Finite State Automata

Automata: Theory and Practice

Paritosh K. Pandya

(TIFR, Mumbai, India)

Unversity of Trento

10-24 May 2005

Finite Word Langauges

- Alphabet Σ is collection of symbols (letters). E.g. $\Sigma = \{a, b\}$.
- Finite word is a finite sequence of letters. E.g. $aabb$. Set of all finite words is denoted by Σ^*.
- Langauge U is a set of words, i.e. $U \subseteq \Sigma^*$.

  Example: (a) Words over $\Sigma = \{a, b\}$ with equal number of a and b. E.g. $aabb$ or $abba$.

Language recognition problem: To determine whether a word belongs to a language. *Most computational problems can be encoded as language recognition problem*.

Automata are computational devices to solve langauge recognition problems.

Finite State Automata

Basic model of computational systems with finite memory.

Widely applicable

- Embedded System Controllers.
  Languages: Esterel, Lustre, Verilog, SMV.
- Synchronous Circuits.
- Regular Expression Pattern Matching
  Grep, Lex, Perl, Awk, Emacs.
- Protocols
  Network Protocols
  Architecture: Bus, Cache Coherence ...
  Mobile Telephony.

Good decision procedures

Topics

- DFA, NFA and Equivalence
- Closure Properties and Decision Problems
- Regular Expressions and McNaughton Yamada Lemma
- Homomorphisms
- DFA minization
- Pumping Lemma
- Myhill Nerode Theorem
- Bisimulation and collapsing nondeterministic automta

Textbook: Dexter Kozen, *Automata and Computability*. Springer, 1997.

Notation

$a, b \in \Sigma$ finite alphabet.

$u, v, w \in \Sigma^*$ finite words.

ϵ empty word.

$u.v$ catenation.

$u^i = u.u. \ldots .u$ repeated i-times.

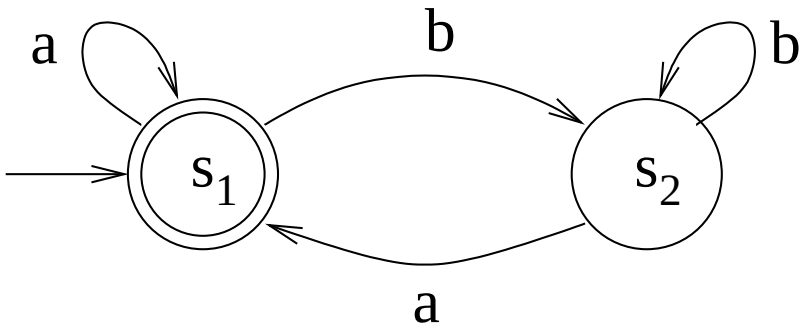
$U, V \subseteq \Sigma^*$ Finite word languages.

For a set S we use 2^S to denote all its subsets.

$$2^S \stackrel{\text{def}}{=} \{X \mid X \subseteq S\}.$$

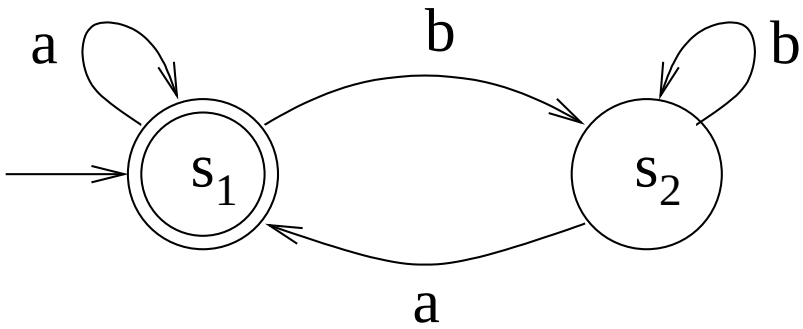
Deterministic Finite State Automaton

Example: DFA A_1 over $\Sigma = \{a, b\}$.



Deterministic Finite State Automaton

Example: DFA A_1 over $\Sigma = \{a, b\}$.



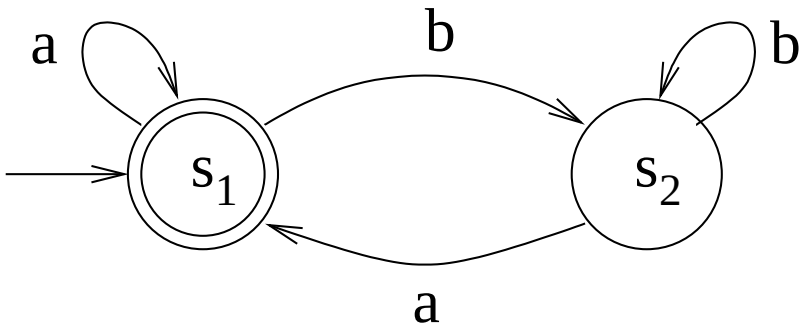
The run of A_1 over the word $abba$ is :

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2 \xrightarrow{a} s_1.$$

For a given word the run is unique.

Deterministic Finite State Automaton

Example: DFA A_1 over $\Sigma = \{a, b\}$.



The run of A_1 over the word $abba$ is :

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2 \xrightarrow{a} s_1.$$

For a given word the run is unique.

Recognises words which do not end in b.

DFA Definition

DFA is $(Q, \Sigma, \delta, I, F)$

Q Finite set of states.
Σ is the finite alphabet.
$q_0 \in Q$ the initial states.
$F \subseteq Q$ set of final states.
$\delta : Q \times \Sigma \to Q$, transition function (total).

Notation: We use $q \xrightarrow{a} q'$ to denote $\delta(q, a) = q'$.

- A run of DFA A on $u = a_0, a_1, \ldots, a_{n-1}$ is a sequence of states $q_0, q_1, \ldots, q_n$ s.t. $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i < n$.
- For a given word w the DFA has a unique run.
- A run accepting if last state $q_n \in F$.

Regular Langauges

- Language accepted by A

  $$L(A) = \{u \in \Sigma^* \mid A \text{ has an accepting run on } u\}.$$

  A langauge accepted by a DFA is called a regular language.

Regular Langauges

- Language accepted by A

  $$L(A) \;=\; \{u \in \Sigma^* \;\mid\; A \text{ has an accepting run on } u\}.$$

  A langauge accepted by a DFA is called a regular language.

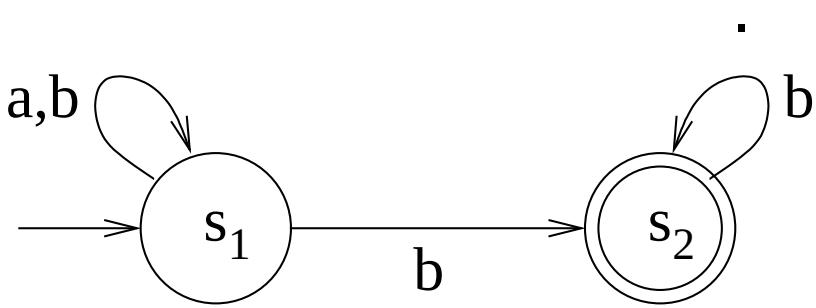
- Define $\hat{\delta}(q,u)$ inductively over $u \in \Sigma^*$.

  $$\hat{\delta}(q,\epsilon) \;=\; q, \qquad \hat{\delta}(q,ua) \;=\; \delta(\hat{\delta}(q,u),a).$$

- For $X \subseteq Q$ define $\hat{\delta}(X,u) \stackrel{\text{def}}{=} \bigcup_{q\in X} \hat{\delta}(q,u)$.

