More on Finite Automata and Regular Languages

Pumping Lemma is not a Sufficient Condition

Example 1

We know $L = \{b^m c^m | m > 0\}$ is not regular. Let us consider $L' = a^+ L \cup (b + c)^*$. L' is not regular. If L' would be regular, then we can prove that L is regular (using the closure properties we will see next). However, the Pumping lemma does apply for L' with $n = 1$.

Consider string $ab^n c^n$ and partition $\overbrace{\epsilon}^{u}\overbrace{a}^{v}\overbrace{b^n c^n}^{w}$. Then $uv^i w, \forall i \geq 0$ remains in L'.

This shows the Pumping lemma is not a sufficient condition for a language to be regular.

Use of closure properties to show non-regularity

- We can easily prove $L_1 = \{0^n 1^n | n > 0\}$ is not a regular language.
- L_2 = the set of strings with an equal number of 0's and 1's isn't either, but that fact is trickier to prove.
- Regular languages are closed under $\cap$.
- If L_2 were regular, then $L_2 \cap L(0^*1^*) = L_1$ would be, but it isn't.

Closure properties

Let L and M be regular. Then $L = L(R) = L(D)$ and $M = L(S) = L(F)$ for regular expressions R and S, and DFA D and F.
We have seen that RL are closed under the following operations:

- Union : $L \cup M = L(R + S)$ or $L \cup M = L(D \bigoplus F)$
- Complement : $\bar{L} = L(\bar{D})$
- Intersection : $L \cap M = \overline{\bar{L} \cup \bar{M}}$ or $L \cap M = L(D \times F)$
- Difference : $L - M = L \cap \bar{M}$
- Concatenation : $LM = L(RS)$
- Closure : $L^* = L(R^*)$
- Prefix : $Prefix(L) = \{x \mid \exists y \in \Sigma^*, xy \in L\}$ (Hint: in D, make final all states in a path from the start state to final state)
- quotient, morphism, inverse morphism, substitution, ...

Quotient

Definition 2

$L_1, L_2 \subseteq \Sigma^*, L_1/L_2 = \{x \in \Sigma^* \mid \exists y \in L_2, xy \in L_1\}$.

Note: $Pref(L) = L/\Sigma^*$.

Theorem 3

$L, R \subseteq \Sigma^*$. If R is regular, then R/L is also regular.

Proof Idea: $F' = \{q \in Q \mid \exists y \in L, \hat{\delta}(q, y) \in F\}$

Example 4

$L = \{a^{n^2} \mid n \geq 0\}$. $L/L = \{a^{n^2-m^2} \mid m, n \geq 0\} = a(aa)^* + (a^4)^*$.

Morphisms

$h : \Sigma \to \Delta^*$

$h : \Sigma^* \to \Delta^* \quad h(xy) = h(x)h(y), h(\epsilon) = \epsilon$

$h : 2^{\Sigma^*} \to 2^{\Delta^*} \quad h(L) = \bigcup_{x \in L} \{h(x)\}$

Example 5

$h(0) = ab, h(1) = ba, h(2) = \epsilon.$
$h(00212) = abbaba;$
$h(\{0^n 21^n | n \geq 0\}) = \{(ab)^n (ba)^n | n \geq 0\}$

Theorem 6

$h(K \cup L) = h(K) \cup (L);$
$h(K \cdot L) = h(K) \cdot h(L);$
$h(K^*) = h(K)^*.$

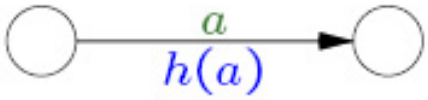
Inverse Morphisms

$h : \Sigma^* \to \Delta^*, K \subseteq \Delta^*$
$h^{-1}(K) = \{x \in \Sigma^* \mid h(x) \in K\}$

Theorem 7

Regular languages are closed under inverse morphism.

