Text Search

- Nondeterministic Finite Automata
- Transformation NFA to DFA and Simulation of NFA
- Text Search Using Automata
- Power of Nondeterministic Approach
- Regular Expression Search
- Dealing with ε-transitions
Languages, grammars, automata

Czech instant sources:
[1] M. Demlová: **A4B01JAG**
http://math.feld.cvut.cz/demlova/teaching/jag/
  Pages 1-27, in PAL, you may wish to skip: Proofs, chapters 2.4, 2.6, 2.8.

http://is.muni.cz/do/1499/el/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf
  Chapters 1 and 2, skip same parts as in [1].

English sources:
  Chapters 1.4 and 1.5, it is probably reasonably short, there is nothing to skip.

follow the link at http://cw.felk.cvut.cz/doku.php/courses/a4m33pal/literatura_odkazy
  Chapters 1., 2., 3., there is a lot to skip, consult the teacher preferably.

For more references see PAL links pages
http://cw.felk.cvut.cz/doku.php/courses/b4m33pal/odkazy-zdroje (CZ)
https://cw.fel.cvut.cz/wiki/courses/be4m33pal/references (EN)
Deterministic Finite Automaton (DFA)
Nondeterministic Finite Automaton (NFA)

Both DFA and NFA consist of:
- Finite input alphabet $\Sigma$.
- Finite set of internal states $Q$.
- One start state $q_0 \in Q$.
- Nonempty set of accept states $F \subseteq Q$.
- Transition function $\delta$.

DFA transition function is $\delta : Q \times \Sigma \rightarrow Q$.
DFA is always in one of its states $q \in Q$.
DFA transits from current state to another state depending on the current input symbol.

NFA transition function is $\delta : Q \times \Sigma \rightarrow P(Q)$  ($P(Q)$ is the powerset of $Q$)
NFA is always (simultaneously) in a set of some number of its states.
NFA transits from set of current states to another set of states depending on the current input symbol.
NFA $A_1$, its transition diagram and its transition table

<table>
<thead>
<tr>
<th>states</th>
<th>alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2</td>
</tr>
<tr>
<td>1</td>
<td>3,4</td>
</tr>
<tr>
<td>2</td>
<td>4,5</td>
</tr>
<tr>
<td>3</td>
<td>6 0</td>
</tr>
<tr>
<td>4</td>
<td>6,7,8</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>6 6</td>
</tr>
<tr>
<td>8</td>
<td>7 7</td>
</tr>
</tbody>
</table>

accept states marked
Indeterminism

NFA $A_1$ processing input word $abcba$

1. $A_1$

2. $abcba$

3. Active states

4. $abckba$

continue...
NFA $A_1$ has processed the word $abcba$ and went through the input characters and respective sets(!) of states:

$\{0\} \rightarrow a \rightarrow \{1\} \rightarrow b \rightarrow \{3, 4\} \rightarrow c \rightarrow \{0, 6, 7, 8\} \rightarrow b \rightarrow \{2, 6, 7\} \rightarrow a \rightarrow \{0, 4, 5, 6\}$.
Indeterminism

NFA simulation without transform to DFA

Each of the current states is occupied by one token. Read an input symbol and move the tokens accordingly. If a token has more movement possibilities it will split into two or more tokens, if it has no movement possibility it will leave the board, uhm, the transition diagram.

Read b from input
NFA simulation without transform to DFA

Idea:
Register all states to which you have just arrived. In the next step, read the input symbol \( x \) and move SIMULTANEOUSLY to ALL states to which you can get from ALL active states along transitions marked by \( x \).