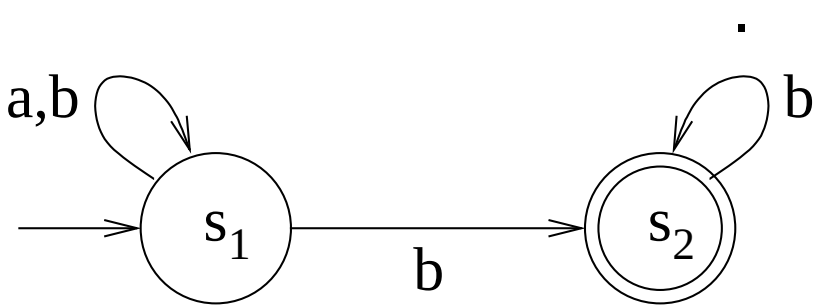
Nondeterministic FSA

NFA A_2



Nondeterministic FSA

NFA A_2



- A run of A_2 over the word $abbb$ is:

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2.$$

- Another run over the same word $abbb$ is:

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_1 \xrightarrow{b} s_1 \xrightarrow{b} s_1.$$

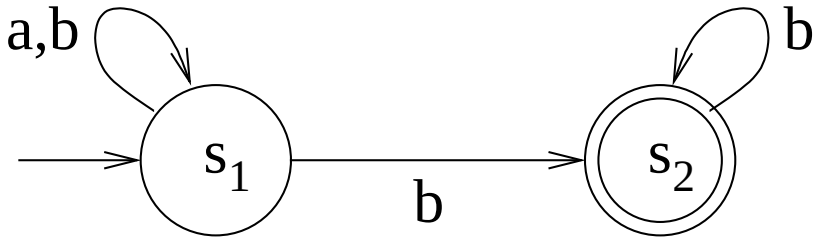
- A_2 has no accepting run over the word aba as:

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} \times$$

NFA can have 0, 1 or more than one runs on a given word.

Nondeterministic FSA

NFA A_2 recognises words which end in b.



- A run of A_2 over the word $abbb$ is:

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2.$$

- Another run over the same word $abbb$ is:

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_1 \xrightarrow{b} s_1 \xrightarrow{b} s_1.$$

- A_2 has no accepting run over the word aba as:

$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} \times$$

NFA can have 0, 1 or more than one runs on a given word.

NFA Definition

Nodeterministic Finite State Automaton:

NFA is $(Q, \Sigma, \delta, I, F)$

Q Finite set of states.

$I \subseteq Q$ set of initial states.

$F \subseteq Q$ set of final states.

$\delta \subseteq Q \times \Sigma \times Q$ transition relation (edges).

We use $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \delta$.

Alternate Definition of δ

$$\delta : Q \times \Sigma \to 2^Q.$$

Exercise: Given that $|Q| = n$ and $|\Sigma| = k$, give the number of syntactically distinct NFAs.

NFA Runs

- A run of NFA A on $u = a_0, a_1, \ldots, a_{n-1}$ is a sequence of states $q_0, q_1, \ldots, q_n$ s.t. $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i < n$ and $q_0 \in I$.
- For a given word w the NFA can have many runs.
- A run accepting if last state $q_n \in F$.
- Language accepted by A is $L(A) = \{u \in \Sigma^* \mid A \text{ has an accepting run on } u\}$.

NFA Runs

- A run of NFA A on $u = a_0, a_1, \ldots, a_{n-1}$ is a sequence of states $q_0, q_1, \ldots, q_n$ s.t. $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i < n$ and $q_0 \in I$.
- For a given word w the NFA can have many runs.
- A run accepting if last state $q_n \in F$.
- Language accepted by A is $L(A) = \{u \in \Sigma^* \mid A \text{ has an accepting run on } u\}$.

Define $\hat{\delta}(S, w)$ inductively on w as follows.

$$\hat{\delta}(S, a) = \cup_{s \in S} \ \delta(s, a)$$

$$\hat{\delta}(S, \epsilon) = S, \qquad \hat{\delta}(S, wa) = \hat{\delta}(\hat{\delta}(S, w), a).$$

Claim $x \in L(A) \quad \Leftrightarrow \quad \hat{\delta}(I, x) \cap F \neq \emptyset$

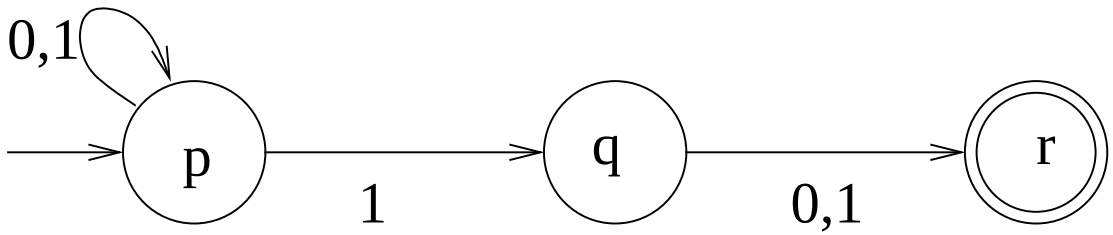
Subset Construction

Theorem (determinisation) Given NFA $A = (Q, \Sigma, \delta, I, F)$ we can construct a DFA A' s.t. $L(A) = L(A')$.

Subset Construction

Theorem (determinisation) Given NFA $A = (Q, \Sigma, \delta, I, F)$ we can construct a DFA A' s.t. $L(A) = L(A')$.

Subset construction:



Consider $\hat{\delta}(I, w)$ the set of states reached after word w. We define $\Delta(S, a)$.

	0	1
$\rightarrow \{p\}$	$\{p\}$	$\{p, q\}$
$\{p, q\}$	$\{p, r\}$	$\{p, q, r\}$
$\{p, r\}$	$\{p\}$	$\{p, q\}$
$\{p, q, r\}$	$\{p, r\}$	$\{p, q, r\}$

Subset Construction (cont)

Let $A' = (2^Q, \Sigma, \Delta, I, F')$ where

$$\Delta(S, a) = \{q \mid q \in \delta(p, a) \wedge p \in S\}$$

$$F' = \{S \in 2^Q \mid S \cap F \neq \emptyset\}.$$

Claim: $L(A) = L(A')$.

Proof Method: Show that $\hat{\delta}(S, w) = \hat{\Delta}(S, w)$.

Complexity $|A'| \leq 2^{|A|}$.

Lower Bound Example: Consider NFA recognising words over $\{0, 1\}$ such that the n-th last symbol is 1.

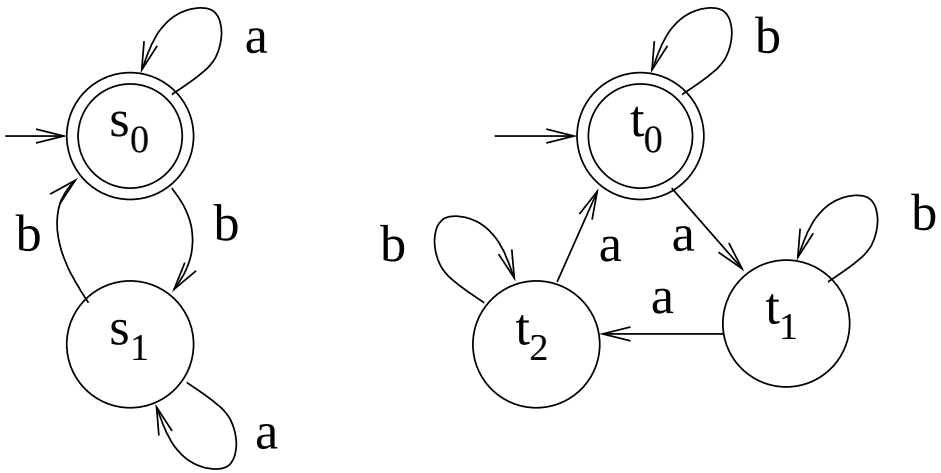
Claim: Equivalent DFA will have at least 2^n states.
(Home Exercise: Prove this.)

