Non-deterministic Finite Automata

H. Geuvers and T. van Laarhoven

Institute for Computing and Information Sciences – Intelligent Systems
Radboud University Nijmegen

Version: fall 2014
Outline

Non-deterministic Finite Automata

Eliminating non-determinism
Regular Expressions and Regular Languages

\[ \text{rexp}_\Sigma ::= \emptyset \mid \lambda \mid s \mid \text{rexp}_\Sigma \text{rexp}_\Sigma \mid \text{rexp}_\Sigma + \text{rexp}_\Sigma \mid \text{rexp}_\Sigma^* \]

with \( s \in \Sigma \)

\( L \subseteq \Sigma^* \) is regular if \( L = \mathcal{L}(e) \) for some regular expression \( e \).

Deterministic Finite Automata, DFA

**Proposition** Closure under complement, union, intersection

If \( L_1, L_2 \) are accepted by some DFA, then so are

- \( \overline{L_1} = \Sigma^* - L_1 \)
- \( L_1 \cup L_2 \)
- \( L_1 \cap L_2 \).
Theorem
The languages accepted by DFAs are exactly the regular languages
We prove this by
1. If \( L = \mathcal{L}(M) \) for some DFA \( M \), then there is a regular expression \( e \) such that \( L = \mathcal{L}(e) \) (Previous lecture)
2. If \( L = \mathcal{L}(e) \), for some regular expression \( e \), then there is a non-deterministic finite automaton with \( \lambda \)-steps (NFA\(_\lambda\)) \( M \) such that \( L = \mathcal{L}(M) \). (This lecture)
3. For every NFA\(_\lambda\), \( M \), there is a DFA \( M' \) such that \( \mathcal{L}(M) = \mathcal{L}(M') \) (This lecture)
Non-deterministic finite automaton (NFA)

$\delta(q, a)$ is not one state, but a set of states.

\[
\begin{array}{c|cc}
\delta & a & b \\
\hline
q_0 & \{q_0\} & \{q_0, q_1\} \\
q_1 & \emptyset & \{q_2\} \\
q_2 & \emptyset & \emptyset
\end{array}
\]

in shorthand

\[
\begin{array}{c|cc}
\delta & a & b \\
\hline
q_0 & q_0 & q_0, q_1 \\
q_1 & q_0 & q_2 \\
q_2 & & q_2
\end{array}
\]
Non-deterministic Finite Automata: NFA (formally)

$M$ is a NFA over $\Sigma$ if $M = (Q, q_0, \delta, F)$ with

- $Q$ is a finite set of states
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a finite set of final states
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}Q$ is the transition function

$\mathcal{P}Q$ denotes the collection of subsets of $Q$.

Reading function $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}Q$ (multi-step transition)

- $\delta^*(q, \lambda) = \{q\}$
- $\delta^*(q, aw) = \{q' \mid q' \in \delta^*(p, w) \text{ for some } p \in \delta(q, a)\}$
- $= \bigcup_{p \in \delta(q, a)} \delta^*(p, w)$

$[\bigcup X_i \text{ denotes the union of all the } X_i]$.

The language accepted by $M$, notation $\mathcal{L}(M)$, is:

$\mathcal{L}(M) = \{w \in \Sigma^* \mid \exists q_f \in F(q_f \in \delta^*(q_0, w))\}$
For the union of languages we can put NFAs in parallel

Example. Suppose we want to have an NFA for \( L_1 \cup L_2 = \{ w \mid |w|_a \text{ is even or } |w|_b \geq 1 \} \)

First idea: put the two machines “non-deterministically” in parallel

But this is **wrong**: The NFA accepts \( aaa \).
We add $\lambda$-transitions or ‘silent steps’ to NFAs.

The correct union of $M_1$ and $M_2$ is:

In an NFA$_\lambda$ we allow

$$\delta(q, \lambda) = q'$$

for $q \neq q'$. That means

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}Q$$
NFA$\lambda$ formally

$M$ is an NFA$\lambda$ over $\Sigma$ if $M = (Q, q_0, \delta, F)$ with

- $Q$ is a finite set of states
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a finite set of final states
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P} Q$ is the transition function

The $\lambda$-closure of a state $q$, $\lambda$-closure($q$), is the set of states reachable with only $\lambda$-steps.