Input: NFA, text in array \( t \)

```java
SetOfStates S = {q0}, S_tmp;

i = 1;
while( (i <= t.length) && (!S.isEmpty()) ) {
    S_tmp = Set.emptySet();
    for( q in S ) // for each state in S
        S_tmp.union( delta(q, t[i]) );
    S = S_tmp;
i++;
}
return S.containsFinalState(); // true or false
```
Generating DFA $A_2$ equivalent to NFA $A_1$ using transition tables

**Data**
Each state of DFA is a subset of states of NFA.
Start state of DFA is an one-element set containing the start state of NFA.
A state of DFA is an accept state iff it contains at least one accept state of NFA.

**Construction**
Create the start state of DFA and the corresponding first line of its transition table (TT).

For each state $Q$ of DFA not yet processed do {
    Decompose $Q$ into its constituent states $Q_1,\ldots, Q_k$ of NFA
    For each symbol $x$ of alphabet do {
        $S =$ union of all references in NFA table at positions $[Q_1][x],\ldots,[Q_k][x]$
        if (S is not among states of DFA yet)
            add S to the states of DFA and add a corresponding line to TT of DFA
        \[TT[Q][x] := S\]
    } // for each symbol
    Mark $Q$ as processed
} // for each state

// Remember, empty set is also a set of states, it can be a state of DFA
Generating DFA $A_2$ equivalent to NFA $A_1$

### Example 8

#### NFA to DFA

**Copy start state**

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6,7,8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Add new state(s)**

- **State 1**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 2**
  - Transition table:
    - On input $a$: State 5, 8
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 3**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 4**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 5**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 6**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 7**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

- **State 8**
  - Transition table:
    - On input $a$: State 3, 4, 6
    - On input $b$: State 2, 4, 7
    - On input $c$: State 3, 4, 6

**Conclusion**

The DFA $A_2$ is equivalent to the NFA $A_1$. The transitions and states have been accurately mapped to ensure the correct behavior of the automaton.

continue...
Generating DFA $A_2$ equivalent to NFA $A_1$

$$
\begin{array}{|c|c|c|c|}
\hline
\text{State} & a & b & c \\
\hline
0 & 1 & 2 & \text{F} \\
1 & \text{F} & \text{F} & \text{F} \\
2 & 4,5 & 6,7,8 & \text{F} \\
3 & 6 & 0 & \text{F} \\
4 & 6 & 6 & \text{F} \\
5 & 7 & 7 & \text{F} \\
\hline
\end{array}
$$

$A_2$

Add new state(s)

$$
\begin{array}{|c|c|c|}
\hline
\text{State} & a & b & c \\
\hline
0 & 1 & 2 & \text{F} \\
1 & 34 & \text{F} & \text{F} \\
2 & \text{F} & \text{F} & \text{F} \\
\hline
\end{array}
$$

$A_1$

Continue...
Generating DFA $A_2$ equivalent to NFA $A_1$

### $A_1$

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3,4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6,7,8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

### $A_2$

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>

Add new state(s)
Generating DFA $A_2$ equivalent to NFA $A_1$

\[
\begin{array}{c|ccc}
   & a & b & c \\
\hline
0 & 1 & 2 & \\
1 & 3,4 & \\
2 & 4,5 & \\
3 & 6 & 0 & \\
4 & 6 & 6 & 6,7,8 \\
5 & 8 & & \\
6 & 0 & & \\
7 & 6 & 6 & \\
8 & 7 & 7 & \\
\end{array}
\]

$A_2$

Add new state(s)

\[
\begin{array}{c|ccc}
   & a & b & c \\
\hline
0 & 1 & 2 & \\
1 & 34 & \\
2 & 45 & \\
3 & 6 & 0678 & \\
4 & & & \\
\end{array}
\]

$A_1$
Generating DFA $A_2$ equivalent to NFA $A_1$

### Example 12

Generating DFA $A_2$ equivalent to NFA $A_1$

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3,4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6,7,8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

### Add new state(s)

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>678</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>678</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>678</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### $A_1$