Closure Properties

Theorem (boolean closure) Given NFA A_1, A_2 over Σ we can construct NFA A over Σ s.t.

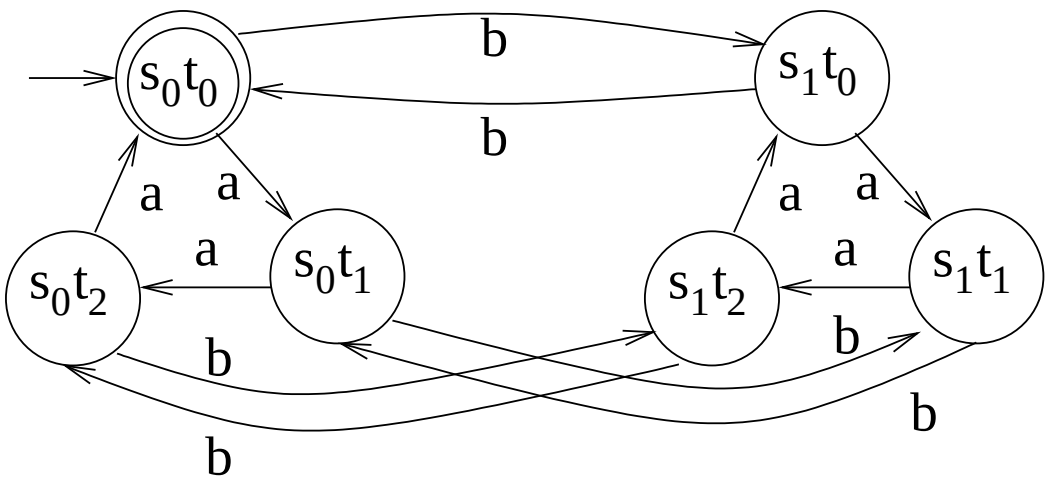
- $L(A) = \overline{L(A_1)}$ (Complement). Size: $|A| = 2^{|A_1|}$.
- $L(A) = L(A_1) \cup L(A_2)$ (union). Size: $|A| = |A_1| + |A_2|$.
- $L(A) = L(A_1) \cap L(A_2)$ (intersection). $|A| = |A_1| \times |A_2|$.

Synchronous Product: Example



- A_1 recognises words with even number of b.
- A_2 recognises words with number of $a \ mod \ 3 \ = 0$.

The Product Automaton $A_1 \times A_2$ with $F = \{s_0, t_0\}$.



Synchronous Product Construction

Let $A_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$. Then,

$A_1 \times A_2 = (Q, \Sigma, \delta, I, F)$ where

- $Q = Q_1 \times Q_2$. $\quad I = I_1 \times I_2$.
  $F = F_1 \times F_2$.
- $< p, q > \xrightarrow{a} < p', q' >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$.

Theorem $L(A_1 \times A_2) = L(A_1) \cap L(A_2)$.

Semantics of many concurrent systems E.g. Esterel, Statecharts, Lustre, Synchronous circuits, SMV.

Synchronised Product Construction

Let $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$ and $A_2 = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$. Then,

$A_1 \| A_2 = (Q, \Sigma, \delta, I, F)$ where

- $Q = Q_1 \times Q_2$. $\quad \Sigma = \Sigma_1 \cup \Sigma_2$.
  $I = I_1 \times I_2$. $\quad F = F_1 \times F_2$.
- $< p, q > \xrightarrow{a} < p', q' >$ if $a \in \Sigma_1 \cap \Sigma_2$ and $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$.
- $< p, q > \xrightarrow{a} < p', q >$ if $a \in \Sigma_1$, $a \notin \Sigma_2$ and $p \xrightarrow{a} p'$.
- $< p, q > \xrightarrow{a} < p, q' >$ if $a \notin \Sigma_1$, $a \in \Sigma_2$ and $q \xrightarrow{a} q'$.

Asynchronous Product Construction

Let $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$ and $A_2 = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$. Then,

$A_1 \|_A A_2 = (Q, \Sigma, \delta, I, F)$ where

- $Q = Q_1 \times Q_2$. $\quad \Sigma = \Sigma_1 \cup \Sigma_2$.
  $I = I_1 \times I_2$. $\quad F = F_1 \times F_2$.
- $< p, q > \xrightarrow{a} < p', q >$ if $a \in \Sigma_1$ and $p \xrightarrow{a} p'$.
- $< p, q > \xrightarrow{a} < p, q' >$ if $a \in \Sigma_2$ and $q \xrightarrow{a} q'$.

Decision Problems

Theorem (Emptiness) Given NFA A we can decide whether $L(A) = \emptyset$.

Method Forward/Backward Reachability in Automaton graph using say Depth First Search. Complexity is $O(|Q| + |\delta|)$. *In practice, symbolic techniques are used and often more effective.*

Theorem (Language Containment) Given NFA A_1 and A_2 we can decide whether $L(A_1) \subseteq L(A_2)$.

Method: $L(A_1) \subseteq L(A_2)$ iff $L(A_1) \cap \overline{L(A_2)} = \emptyset$. Complexity is $O(|A_1| \times 2^{|A_2|})$.

Kleene Closure

Let $U, V \subseteq \Sigma^*$.

- (Catenation) Let $U \cdot V \stackrel{\text{def}}{=} \{x \cdot y \mid x \in U \wedge y \in V\}$.
- (Iteration) Let $U^i \stackrel{\text{def}}{=} \{x_1 \cdot x_2 \cdot \ldots \cdot x_i \mid x_k \in U, 1 \leq k < i\}$.
  By convention $U^0 \stackrel{\text{def}}{=} \{\epsilon\}$.
- (Kleene Closure) Let $U^* \stackrel{\text{def}}{=} \bigcup_{0 \leq i} U^i$.

Theorem Regular langauges are closed under the operations $U \cdot V$ and $U \cup V$ and U^*.
Proof method: Using ϵ-NFA.

Regular Expressions

Syntax: $\emptyset \mid \epsilon \mid a \mid reg_1.reg_2 \mid reg_1 + reg_2 \mid reg^*$.

Regular Expressions

Syntax: $\emptyset \mid \epsilon \mid a \mid reg_1.reg_2 \mid reg_1 + reg_2 \mid reg^*$.

Language $L(reg)$ is defined inductively:

- $L(\emptyset) = \emptyset$, $L(a) = \{a\}$, $L(\epsilon) = \{\epsilon\}$.
- $L(re_1 \cdot re_2) = L(re_1) \cdot L(re_2)$,
  $L(re_1 + re_2) = L(re_1) \cup L(re_2)$,
  $L(re^*) = (L(re_1))^*$

Example: $a^*.(b + bb).a^*$. Words over a, b having either 1 b or 2 consecutive b.

