Non-deterministic Finite Automata

H. Geuvers and A. Kissinger

Institute for Computing and Information Sciences
Radboud University Nijmegen

Version: fall 2015
Outline

Non-deterministic Finite Automata

From Regular Expressions to NFA-λ

Eliminating non-determinism
Regular Expressions and Regular Languages

$$\text{rexp}_\Sigma ::= 0 \mid 1 \mid s \mid \text{rexp}_\Sigma \text{rexp}_\Sigma \mid \text{rexp}_\Sigma + \text{rexp}_\Sigma \mid \text{rexp}_\Sigma^*$$

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

- $\overline{L_1} = \Sigma^* - L_1$
- $L_1 \cup L_2$
- $L_1 \cap L_2$. 
Kleene’s Theorem (announced last lecture)

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)

\[ \delta(q, a) \text{ is not one state, but a set of states.} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( { q_0 } )</td>
<td>( { q_0, q_1 } )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( \emptyset )</td>
<td>( { q_2 } )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

in shorthand

\[ \delta \begin{array}{c|c|c}
q_0 & q_0, q_1 \\
q_1 & q_2 \\
q_2 & \\
\end{array} \]

H. Geuvers & A. Kissinger
Version: fall 2015
Non-deterministic Finite Automata: NFA (definition)

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

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

\([P Q\) denotes the \textbf{collection of subsets of} \(Q\)]

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

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

The \textbf{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

\[
d(q, \lambda) = q'
\]

for \( q \neq q' \). That means

\[
d : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}Q
\]
NFA-\(\lambda\) (definition)

\(M\) 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
- \(\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}Q\) 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 \(\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}Q\) (multi-step transition)

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

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 each regular expression, we construct an NFA-$\lambda$.

<table>
<thead>
<tr>
<th>$e$</th>
<th>$M$ such that $\mathcal{L}(M) = \mathcal{L}(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\begin{array}{c} \text{start} \longrightarrow q_0 \end{array}$</td>
</tr>
<tr>
<td>1</td>
<td>$\begin{array}{c} \text{start} \longrightarrow q_0 \end{array}$</td>
</tr>
<tr>
<td>$a$ (for $a \in \Sigma$)</td>
<td>$\begin{array}{c} \text{start} \longrightarrow q_0 \quad \text{a} \quad \longrightarrow f \end{array}$</td>
</tr>
</tbody>
</table>

$e = e_1 + e_2$

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 $\mathcal{L}(M_1) = \mathcal{L}(e_1)$, $\mathcal{L}(M_2) = \mathcal{L}(e_2)$</td>
<td><img src="image1" alt="Diagram" /></td>
</tr>
<tr>
<td>$e = (e_1)^*$ with $\mathcal{L}(M_1) = \mathcal{L}(e_1)$</td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Proposition. For every regular expression $e$ there is an NFA-$\lambda$ $M_e$ such that

$$\mathcal{L}(M_e) = \mathcal{L}(e).$$

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 $\mathcal{L}(M) = L$).
Avoiding non-determinism

We can transform any NFA (and NFA-$\lambda$) into a DFA that accepts the same language.

Idea:

- Keep track of the set of all states you can go to!
- States of the DFA are sets-of-states from the original NFA-$\lambda$.
- A set 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 $\overline{M}$ as $(\overline{Q}, \overline{q_0}, \overline{\delta}, F)$ where

\[
\begin{align*}
\overline{Q} & = \mathcal{P} Q \\
\overline{q_0} & = \{ q_0 \} \\
\overline{\delta}(H, a) & = \bigcup_{q \in H} \delta(q, a), \quad \text{for } H \subseteq Q, \\
\overline{F} & = \{ H \subseteq Q \mid H \cap F \neq \emptyset \}
\end{align*}
\]

If $M$ is an NFA-$\lambda$, we define

\[
\begin{align*}
\overline{\delta}(H, a) & = \bigcup_{q \in H} \bigcup_{p \in \lambda\text{-closure}(q)} \lambda\text{-closure}(\delta(p, a)) \\
\overline{F} & = \{ H \subseteq Q \mid \lambda\text{-closure}(H) \cap F \neq \emptyset \}
\end{align*}
\]
Correctness

Given \( M \), an NFA-\( \lambda \), we have defined the DFA \( \overline{M} \) by

\[
\overline{q}_0 = \{q_0\} \\
\overline{\delta}(H, a) = \bigcup_{q \in H} \bigcup_{p \in \lambda\text{-closure}(q)} \lambda\text{-closure}(\delta(p, a)) \\
\overline{F} = \{H \subseteq Q \mid \lambda\text{-closure}(H) \cap F \neq \emptyset\}
\]

**Theorem** \( M \) and \( \overline{M} \) accept the same languages.

**Proof:** This follows from

**Lemma**

\[\delta^*(q, w) \cap F \neq \emptyset \iff \overline{\delta}^*(\{q\}, w) \in \overline{F}\]

(Take \( q := q_0 \))

**Proof** of the Lemma: induction on \( w \), considering the cases \( w = \lambda \) and \( w = au \).
Conclusion. Every NFA-λ (or NFA) $M$ can be turned into a DFA $\overline{M}$ 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