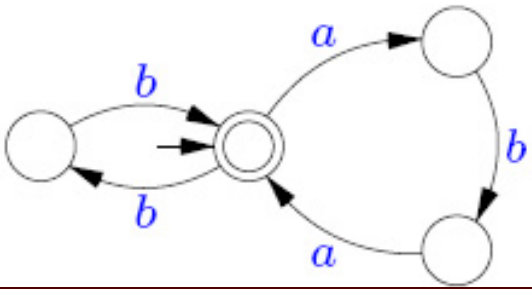
Proof Idea:

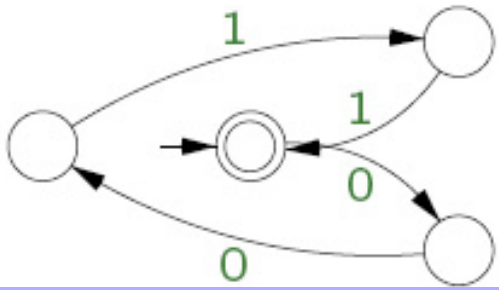


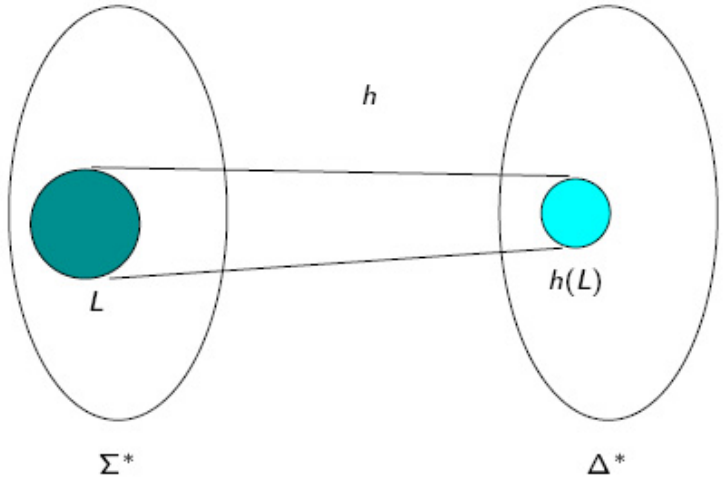
$\delta'(p, a) = \delta(p, h(a))$

$h : 0 \mapsto ab, 1 \mapsto ba$

$h^{-1}(\ \{bb, aba\}^*\ ) = \{0011\}^*$







Shuffle

Definition 8

$x\|\epsilon = \epsilon\|x = \{x\}$
$ax\|by = a(x\|by) \cup b(ax\|y)$
$K\|L = \bigcup_{x\in K, y\in L} x\|y$

$abb\|aca = \{aabbca, aabcba, aabcab, aacabb, aacbab,$
$aacbba, abbaca, ababca, abacba, abacab, acabba, acabab, acaabb\}.$

Theorem 9

If K, L are regular, so is $K\|L$.

Shuffle (cont'd)

Proof.

copies of alphabet

$\Sigma,\ \Sigma_1 = \{\, a_1 \mid a \in \Sigma \,\},\ \Sigma_2 = \{\, a_2 \mid a \in \Sigma \,\}$

$h_1 : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto a \qquad a_2 \mapsto \epsilon$

$h_2 : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto \epsilon \qquad a_2 \mapsto a$

$g : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto a \qquad a_2 \mapsto a$

$$\begin{array}{ccccc} abbba & \overset{h_1}{\leftarrow} & a_1b_1a_2c_2b_1a_2c_2b_1a_1 & \overset{h_2}{\rightarrow} & acac \\ \in K & & \downarrow g & & \in L \\ & & abacbacba & & \end{array}$$

$$K \parallel L = g(\ h_1^{-1}(K) \cap h_2^{-1}(L)\ )$$

Definition 10

$\frac{1}{2}L = \{x \in \Sigma^* | \exists y \in \Sigma^*, xy \in L;\ \ |y| = |x|\}$.