Regular Expressions

Syntax: $\emptyset \mid \epsilon \mid a \mid reg_1.reg_2 \mid reg_1 + reg_2 \mid reg^*$.

Language $L(reg)$ is defined inductively:

- $L(\emptyset) = \emptyset$, $L(a) = \{a\}$, $L(\epsilon) = \{\epsilon\}$.
- $L(re_1 \cdot re_2) = L(re_1) \cdot L(re_2)$,
  $L(re_1 + re_2) = L(re_1) \cup L(re_2)$,
  $L(re^*) = (L(re_1))^*$

Example: $a^*.(b + bb).a^*$. Words over a, b having either 1 b or 2 consecutive b.

Equivalence of RegExp and FSA (McNaughton, Yamada)

Theorem: For every regular expression reg we can construct a language equivalent ϵ-NFA of size $O(|reg|)$.

Automata to RegExp

Theorem: For every NFA $A = (Q, \Sigma, \delta, I, F)$ we can construct a langauge equivalent regular expression $reg(A)$.

Automata to RegExp

Theorem: For every NFA $A = (Q, \Sigma, \delta, I, F)$ we can construct a langauge equivalent regular expression $reg(A)$.

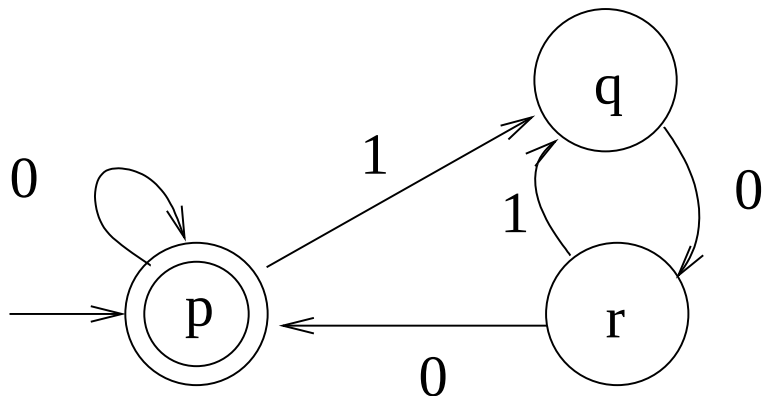
Construction Define regular expression $\alpha_{p,q}^{X}$ for $X \subseteq Q$ and $p, q \in Q$. This denotes set of words w such that A has a run on w from p to q where all intermediate states are from X.

Automata to RegExp

Theorem: For every NFA $A = (Q, \Sigma, \delta, I, F)$ we can construct a langauge equivalent regular expression $reg(A)$.

Construction Define regular expression $\alpha_{p,q}^{X}$ for $X \subseteq Q$ and $p, q \in Q$. This denotes set of words w such that A has a run on w from p to q where all intermediate states are from X.

Example:



$\alpha_{p,r}^{q} = 10$ $\alpha_{p,r}^{p,q} = 0^* \cdot 1 \cdot 0$

Aim: To compute $\alpha_{p,p}^{p,q,r}$

Automata to RegExp (2)

$\alpha_{p,r}^{X}$ can be recursively defined using the following rules:

- $\alpha_{p,r}^{\emptyset} = a_1 + \ldots + a_k$
  where $r \in \delta(p, a_i)$ provided $p \neq r$

Automata to RegExp (2)

$\alpha_{p,r}^{X}$ can be recursively defined using the following rules:

- $\alpha_{p,r}^{\emptyset} = a_1 + \ldots + a_k$
  where $r \in \delta(p, a_i)$ provided $p \neq r$
- $\alpha_{p,p}^{\emptyset} = a_1 + \ldots + a_k + \epsilon$
  where $p \in \delta(p, a_i)$

Automata to RegExp (2)

$\alpha_{p,r}^{X}$ can be recursively defined using the following rules:

- $\alpha_{p,r}^{\emptyset} = a_1 + \ldots + a_k$
  where $r \in \delta(p, a_i)$ provided $p \neq r$
- $\alpha_{p,p}^{\emptyset} = a_1 + \ldots + a_k + \epsilon$
  where $p \in \delta(p, a_i)$
- For any $q \in X$,

$$\alpha_{p,r}^{X} = \alpha_{p,r}^{(X-\{q\})} + \alpha_{p,q}^{(X-\{q\})} \cdot \left(\alpha_{q,q}^{(X-\{q\})}\right)^{*} \cdot \alpha_{q,r}^{(X-\{q\})}$$

Automata to RegExp (2)

$\alpha_{p,r}^{X}$ can be recursively defined using the following rules:

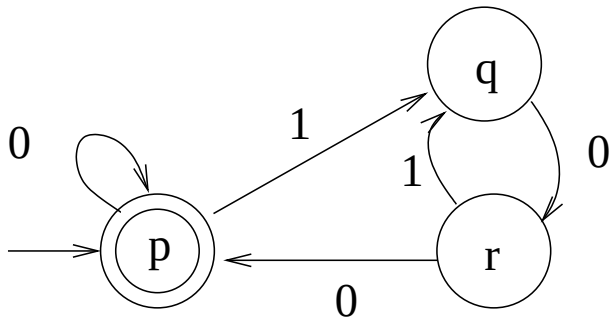
- $\alpha_{p,r}^{\emptyset} = a_1 + \ldots + a_k$
  where $r \in \delta(p, a_i)$ provided $p \neq r$
- $\alpha_{p,p}^{\emptyset} = a_1 + \ldots + a_k + \epsilon$
  where $p \in \delta(p, a_i)$
- For any $q \in X$,

$$\alpha_{p,r}^{X} = \alpha_{p,r}^{(X-\{q\})} + \alpha_{p,q}^{(X-\{q\})} \cdot \left(\alpha_{q,q}^{(X-\{q\})}\right)^{*} \cdot \alpha_{q,r}^{(X-\{q\})}$$

The desired regular expression is given as follows:

$$\sum_{p \in I} \sum_{q \in F} \alpha_{p,q}^{Q}.$$

Example



Compute $\alpha_{p,p}^{p,q,r}$ as

$$\alpha_{p,p}^{p,q,r} \;=\; \alpha_{p,p}^{p,r} \;+\; \alpha_{p,q}^{p,r} \cdot (\alpha_{q,q}^{p,r})^* \cdot \alpha_{q,p}^{p,r}.$$

It is easy to see that

$$\alpha_{p,p}^{p,r} \;=\; 0^*, \qquad \alpha_{p,q}^{p,r} \;=\; 0^* \cdot 1$$

$$\alpha_{q,p}^{p,r} \;=\; 00(0^*), \qquad \alpha_{q,q}^{p,r} \;=\; \epsilon \;+\; 01 \;+\; 00(0^*)1$$

Hence $\alpha_{p,p}^{p,q,r}$

$$=\quad 0^* \;+\; 0^*1 \cdot (\epsilon \;+\; 01 \;+\; 00(0^*)1)^* \cdot 00(0^*)$$

Kleene Algebra

Axiomatizing equivalence of regular expression.

Not covered in this course. (See Kozen.)

Homomorphism

Homomorphism is a map $h : \Sigma^* \to \Gamma^*$ with the property

$$\forall u, v \in \Sigma^*.\ \ h(u \cdot v) \ = \ h(u) \cdot h(v).$$

Consequences:

- $h(\epsilon) = \epsilon$.
- Giving $h(a)$ for $a \in \Sigma$ uniquely determines h.

