** start with the initial configuration of $M$ on $x$
  3. All words in which some configuration **is not** followed by the successor configuration
  4. All words that **do not** end with a final configuration of $M$

- Let $A$ be a NFA recognizing $L(A_1) \cup L(A_2) \cup L(A_3) \cup L(A_4)$
Universality is PSPACE complete

• We construct NFAs $A_1, \ldots, A_4$ with polynomially many states recognizing
  1. All words that do not consist of a sequence of configurations separated by #
Universality is PSPACE complete

• We construct NFAs $A_1, \ldots, A_4$ with polynomially many states recognizing

  2. All words that do not start with the initial configuration of $M$ on $x$
Universality is PSPACE complete

- We construct NFAs $A_1, \ldots, A_4$ with polynomially many states recognizing
  3. All words in which some configuration is not followed by the successor configuration
Universality is PSPACE complete

- We construct NFAs $A_1, \ldots, A_4$ with polynomially many states recognizing
  4. All words that **do not** end with a final configuration of $M$
Deciding universality of NFAs

• Complement and check for emptiness
  – Needs exponential time and space.

• Improvements:
  – Check for emptiness \textbf{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

1. $\emptyset \leftarrow \emptyset$
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 $\emptyset$
7.    for all $a \in \Sigma$ do
8.       if $\mathcal{W} \cup \emptyset$ contains no $Q'' \subseteq \delta(Q', a)$ then add $\delta(Q', a)$ to $\mathcal{W}$
9.    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 $Q$ contains all minimal states of $B$.

Proof: Assume the contrary. Then $B$ has a shortest path $Q_1 \rightarrow Q_2 \rightarrow \ldots \rightarrow Q_n$ such that
- $Q_1 \in Q$ (after termination), and
- $Q_n \notin Q$ and $Q_n$ is minimal.
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 $Q_1 \rightarrow Q_2 \rightarrow \ldots \rightarrow Q_n$ 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 $Q'_2 \subset Q_2$. 
Subsumption test

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 $Q_1 \rightarrow Q_2 \rightarrow \ldots \rightarrow Q_n$ 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 $Q_2' \rightarrow \ldots \rightarrow Q_n'$ with $Q_n' \subseteq Q_n$.

Since $Q_n$ is minimal, $Q_n'$ is minimal (actually $Q_n' = Q_n$).
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 $Q_1 \rightarrow Q_2 \rightarrow \ldots \rightarrow Q_n$ 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 $Q'_2 \subseteq Q_2$.

But then $B$ has a path $Q'_2 \rightarrow \ldots \rightarrow Q'_n$ with $Q'_n \subseteq Q_n$.

Since $Q_n$ is minimal, $Q'_n$ is minimal (actually $Q'_n = Q_n$).

So the path $Q'_2 \rightarrow \ldots \rightarrow Q'_n$ satisfies

- $Q'_2 \in Q$ (after termination), and
- $Q'_n$ is minimal.

contradicting that $Q_1 \rightarrow Q_2 \rightarrow \ldots \rightarrow Q_n$ 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 $L(A) \supseteq L(B)$, where $B$ is the one-state DFA for $\Sigma^*$. 
Deciding inclusion

- Algorithm: use $L(A_1) \subseteq L(A_2)$ iff $L(A_1) \cap \overline{L(A_2)} = \emptyset$
- Concatenate four algorithms:
  1. determinize $A_2$ $\Rightarrow B_1$
  2. complement the result $\Rightarrow B_2$
  3. intersect $B_2$ with $A_1$ $\Rightarrow B_3$
  4. check for emptiness of $B_3$.
- State of $B_3$: pair $(q, Q)$, with $q$ state of $A_1$ and $Q$ (sub)set of states of $A_2$
- Easy optimizations:
  - store only the states of $B_3$, not its transitions;
  - do not fully construct $B_1$, then $B_2$, then $B_3$; instead, construct directly the states of $B_3$;
  - check for emptiness while constructing $B_3$. 
Deciding inclusion

- Further optimization: subsumption test.

Algorithm 18 NFA inclusion check.

\[ \text{InclNFA}(A_1, A_2) \]
\[ \text{Input: } \text{NFAs } A_1 = (Q_1, \Sigma, \delta_1, Q_{01}, F_1), A_2 = (Q_2, \Sigma, \delta_2, Q_{02}, F_2) \]
\[ \text{Output: } \text{true if } \mathcal{L}(A_1) \subseteq \mathcal{L}(A_2), \text{false otherwise} \]

1. \( Q \leftarrow \emptyset; \)
2. \( W \leftarrow \{[q_{01}, Q_{02}] : q_{01} \in Q_{01}\} \)
3. \( \text{while } W \neq \emptyset \text{ do} \)
   4. \( \text{pick } [q_1, Q'_2] \text{ from } W \)
   5. \( \text{if } (q_1 \in F_1) \text{ and } (Q'_2 \cap F_2 = \emptyset) \text{ then return false} \)
   6. \( \text{add } [q_1, Q'_2] \text{ to } Q \)
   7. \( \text{for all } a \in \Sigma \text{ do} \)
      8. \( Q''_2 \leftarrow \bigcup_{q_2 \in Q'_2} \delta_2(q_2, a) \)
      9. \( \text{for all } q'_1 \in \delta_1(q_1, a) \text{ do} \)
         10. \( \text{if } W \cup Q \text{ contains no } [q''_1, Q''''_2] \text{ s.t. } q''_1 = q'_1 \text{ and } Q''''_2 \subseteq Q''_2 \text{ then} \)
             11. \( \text{add } [q'_1, Q''_2] \text{ to } W \)
   12. \( \text{return true} \)
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 outer for-loop is executed $k$ times.
    • Line 8 takes $O(n_2^2)$ time.
    • The inner for-loop is executed at most $n_1$ times.
    • Line 19 (without subsumption!) takes constant 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 outer for-loop is executed $k$ times.
  - Line 8 takes constant time
  - The inner for-loop is executed $O(n_1)$ times
  - Line 10 (without subsumption) takes constant time
  - Overall: $O(k \cdot n_1^2 \cdot n_2)$ time.

- Checking equality: check inclusion in both directions.