Theorem 11

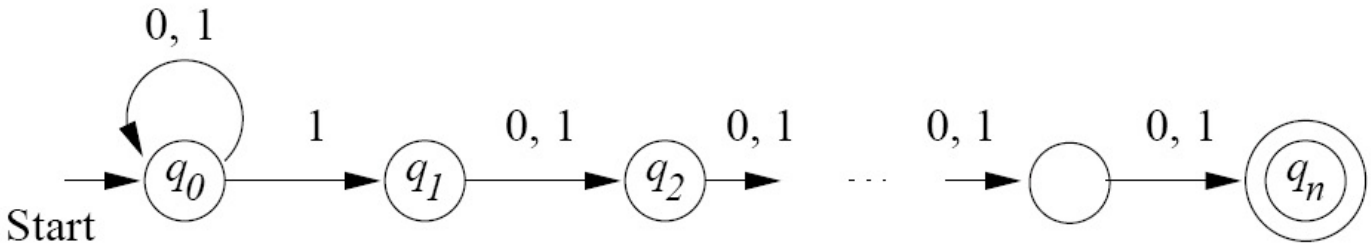
If L is regular, so is $\frac{1}{2}L$.

Proof.

guess middle state, simulate halves in parallel
$Q' = \{q_0'\} \cup Q \times Q \times Q$ (Note: middle, 1st, 2nd)
$\delta'(q_0', \epsilon) = \{[q, q_0, q] | q \in Q\}$ – ϵ-move
$\delta'([q, p, r], a) = \{[q, \delta(p, a), \delta(r, b)] | b \in \Sigma\}$
$F' = \{[q, q, p] | q \in Q, p \in F\}$ □

Exponential Blow-Up in Subset Construction

There is an NFA N with $n+1$ states that has no equivalent DFA with fewer than 2^n states.



$L(N) = \{x1c_2c_3 \cdots c_n : x \in \{0,1\}^*, c_i \in \{0,1\}\}$.

- Suppose an equivalent DFA D with fewer than 2^n states exists.
- D must remember the last n symbols it has read.
- There are 2^n bitsequences $a_1a_2 \cdots a_n$.
- $\exists q, a_1a_2 \cdots a_n, b_1b_2 \cdots b_n : q \in \hat{\delta}_N(q_0, a_1a_2 \cdots a_n), \quad q \in \hat{\delta}_N(q_0, b_1b_2 \cdots b_n), a_1a_2 \cdots a_n \neq b_1b_2 \cdots b_n$

Exponential Blow-Up (Cont'd)

$a_1 \cdots a_{i_1} 1 a_{i+1} \cdots a_n$
$b_1 \cdots b_{i_1} 0 b_{i+1} \cdots b_n$

Now

$$\hat{\delta}_N(q_0, a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n 0^{i-1}) = \hat{\delta}_N(q_0, b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n 0^{i-1})$$

And

$$\hat{\delta}_N(q_0, a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n 0^{i-1}) \in F_D$$

$$\hat{\delta}_N(q_0, b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n 0^{i-1}) \notin F_D$$

– A contradiction!

Decision Properties

- A decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

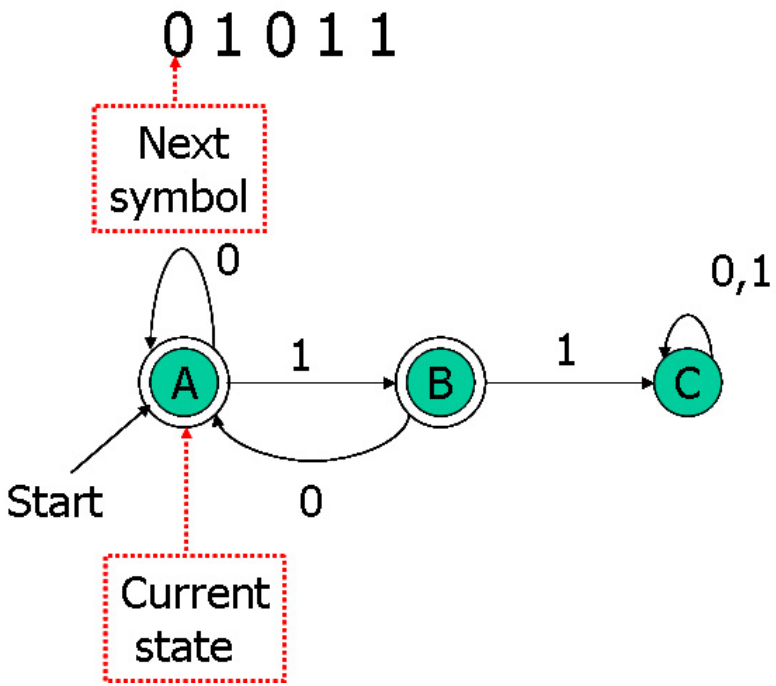
- Example: Is language L empty?
  - The representation is a DFA (or a RE that you will convert to a DFA).
  - Can you tell if $L(A) = \emptyset$ for DFA A?