Homomorphism

Homomorphism is a map $h : \Sigma^* \to \Gamma^*$ with the property

$$\forall u, v \in \Sigma^*.\ \ h(u \cdot v) \ = \ h(u) \cdot h(v).$$

Consequences:

- $h(\epsilon) = \epsilon$.
- Giving $h(a)$ for $a \in \Sigma$ uniquely determines h.

Theorem If $h : \Sigma^* \to \Gamma^*$ is a homomorphism and $B \subseteq \Gamma^*$ is regular then $h^{-1}(B) \ = \ \{x \in \Sigma^* \ \mid \ h(x) \in B\}$ is also regular.

Homomorphism

Homomorphism is a map $h : \Sigma^* \to \Gamma^*$ with the property

$$\forall u, v \in \Sigma^*.\ \ h(u \cdot v) \ = \ h(u) \cdot h(v).$$

Consequences:

- $h(\epsilon) = \epsilon$.
- Giving $h(a)$ for $a \in \Sigma$ uniquely determines h.

Theorem If $h : \Sigma^* \to \Gamma^*$ is a homomorphism and $B \subseteq \Gamma^*$ is regular then $h^{-1}(B) \ = \ \{x \in \Sigma^* \ \mid \ h(x) \in B\}$ is also regular.

Proof method: Given DFA $BB = (Q, \Gamma, \delta, q_0, F)$ for B construct DFA AA for $h^{-1}(B)$ as follows.

$$AA \ \stackrel{\text{def}}{=} \ (Q, \Sigma, \delta', q_0, F) \text{ with } \delta'(q, a) \ = \ \hat{\delta}(q, h(a)).$$

Homomorphism (2)

Theorem If $h : \Sigma^* \to \Gamma^*$ is a homomorphism and $A \subseteq \Sigma^*$ is regular then $h(A) = \{h(x) \mid x \in A\}$ is also regular.

Homomorphism (2)

Theorem If $h : \Sigma^* \to \Gamma^*$ is a homomorphism and $A \subseteq \Sigma^*$ is regular then $h(A) = \{h(x) \mid x \in A\}$ is also regular.

Proof method: Given regular epxression re for A, obtain $h(re)$ by substituting every letter a by $h(a)$. Then $L(h(re)) = h(A)$.

Homomorphism (2)

Theorem If $h : \Sigma^* \rightarrow \Gamma^*$ is a homomorphism and $A \subseteq \Sigma^*$ is regular then $h(A) = \{h(x) \mid x \in A\}$ is also regular.

Proof method: Given regular epxression re for A, obtain $h(re)$ by substituting every letter a by $h(a)$. Then $L(h(re)) = h(A)$.

Theorem (projection) Given $NFA(A_1)$ over Σ and surjection $h : \Sigma \rightarrow \Gamma$, we can construct $NFA(A_2)$ over Γ s.t. $L(A_2) = h(L(A_1))$.

Proof Method: Substitute label a by $h(a)$ in each transition of A_1 to get A_2. Note that this can make DFA into NFA. We have $|A_2| = |A_1|$.

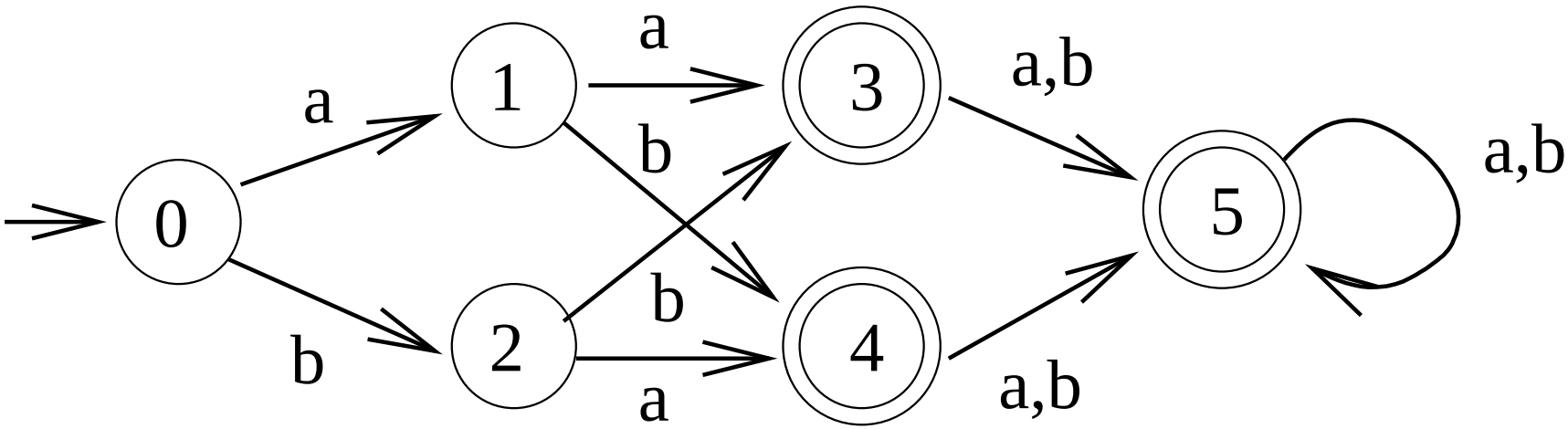
Homomorphism (2)

Theorem If $h : \Sigma^* \rightarrow \Gamma^*$ is a homomorphism and $A \subseteq \Sigma^*$ is regular then $h(A) = \{h(x) \mid x \in A\}$ is also regular.
Proof method: Given regular epxression re for A, obtain $h(re)$ by substituting every letter a by $h(a)$. Then $L(h(re)) = h(A)$.

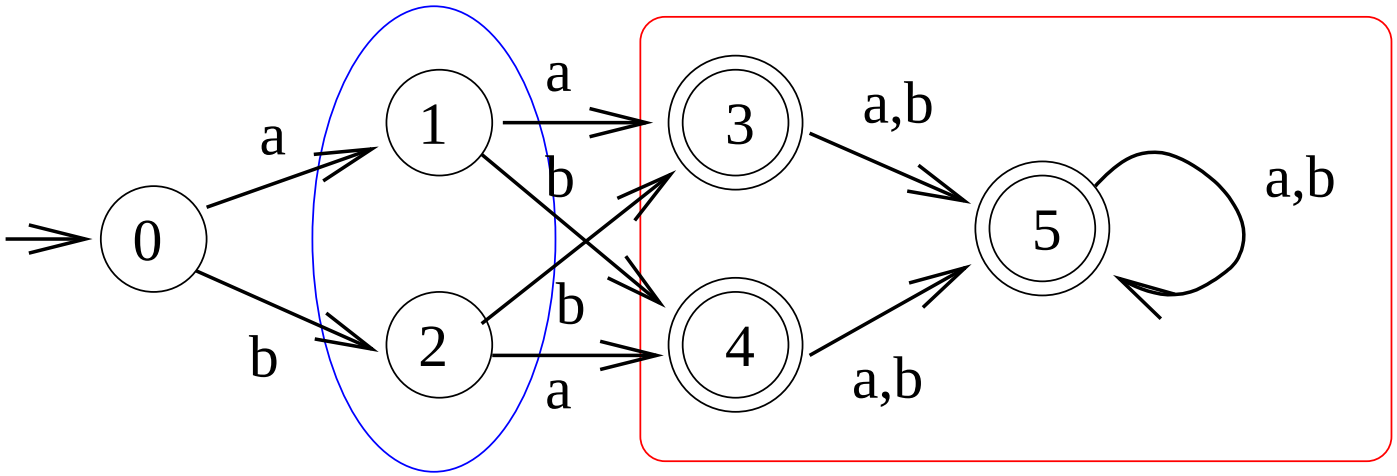
Theorem (projection) Given $NFA(A_1)$ over Σ and surjection $h : \Sigma \rightarrow \Gamma$, we can construct $NFA(A_2)$ over Γ s.t. $L(A_2) = h(L(A_1))$.
Proof Method: Substitute label a by $h(a)$ in each transition of A_1 to get A_2. Note that this can make DFA into NFA. We have $|A_2| = |A_1|$.