### $A_2$
Generating DFA $A_2$ equivalent to NFA $A_1$

### $A_1$

- States: 0, 1, 2, 3, 4, 5, 6, 7, 8
- Transitions:
  - $0 \rightarrow 1, 2, c$
  - $1 \rightarrow 3, 4$
  - $2 \rightarrow 4, 5$
  - $3 \rightarrow 6$
  - $4 \rightarrow 7, 8$
  - $5 \rightarrow 6$
  - $6 \rightarrow 7$
  - $7 \rightarrow 8$

### $A_2$

- States: 0, 1, 2, 3, 4, 5, 6, 8
- Transitions:
  - $0 \rightarrow 1, 2$
  - $1 \rightarrow 34$
  - $2 \rightarrow 45$
  - $34 \rightarrow 6$
  - $45 \rightarrow 8, 678$

### Add new state(s)

- State 6
  - $0 \rightarrow 0678$
  - $6 \rightarrow 8$
  - $8 \rightarrow 678$

### No new state

- $A_1$ and $A_2$ have the same behavior.
### Generating DFA $A_2$ equivalent to NFA $A_1$

#### Example 14

#### Add new state(s)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3,4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6,7,8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**$A_2$**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>6</td>
<td>0678</td>
<td>F</td>
</tr>
<tr>
<td>45</td>
<td>8</td>
<td>678</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0678</td>
<td></td>
<td>0167</td>
<td>267</td>
</tr>
</tbody>
</table>

### Diagram of $A_1$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>c</td>
<td>a,b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>b</td>
<td>a,b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>a</td>
<td>a,b</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>c</td>
<td>a,b</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>a</td>
<td>a,b</td>
</tr>
<tr>
<td>7</td>
<td>b</td>
<td>c</td>
<td>a,b</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

continue...
Generating DFA $A_2$ equivalent to NFA $A_1$

$A_1$

$A_2$

Add new state(s)

Continue...
Generating DFA $A_2$ equivalent to NFA $A_1$

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3,4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6,7,8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>8</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Add new state(s)

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>34</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0678</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>678</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0167</td>
<td>267</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>678</td>
<td>067</td>
<td>67</td>
</tr>
</tbody>
</table>

NFA to DFA

Example 16

Add new state(s)

continue...
DFA $A_2$ equivalent to NFA $A_1$

\[ A_2 \]

\[
\begin{array}{|c|c|c|c|}
\hline
 & a & b & c \\
\hline
0 & 1 & 2 & F \\
1 &  & 3,4 & \\
2 & 4,5 &  & \\
3 & 6 & 0 & \\
4 &  & 6,7,8 & F \\
5 & 8 &  & F \\
6 & 0 &  & \\
7 & 6 & 6 & \\
8 & 7 & 7 & \\
\hline
\end{array}
\]

Example 17

...FINISHED!
Naïve approach
1. Align the pattern with the beginning of the text.
2. While corresponding symbols of the pattern and the text match each other move forward by one symbol in the pattern.
3. When symbol mismatch occurs shift the pattern forward by one symbol, reset position in the pattern to the beginning of the pattern and go to 2.
4. When the end of the pattern is passed report success, shift the pattern forward by one symbol, reset position in the pattern to its beginning and go to 2.
5. When the end of the text is reached stop.

<table>
<thead>
<tr>
<th>Start</th>
<th>Pattern shift</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>text</strong></td>
<td>a b c a b c a b c ...</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>a b c x</td>
</tr>
<tr>
<td><strong>text</strong></td>
<td>a b c a b c a b c ...</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>a b c x</td>
</tr>
</tbody>
</table>

**after a while:**

| text | a b c a b c a b c ... |
| pattern | a b c x |

<table>
<thead>
<tr>
<th>etc...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>text</strong></td>
</tr>
<tr>
<td><strong>pattern</strong></td>
</tr>
</tbody>
</table>

To be used with great caution! Might be efficient in a favourable text
**Alphabet:** Finite set of symbols.
**Text:** Sequence of symbols of the alphabet.
**Pattern:** Sequence of symbols of the same alphabet.
**Goal:** Pattern occurrence is to be detected in the text.