Reading function $\delta^* : Q \times \Sigma^* \to \mathcal{P} Q$ (multi-step transition)

\[
\delta^*(q, \lambda) = \lambda\text{-closure}(q)
\]

\[
\delta^*(q, aw) = \{ q' \mid \exists p \in \lambda\text{-closure}(q) \exists r \in \delta(p, a) (q' \in \delta^*(r, w)) \}
\]

\[
= \bigcup_{p \in \lambda\text{-closure}(q)} \bigcup_{r \in \delta(p, a)} \delta^*(r, w)
\]

The language accepted by $M$, notation $\mathcal{L}(M)$, is:

\[
\mathcal{L}(M) = \{ w \in \Sigma^* \mid \exists q_f \in F (q_f \in \delta^*(q_0, w)) \}
\]
Insulated machines

A finite automaton $M$ is called **insulated** if

- $q_0$ has no in-going arrows
- there is only one final state which has no out-going arrows

**Proposition.** For any machine $M$ one can find an insulated NFA $M'$ such that $M'$ accepts the same language

**Proof.** By adding states and silent steps, for example

![Diagram showing an insulated automaton](image)

gives

![Diagram showing a transformed automaton](image)
For each regular expression, we construct an insulated NFA_{\lambda}.

<table>
<thead>
<tr>
<th>e</th>
<th>M such that ( \mathcal{L}(M) = \mathcal{L}(e) )</th>
</tr>
</thead>
</table>
| \( \emptyset \) | \begin{align*} &\text{start} \\
                   &\overset{}{\longrightarrow} q_0 \end{align*} |
| \( \lambda \)    | \begin{align*} &\text{start} \\
                   &\overset{}{\longrightarrow} S \end{align*} |
| \( a \) (for \( a \in \Sigma \)) | \begin{align*} &\text{start} \\
                   &\overset{}{\longrightarrow} S \end{align*} \begin{align*} &\overset{a}{\longrightarrow} F \end{align*} |

\[ e = e_1 + e_2 \]
with
\[ \mathcal{L}(M_1) = \mathcal{L}(e_1) \]
\[ \mathcal{L}(M_2) = \mathcal{L}(e_2) \]
<table>
<thead>
<tr>
<th>$e$</th>
<th>$M$ such that $\mathcal{L}(M) = \mathcal{L}(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e = e_1 e_2$ with $\mathcal{L}(M_1) = \mathcal{L}(e_1)$ and $\mathcal{L}(M_2) = \mathcal{L}(e_2)$</td>
<td>![Diagram 1]</td>
</tr>
<tr>
<td>$e = (e_1)^*$ with $\mathcal{L}(M_1) = \mathcal{L}(e_1)$</td>
<td>![Diagram 2]</td>
</tr>
</tbody>
</table>
Proposition. For every regular expression $e$ there is an NFA$_\lambda$ $M_e$ such that

$$L(M_e) = L(e).$$

Proof. Apply the toolkit. $M_e$ can be found by induction on the structure of $e$: First do this for the simplest regular expressions. For a composed regular expression compose the automata.

Corollary. For every regular language $L$ there is an NFA$_\lambda$ $M$ that accepts $L$ (so $L(M) = L$).
Avoiding non-determinism

We can transform any NFA (and NFA_λ) into a DFA that accepts the same language.

Idea:

- Keep track of all the states you can go to!
- A combination of states is final if one of the members is final.

Example: \( L = \{ w \mid |w|_a \text{ is even or } |w|_b \geq 1 \} \)
Eliminating non-determinism and $\lambda$-steps

Let $M$ be a NFA given by $(Q, q_0, \delta, F)$.
Define the DFA $M^+$ as $(Q^+, q_0^+, \delta^+, F^+)$ where

\[
Q^+ = \mathcal{P}Q
\]
\[
q_0 = \{q_0\}
\]
\[
\delta^+(H, a) = \bigcup_{q \in H} \delta(q, a), \quad \text{for } H \subseteq Q,
\]
\[
F^+ = \{ H \subseteq Q \mid H \cap F \neq \emptyset \}
\]

Then $M^+$ is a DFA accepting the same language as $M$.

If $M$ is an NFA$_\lambda$, we take

\[
\delta^+(H, a) = \bigcup_{q \in H} \bigcup_{p \in \lambda\text{-closure}(q)} \lambda\text{-closure}(\delta(p, a))
\]
\[
F^+ = \{ H \subseteq Q \mid \lambda\text{-closure}(H) \cap F \neq \emptyset \}
\]
Equivalence of DFA, NFA and NFA_λ

**Conclusion.** Every NFA_λ (or NFA) $M$ can be turned into a DFA $M'$ accepting the same language.

**Corollary.** For every regular language $L$ there is a DFA $M$ that accepts $L$ (so $\mathcal{L}(M) = L$).

**Proof.** Given a regular expression $e$, first construct an NFA_λ $M$ such that $\mathcal{L}(M) = \mathcal{L}(e)$. Then change it into a DFA preserving the language that is accepted.

Rephrasing of Kleene’s Theorem:
The class of regular languages is (equivalently) characterized as

1. The languages described by a regular expression
2. The languages accepted by a DFA
3. The languages accepted by an NFA
4. The languages accepted by a NFA_λ