Example: Given regular A over $\{0, 1\}$ the language $hamming_3(A)$ is regular.

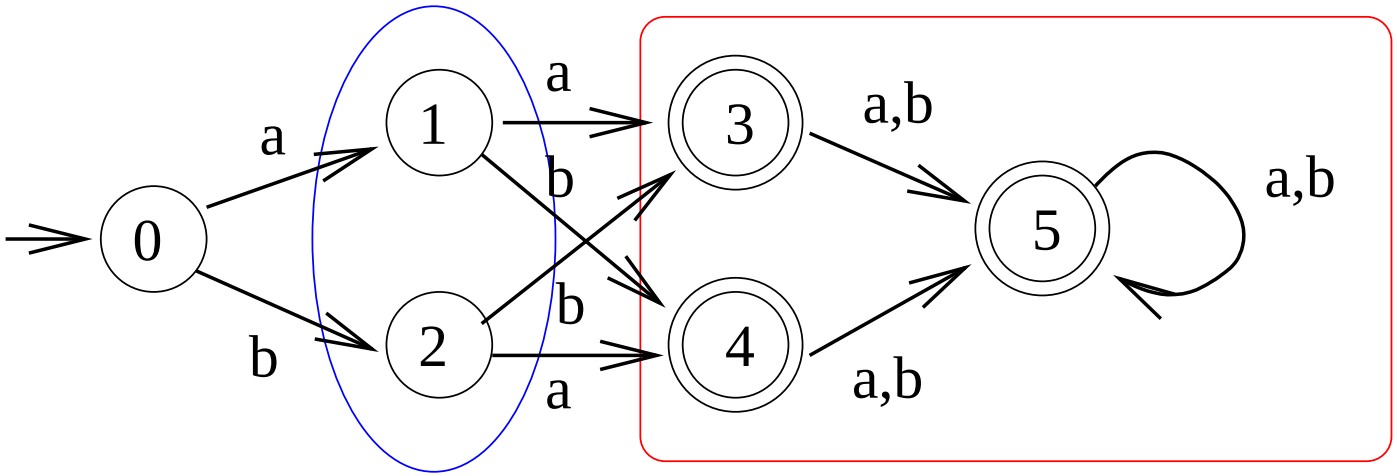
DFA Minimization



DFA Minimization

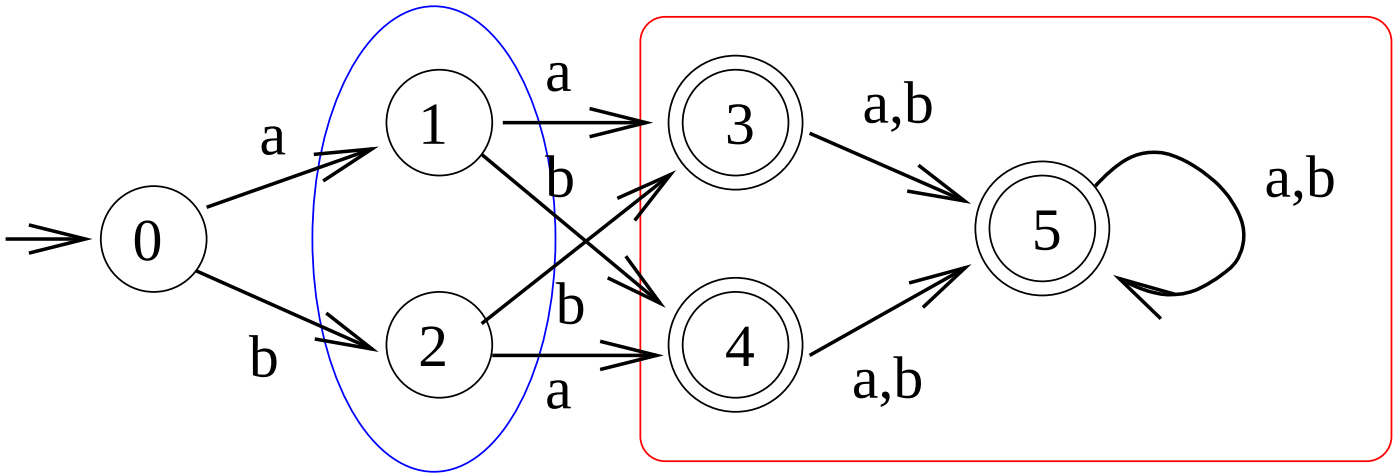


DFA Minimization



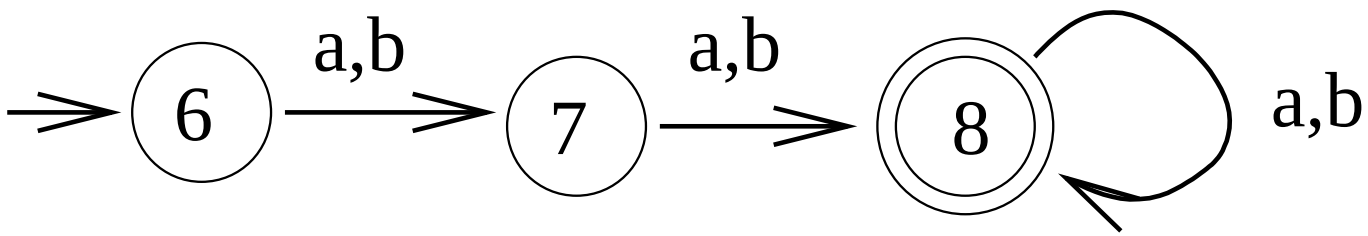
$$p \approx q \quad \stackrel{\text{def}}{=} \quad \forall x \in \Sigma^*.\ \ (\hat{\delta}(p,x) \in F \Leftrightarrow \hat{\delta}(q,x) \in F)$$

DFA Minimization



$$p \approx q \quad \stackrel{\text{def}}{=} \quad \forall x \in \Sigma^*.\ \ (\hat{\delta}(p,x) \in F \Leftrightarrow \hat{\delta}(q,x) \in F)$$

Equivalent Automaton



DFA Minimization (2)

$$p \approx q \quad \stackrel{\text{def}}{=} \quad \forall x \in \Sigma^*.\ (\hat{\delta}(p,x) \in F \Leftrightarrow \hat{\delta}(q,x) \in F)$$

DFA Minimization (2)

$$p \approx q \quad \stackrel{\text{def}}{=} \quad \forall x \in \Sigma^*.\ (\hat{\delta}(p,x) \in F \Leftrightarrow \hat{\delta}(q,x) \in F)$$

Proposition $\approx$ is an equivalence relation, i.e.

(a) $\forall p.\ p \approx p$, (b) $\forall p, q.\ p \approx q \ \Rightarrow\ q \approx p$

(c) $\forall p, q, r.\ p \approx q \wedge q \approx r \ \Rightarrow\ p \approx r$

(Exercise: Check that above properties are true.)