Text is often fixed or seldom changed, pattern typically varies (looking for different words in the same document), pattern is often significantly shorter than the text.

**Notation**
- Alphabet: $\Sigma$
- Symbols in the text: $t_1, t_2, \ldots, t_n$.
- Symbols in the pattern: $p_1, p_2, \ldots, p_m$.
- It holds $m \leq n$, usually $m \ll n$.

**Example**
- **Text:** ...task is very simple but it is used very freq...
- **Pattern:** simple
NFA $A_3$ which accepts just a single word $p_1p_2p_3p_4$.

NFA $A_4$ which accepts each word with suffix $p_1p_2p_3p_4$ and its transition table.

$z \in \Sigma - \{p_1, p_2, p_3, p_4\}$

*all other characters in the alphabet, each over its own column*
NFA A₄ which accepts each word with suffix $p₁ p₂ p₃ p₄$ and its transition table.

$z \in \Sigma - \{p₁, p₂, p₃, p₄\}$

DFA A₅ is a deterministic equivalent of NFA A₄.

Supposing $p₁ p₂ p₃ p₄$ are mutually different!
NFA $A_6$ which accepts each word with suffix $abba$ and its transition table:

- $a$ transitions:
  - from state 0 to 1
- $b$ transitions:
  - from state 0 to 1
  - from state 1 to 2
- $z$ transitions:
  - from state 1 to 2
  - from state 2 to 3
  - from state 3 to 4

DFA $A_7$ is a deterministic equivalent of NFA $A_6$. It also accepts each word with suffix $abba$.

Note the structural difference between $A_5$ and $A_7$. 
NFA accepting exactly one word $p_1p_2p_3p_4$.

NFA accepting any word with suffix $p_1p_2p_3p_4$.

NFA accepting any word with substring (factor) $p_1p_2p_3p_4$ anywhere in it.
NFA accepting any word with substring (factor) $p_1 p_2 p_3 p_4$ anywhere in it.

Can be used for searching, but the following reduction is more frequent.

Text search NFA for finding pattern $P = p_1 p_2 p_3 p_4$ in the text.

Whenever at least one of the active states is final, report pattern detection in the text.

Note that multiple occurrences of the pattern in the text, even overlapping ones, are all detected, due to indeterminism and Sigma-loop in the start state.
Text search automaton may be defined as a 6-tuple consisting of

1. Finite input alphabet $\Sigma$.
2. Finite set of internal states $Q$.
3. One start state $q_0 \in Q$.
4. Nonempty set of accept states $F \subseteq Q$.
5. Transition function $\delta : Q \times \Sigma \rightarrow P(Q)$ ($P(Q)$ is the powerset of $Q$)
6. Detection function $\chi : P(Q) \rightarrow \{0, 1\}$

With the properties:

A. $\forall \alpha \in \Sigma : q_0 \in \delta(q_0, \alpha)$ (loop in the start state labeled by the whole alphabet)
B. $\forall S \subseteq Q : \chi(S) = 1 \iff S \cap F \neq \emptyset$ (a string in text is detected when $\chi(\{\text{active states}\}) = 1$)

Notation and terminology note:

Formally speaking, a text search automaton is clearly neither DFA nor NFA. However, when component 6. and property B are removed from the definition the result is a NFA. This NFA is specified unambiguously by the text search automaton.

Usually, when speaking about "text search NFA x" we keep in mind a "text search automaton y", from which x is obtained by removing component 6 and property B. The rest of the slides follows this slightly inexact terminology, as do also many other sources.
NFA accepting any word with subsequence $p_1p_2p_3p_4$ anywhere in it.

Alternatively: NFA accepting any word containing a subsequence $Q$ whose Hamming distance from $p_1p_2p_3p_4$ is at most 1.
Search NFA can search for more than one pattern simultaneously. The number of patterns can be

- **finite** -- this leads also to a dictionary automaton (we will meet it later)
- **or infinite** -- this leads to a regular language.

**Chomsky language hierarchy remainder**

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Language</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-0</td>
<td>Recursively enumerable</td>
<td>Turing machine</td>
</tr>
<tr>
<td>Type-1</td>
<td>Context-sensitive</td>
<td>Linear-bounded non-deterministic Turing machine</td>
</tr>
<tr>
<td>Type-2</td>
<td>Context-free</td>
<td>Non-deterministic pushdown automaton</td>
</tr>
<tr>
<td>Type-3</td>
<td>Regular</td>
<td>Finite state automaton (NFA or DFA)</td>
</tr>
</tbody>
</table>

Only regular languages can be processed by NFA/DFA. More complex languages cannot. For example, any language containing *well-formed parentheses* is context-free and not regular and cannot be recognized by NFA/DFA.
Let $L_1$ and $L_2$ be any languages. Then

- $L_1 \cup L_2$ is **union** of $L_1$ and $L_2$. It is a set of all words which are in $L_1$ or in $L_2$.
- $L_1.L_2$ is **concatenation** of $L_1$ and $L_2$. It is a set of all words $w$ for which holds $w = w_1w_2$ (concatenation of words $w_1$ and $w_2$), where $w_1 \in L_1$ and $w_2 \in L_2$.
- $L_1^*$ is **Kleene star** or Kleene closure of language $L_1$. It is a set of all words which are concatenations of any number (incl. zero) of any words of $L_1$ in any order.

**Closure property**
Whenever $L_1$ and $L_2$ are regular languages then $L_1 \cup L_2$, $L_1.L_2$, $L_1^*$ are regular languages too.

**Example**
$L_1 = \{001, 0001, 00001, \ldots\}$, $L_2 = \{110, 1110, 11110, \ldots\}$.

$L_1 \cup L_2 = \{001, 110, 0001, 1110, 0001, 1110, \ldots\}$
$L_1.L_2 = \{001110, 0011110, 00111110, \ldots, 0001110, 00011110, 000111110, \ldots\}$
$L_1^* = \{\varepsilon, 001, 001001, 001001001, 001001001001, 001001001001001, \ldots\}$ // this one is not easy to list nicely... or is it?
Regular expressions defined recursively
Symbol $\varepsilon$ is a regular expression.
Each symbol of alphabet $\Sigma$ is a regular expression.
Whenever $e_1$ and $e_2$ are regular expressions then also strings
$(e_1), e_1 + e_2, e_1 e_2, (e_1)^*$ are regular expressions.

Languages represented by regular expressions (RE) defined recursively
RE $\varepsilon$ represents language containing only empty string.
RE $x$, where $x \in \Sigma$, represents language \{x\}.
Let RE's $e_1$ and $e_2$ represent languages $L_1$ and $L_2$. Then
RE $(e_1)$ represents $L_1$, \hspace{1cm} RE $e_1 + e_2$ represents $L_1 \cup L_2$,\nREs $e_1 e_2, e_1 . e_2$ represent $L_1 L_2$, \hspace{1cm} RE $(e_1)^*$ represents $L_1^*$.

Examples
$0+1(0+1)^*$ all integers in binary without leading 0's
$0.(0+1)^*1$ all finite binary fractions $\in (0, 1)$ without trailing 0's
$((0+1)(0+1+2+3+4+5+6+7+8+9) + 2(0+1+2+3)): (0+1+2+3+4+5)(0+1+2+3+4+5+6+7+8+9)$
all 1440 day's times in format hh:mm from 00:00 to 23:59
$(\text{mon+}(\text{wedne+t(ue+ hur))s+fri+s(atur+un))d ay}$
English names of days in the week
$(1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)^*((2+7)5+(5+0)0)$
all decimal integers $\geq 100$ divisible by 25
Convert regular expression to NFA

Input: Regular expression R containing \( n \) characters of the given alphabet.
Output: NFA recognizing language \( L(R) \) described by R.