DFA Minimization (2)

$$p \approx q \quad \stackrel{\text{def}}{=} \quad \forall x \in \Sigma^*.\ (\hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F)$$

Proposition $\approx$ is an equivalence relation, i.e.
(a) $\forall p.\ p \approx p$, (b) $\forall p, q.\ p \approx q \ \Rightarrow\ q \approx p$
(c) $\forall p, q, r.\ p \approx q \wedge q \approx r \ \Rightarrow\ p \approx r$
(Exercise: Check that above properties are true.)

$\approx$ partitions Q into equivalence classes. Let $[p]$ denote the equivalence class of p.

Example The classes are $\{0\}$, $\{1, 2\}$ and $\{3, 4, 5\}$.

Quotient Automaton

Given DFA $A = (Q, \Sigma, \delta, [q_0], F)$ and $\approx$ as before, the

Quotient automaton is $A/\approx \stackrel{\text{def}}{=} (Q', \Sigma, \delta', [q_0], F')$, where

$$
\begin{aligned}
Q' &= \{[p] \mid p \in Q\} \\
\delta'([p], a) &= [\delta(p, a)] \\
F' &= \{[f] \mid f \in F\}
\end{aligned}
$$

Quotient Automaton

Given DFA $A = (Q, \Sigma, \delta, [q_0], F)$ and $\approx$ as before, the

Quotient automaton is $A/\approx \stackrel{\text{def}}{=} (Q', \Sigma, \delta', [q_0], F')$, where

$$
\begin{array}{rcl}
Q' & = & \{[p] \mid p \in Q\} \\
\delta'([p], a) & = & [\delta(p, a)] \\
F' & = & \{[f] \mid f \in F\}
\end{array}
$$

Well-formedness

Theorem(Congruence)

$p \approx q \quad \Rightarrow \quad \forall a \in \Sigma.\ \delta(p, a) \approx \delta(q, a)$.

Quotient Automaton

Given DFA $A = (Q, \Sigma, \delta, [q_0], F)$ and $\approx$ as before, the

Quotient automaton is $A/\approx \stackrel{\text{def}}{=} (Q', \Sigma, \delta', [q_0], F')$, where

$$
\begin{array}{rcl}
Q' & = & \{[p] \mid p \in Q\} \\
\delta'([p], a) & = & [\delta(p, a)] \\
F' & = & \{[f] \mid f \in F\}
\end{array}
$$

Well-formedness

Theorem(Congruence)

$p \approx q \quad \Rightarrow \quad \forall a \in \Sigma.\ \delta(p, a) \approx \delta(q, a)$.

Now we can show that quotient automaton recognises the same language.

Theorem $\hat{\delta}'([p], w) \quad = \quad [\hat{\delta}(p, w)]$.

Corollary $L(A/\approx) \quad = \quad L(A)$.

Minimization algorithm

1. Make pairs table with $(p, q) \in Q \times Q$ and $p < q$.
2. Mark (p, q) if $p \in F \wedge q \notin F$ or vice versa.
3. Repeat following steps until no change occurs.
   (a) Pick unmarked state (p, q).
   (b) If $(\delta(p, a), \delta(q, a))$ is marked for some $a \in \Sigma$ then mark (p, q)

Minimization algorithm

1. Make pairs table with $(p, q) \in Q \times Q$ and $p < q$.
2. Mark (p, q) if $p \in F \wedge q \notin F$ or vice versa.
3. Repeat following steps until no change occurs.
   (a) Pick unmarked state (p, q).
   (b) If $(\delta(p, a), \delta(q, a))$ is marked for some $a \in \Sigma$ then mark (p, q)

Termination: In each pass at least one new pair must get marked.

Lemma (p, q) is marked iff $\exists x \in \Sigma^*.\hat{\delta}(p, x) \in F \wedge \hat{\delta}(q, x) \notin F$ or vice versa.

Minimization algorithm

1. Make pairs table with $(p, q) \in Q \times Q$ and $p < q$.
2. Mark (p, q) if $p \in F \wedge q \notin F$ or vice versa.
3. Repeat following steps until no change occurs.
   (a) Pick unmarked state (p, q).
   (b) If $(\delta(p, a), \delta(q, a))$ is marked for some $a \in \Sigma$ then mark (p, q)

Termination: In each pass at least one new pair must get marked.

Lemma (p, q) is marked iff $\exists x \in \Sigma^*.\hat{\delta}(p, x) \in F \wedge \hat{\delta}(q, x) \notin F$ or vice versa.

Question: Can there be a DFA smaller than $A/\approx$ equivalent to A?

Pumping Lemma

Idea: Consider a long word recognized by a DFA with n states. Consider substring y s.t. $|y| \geq n$. Some state during y part of the run must repeat. The string between repeating states can be pumped.

Pumping Lemma

Idea: Consider a long word recognized by a DFA with n states. Consider substring y s.t. $|y| \geq n$. Some state during y part of the run must repeat. The string between repeating states can be pumped.

Lemma For every regular language L
$\exists n > 0$ such that
$\forall$ x, y, z with $x \cdot y \cdot z \in L$ and $|y| \geq n$
$\exists$ u, v, w s.t. $y = u \cdot v \cdot w$ with
$|v| > 0$ and
$x \cdot u \cdot v^i \cdot w \cdot z \in L$ for all $0 \leq i$.

Pumping Lemma

Idea: Consider a long word recognized by a DFA with n states. Consider substring y s.t. $|y| \geq n$. Some state during y part of the run must repeat. The string between repeating states can be pumped.

Lemma For every regular language L
$\exists n > 0$ such that
$\forall$ x, y, z with $x \cdot y \cdot z \in L$ and $|y| \geq n$
$\exists$ u, v, w s.t. $y = u \cdot v \cdot w$ with
$|v| > 0$ and
$x \cdot u \cdot v^i \cdot w \cdot z \in L$ for all $0 \leq i$.

Pumping Lemma is used to prove that some langugages are not regular. E.g. $\{a^{2^n} \mid n \geq 0\}$.

Proving non-regularity

Game with Demon

- Demon claims L is regular and chooses n.
- You choose a string $xyz \in L$ with $|y| \geq n$.
- Demon partitions $y = uvw$ with $|v| \neq 0$.
- You show that $xu(v^i)wz \notin L$ for some i for your choice.

Myhill-Nerode Relations

Let $\equiv$ be an equivalence relation over Σ^*.

- $\equiv$ is right congruent if $\forall x, y \in \Sigma^*,\ a \in \Sigma$ we have

$$x \equiv y \quad \Rightarrow \quad x \cdot a \equiv y \cdot a$$

Myhill-Nerode Relations

Let $\equiv$ be an equivalence relation over Σ^*.

- $\equiv$ is right congruent if $\forall x, y \in \Sigma^*,\ a \in \Sigma$ we have

$$x \equiv y \quad \Rightarrow \quad x \cdot a \equiv y \cdot a$$

- Let $R \subseteq \Sigma^*$ (not necessarily regular).

  $\equiv$ refines R if $\forall x, y \in \Sigma^*$

$$x \equiv y \quad \Rightarrow \quad (x \in R \Leftrightarrow y \in R)$$

Myhill-Nerode Relations

Let $\equiv$ be an equivalence relation over Σ^*.