Create start state \( S \)

for each \( k \ (1 \leq k \leq n) \) {
    assign index \( k \) to the \( k \)-th character in R
    // this makes all characters in R unique: \( c[1], c[2], ..., c[n] \).
    create state \( S[k] \)       // \( S[k] \) corresponds directly to \( c[k] \)
}

for each \( k \ (1 \leq k \leq n) \) {
    if \( c[k] \) can be the first character in some string described by R
        then create transition \( S \to S[k] \) labeled by \( c[k] \) with index stripped off
    if \( c[k] \) can be the last character in some string described by R
        then mark \( S[k] \) as final state
    for each \( p \ (1 \leq p \leq n) \)
        if (\( c[k] \) can follow immediately after \( c[p] \) in some string described by R)
            then create transition \( S[p] \to S[k] \) labeled by \( c[k] \) with index stripped off
}
Regular expression

\[ R = a^*b(c + a^*b)^*b + c \]

Add indices:

\[ R = a_1^*b_2(c_3 + a_4^*b_5)^*b_6 + c_7 \]

NFA accepts \( L(R) \)
NFA searches the text for any occurrence of any word of $L(R)$

$$R = a^*b \ (c + a^*b)^* \ b + c$$
To find a subsequence representing a word $\in L(R)$, where $R$ is a regular expression, do the following:

Create NFA accepting $L(R)$
Add self loops to the states of NFA:
1. Self loop labeled by $\Sigma$ (whole alphabet) at the start state.
2. Self loop labeled $\Sigma - \{x\}$ at each state whose outgoing transition(s) are labeled by single $x \in \Sigma$. // serves as an "optimized" wait loop
3. Self loop labeled by $\Sigma$ at each state whose outgoing transition(s) are labeled by more than single symbol from $\Sigma$. // serves as an "usual" wait loop
4. No self loop to all other states. // which have no outgoing loop = final ones
NFA searches the text for any occurrence of any subsequence representing a word of \( L(R) \)

\[ R = ab + (abcb + cc)^* \ a \]
Transforming NFA which searches text for an occurrence of a word of a given regular language into the equivalent DFA might take exponential space and thus also exponential time. Not always, but sometimes yes:

Consider regular expression $R = a(a+b)(a+b)\ldots(a+b)$ over alphabet \{a, b\}.

Text search NFA1 for $R$

Text search NFA2 for $R$, why not this one?
R = a(a+b)(a+b)

### NFA table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### DFA table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>012</td>
<td>02</td>
</tr>
<tr>
<td>012</td>
<td>0123</td>
<td>023</td>
</tr>
<tr>
<td>0123</td>
<td>0123</td>
<td>023</td>
</tr>
<tr>
<td>02</td>
<td>013</td>
<td>03</td>
</tr>
<tr>
<td>023</td>
<td>013</td>
<td>03</td>
</tr>
<tr>
<td>013</td>
<td>012</td>
<td>02</td>
</tr>
<tr>
<td>03</td>
<td>01</td>
<td>0</td>
</tr>
</tbody>
</table>
NFA with $\varepsilon$-transitions
The transition from one state to another can be performed \textbf{without} reading any input symbol. Such transition is labeled by symbol $\varepsilon$.

$\varepsilon$-closure
Symbol $\varepsilon$-CLOSURE($p$) denotes the union of $\{p\}$ and the set of all states $q$, which can be reached from state $p$ using only $\varepsilon$-transitions.

By definition, $\varepsilon$-CLOSURE($p$) = $\{p\}$ when there is no $\varepsilon$-transition out from $p$.

$\varepsilon$-CLOSURE(0) = \{0, 2, 3, 4\}
$\varepsilon$-CLOSURE(1) = \{1\}
$\varepsilon$-CLOSURE(2) = \{2, 3, 4\}
$\varepsilon$-CLOSURE(3) = \{3\}
...
Construction of equivalent NFA without $\varepsilon$-transitions

Input: NFA $A$ with some $\varepsilon$–transitions.
Output: NFA $A'$ without $\varepsilon$–transitions.

1. $A' = \text{exact copy of } A$.
2. Remove all $\varepsilon$–transitions from $A'$.
3. In $A'$ for each $(q, a)$ do:
   add to the set $\delta(p, a)$ all such states $r$ for which it holds 
   $q \in \varepsilon$–CLOSURE($p$) and $\delta(q, a) = r$.
4. Add to the set of final states $F$ in $A'$ all states $p$ for which it holds 
   $\varepsilon$–CLOSURE($p$) $\cap F \neq \emptyset$.
Epsilon Transitions

NFA with $\varepsilon$-transitions

Equivalent NFA without $\varepsilon$-transitions

New transitions and accept states are highlighted
NFA for search for any unempty substring of pattern $p_1p_2p_3p_4$ over alphabet $\Sigma$. Note the $\varepsilon$-transitions.

**Epsilon Transitions**

**Application 40**

**Powerful trick!**

**Union** of two or more NFA:
Create additional start state $S$ and add $\varepsilon$-transitions from $S$ to the start states of all involved NFA's. Draw an example yourself!
Equivalent NFA for search for any unempty substring of pattern $p_1p_2p_3p_4$ with $\varepsilon$-transitions removed.

States 5, 9, 12 are unreachable. Transformation algorithm NFA -> DFA if applied, will neglect them.
### Transition table of NFA $A_{11}$ (without $\varepsilon$-transitions).

<table>
<thead>
<tr>
<th>States (p)</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>10</td>
<td>13</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>0.13</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>7</td>
<td>0.26</td>
<td>0.1</td>
<td>0.6</td>
<td>0.3.7.10</td>
<td>0.13</td>
</tr>
<tr>
<td>8</td>
<td>0.7.10</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.8.11.13</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>0.11.13</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
</tr>
<tr>
<td>10</td>
<td>0.3.7.10</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.4.8.11.13</td>
</tr>
<tr>
<td>11</td>
<td>0.8.11.13</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>12</td>
<td>0.4.8.11.13</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
</tbody>
</table>

### Transition table of DFA which is equivalent to $A_{11}$.

<table>
<thead>
<tr>
<th>States (p)</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>10</td>
<td>13</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>0.13</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>7</td>
<td>0.26</td>
<td>0.1</td>
<td>0.6</td>
<td>0.3.7.10</td>
<td>0.13</td>
</tr>
<tr>
<td>8</td>
<td>0.7.10</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.8.11.13</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>0.11.13</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
</tr>
<tr>
<td>10</td>
<td>0.3.7.10</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.4.8.11.13</td>
</tr>
<tr>
<td>11</td>
<td>0.8.11.13</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>12</td>
<td>0.4.8.11.13</td>
<td>0.1</td>
<td>0.6</td>
<td>0.10</td>
<td>0.13</td>
</tr>
</tbody>
</table>

### Question

DFA in this case has less states than the equivalent NFA.
Q: Does it hold for any automaton of this type? Proof?
Input: NFA, text in array t

```plaintext
SetOfStates S = eps_CLOSURE(q0), S_tmp;
int i = 1;
while ((i <= t.length) && (!S.empty())) {
    for (q in S)
        if (q.isFinal) // if( chi(S) == 1 )
            print(q.final_state_info); // pattern found

    S_tmp = Set.empty(); // transiton to next
    for (q in S) // set of states
        S_tmp.union(eps_CLOSURE(delta(q, t[i])));
    S = S_tmp;
    i++; // next char in text
}
return S.containsFinalState(); // true or false
```