- $\equiv$ is right congruent if $\forall x, y \in \Sigma^*,\ a \in \Sigma$ we have

$$x \equiv y \quad \Rightarrow \quad x \cdot a \equiv y \cdot a$$

- Let $R \subseteq \Sigma^*$ (not necessarily regular).
  $\equiv$ refines R if $\forall x, y \in \Sigma^*$

$$x \equiv y \quad \Rightarrow \quad (x \in R \Leftrightarrow y \in R)$$

- $\equiv$ is of finite index if $\equiv$ partitions the Σ^* into only finitely many equivalence classes.

Myhill-Nerode Relations

Let $\equiv$ be an equivalence relation over Σ^*.

- $\equiv$ is right congruent if $\forall x, y \in \Sigma^*,\ a \in \Sigma$ we have
$$x \equiv y \quad \Rightarrow \quad x \cdot a \equiv y \cdot a$$
- Let $R \subseteq \Sigma^*$ (not necessarily regular).
$\equiv$ refines R if $\forall x, y \in \Sigma^*$
$$x \equiv y \quad \Rightarrow \quad (x \in R \Leftrightarrow y \in R)$$
- $\equiv$ is of finite index if $\equiv$ partitions the Σ^* into only finitely many equivalence classes.

Equivalence relation $\equiv$ is called Myhill-Nerode Relation refining R if it satisfies all the three properties above.

Machine Equivalence

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ define the induced equivalence $\equiv_A$ over Σ^* as follows:

$$x \equiv_A y \quad \stackrel{\text{def}}{=} \quad \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y).$$

Machine Equivalence

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ define the induced equivalence $\equiv_A$ over Σ^* as follows:

$$x \equiv_A y \quad \stackrel{\text{def}}{=} \quad \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y).$$

Proposition $\equiv_A$ is a Myhill-Nerode relation refining $L(A)$.

Proof Method Check the following:

(a) $\equiv_A$ is an equivalence relation over Σ^*.

(b) $x \equiv_A y \quad \Rightarrow \quad \forall a.\ xa \equiv_A ya$.

(c) $x \equiv_A y \quad \Rightarrow \quad x \in L(A) \ \Leftrightarrow \ y \in L(A)$

(d) $\equiv_A$ is of finite index.

Machine Equivalence

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ define the induced equivalence $\equiv_A$ over Σ^* as follows:

$$x \equiv_A y \quad \stackrel{\text{def}}{=} \quad \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y).$$

Proposition $\equiv_A$ is a Myhill-Nerode relation refining $L(A)$.

Proof Method Check the following:

(a) $\equiv_A$ is an equivalence relation over Σ^*.

(b) $x \equiv_A y \quad \Rightarrow \quad \forall a.\ xa \equiv_A ya$.

(c) $x \equiv_A y \quad \Rightarrow \quad x \in L(A) \quad \Leftrightarrow \quad y \in L(A)$

(d) $\equiv_A$ is of finite index.

Example: We give automaton A and the induced equivalence partitions.

From Equivalence to DFA

Let $\equiv$ be Myhill-Nerode refining R. Define DFA $A_{\equiv} \stackrel{\text{def}}{=} (Q, \Sigma, \delta, q_0, F)$ as follows:

$$Q \stackrel{\text{def}}{=} \{[x] \mid x \in \Sigma^*\}$$
$$q_0 \stackrel{\text{def}}{=} [\epsilon]$$
$$F \stackrel{\text{def}}{=} \{[x] \mid x \in R\}$$
$$\delta([x], a]) \stackrel{\text{def}}{=} [xa].$$

(Check well-formedness.)

From Equivalence to DFA

Let $\equiv$ be Myhill-Nerode refining R. Define DFA $A_{\equiv} \stackrel{\text{def}}{=} (Q, \Sigma, \delta, q_0, F)$ as follows:

$$Q \stackrel{\text{def}}{=} \{[x] \mid x \in \Sigma^*\}$$
$$q_0 \stackrel{\text{def}}{=} [\epsilon]$$
$$F \stackrel{\text{def}}{=} \{[x] \mid x \in R\}$$
$$\delta([x], a]) \stackrel{\text{def}}{=} [xa].$$

(Check well-formedness.)

Theorem $L(A_{\equiv}) = R$.

Lemma $\hat{\delta}([x], y) = [xy]$.

Correspondence

The $\equiv_A$ and $A_{\equiv}$ are inverses of each other.

Theorem $\equiv_{A_{\equiv}} \quad = \quad \equiv$.

Theorem If A is automaton without unreachable states then $A_{\equiv_A}$ is isomorphic to A.

Language Induced Equivalence

An equivalence relation $\equiv_1$ refines equivalence relation $\equiv_2$ provided $x \equiv_1 y \quad \Rightarrow \quad x \equiv_2 y$ (Set theoretically, $\equiv_1 \subseteq \equiv_2$.)
Intuitively, $\equiv_1$ makes finer partitions than $\equiv_2$.

Language Induced Equivalence

An equivalence relation $\equiv_1$ refines equivalence relation $\equiv_2$ provided $x \equiv_1 y \quad \Rightarrow \quad x \equiv_2 y$ (Set theoretically, $\equiv_1 \subseteq \equiv_2$.)
Intuitively, $\equiv_1$ makes finer partitions than $\equiv_2$.

Definition (Language Induced Equivalence) Given $R \subseteq \Sigma^*$,

$$x \equiv_R y \quad \stackrel{\text{def}}{=} \quad \forall z \in \Sigma^*.\ (xz \in R \Leftrightarrow yz \in R).$$

Language Induced Equivalence

An equivalence relation $\equiv_1$ refines equivalence relation $\equiv_2$ provided $x \equiv_1 y \quad \Rightarrow \quad x \equiv_2 y$ (Set theoretically, $\equiv_1 \subseteq \equiv_2$.)
Intuitively, $\equiv_1$ makes finer partitions than $\equiv_2$.

Definition (Language Induced Equivalence) Given $R \subseteq \Sigma^*$,

$$x \equiv_R y \quad \stackrel{\text{def}}{=} \quad \forall z \in \Sigma^*.\ (xz \in R \Leftrightarrow yz \in R).$$

Proposition $\equiv_R$ is right congruent.

Proposition $\equiv_R$ refines R.

Language Induced Equivalence

An equivalence relation $\equiv_1$ refines equivalence relation $\equiv_2$ provided $x \equiv_1 y \quad \Rightarrow \quad x \equiv_2 y$ (Set theoretically, $\equiv_1 \subseteq \equiv_2$.) Intuitively, $\equiv_1$ makes finer partitions than $\equiv_2$.

Definition (Language Induced Equivalence) Given $R \subseteq \Sigma^*$,

$$x \equiv_R y \quad \stackrel{\text{def}}{=} \quad \forall z \in \Sigma^*.\ (xz \in R \Leftrightarrow yz \in R).$$

Proposition $\equiv_R$ is right congruent.

Proposition $\equiv_R$ refines R.

Proposition(Coarseness) Let $\equiv$ be right-congruent refining R. Then, $\equiv \quad \subseteq \quad \equiv_R$.

Myhill-Nerode Theorem

Let $R \subseteq \Sigma^*$. The following statements are equivalent.

1. R is regular
2. There exists a Myhill-Nerode relation refinining R.
3. The relation $\equiv_R$ is of finite index.

The automaton $A_{\equiv_R}$ gives the minimal DFA for R.

State Minimization is optimal

Let A recognise R. Consider the quotient automaton $M = A/\approx$ as before, and assume that it has no inaccessible states.

Lemma $x \equiv_R y \quad \Leftrightarrow \quad x \equiv_M y$

Bisimulation

Postponed to the End of course.

See Kozen Supplementary Lecture B.