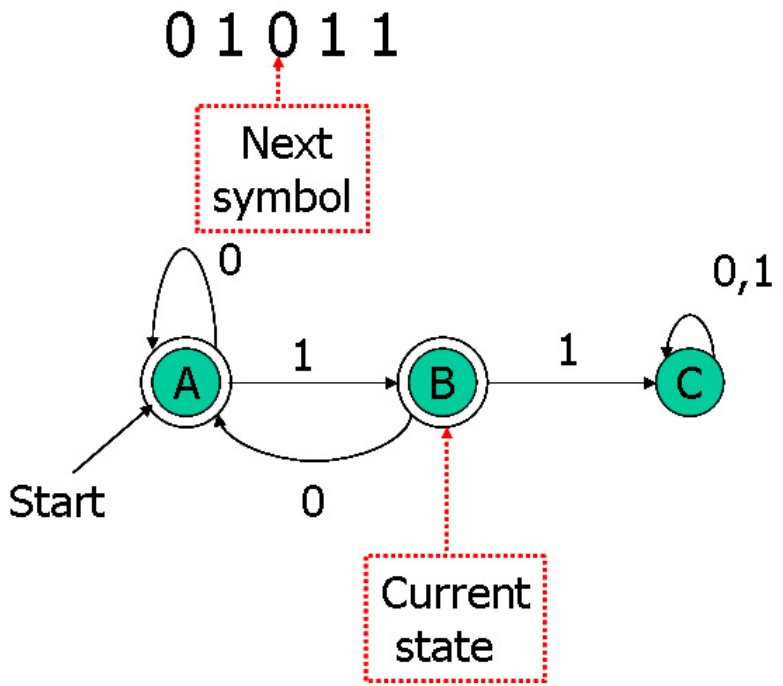
Why Decision Properties

- When we talked about protocols represented as DFAs, we noted that important properties of a good protocol were related to the language of the DFA.
- Example: Does the protocol terminate? = Is the language finite?
- Example: Can the protocol fail? = Is the language nonempty?

The Membership Question

- Our first decision property is the question: is string w in regular language L?
- Assume L is represented by a DFA A.
- Simulate the action of A on the sequence of input symbols forming w



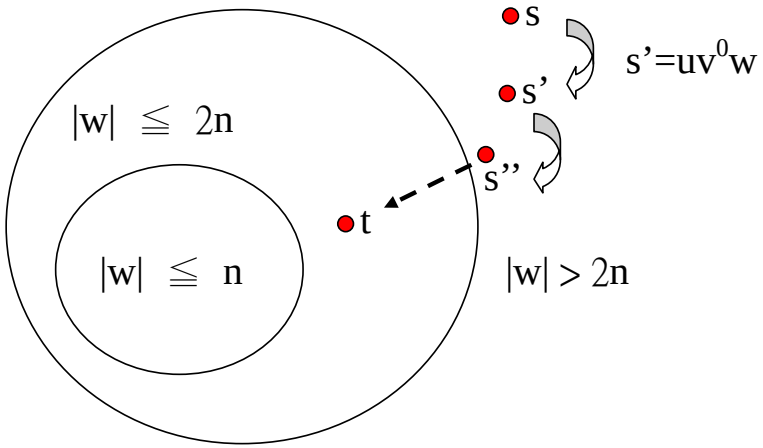


The Emptiness Problem

- Given a regular language, does the language contain any string at all.
- Assume representation is DFA.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes, else no.

The Infiniteness Problem

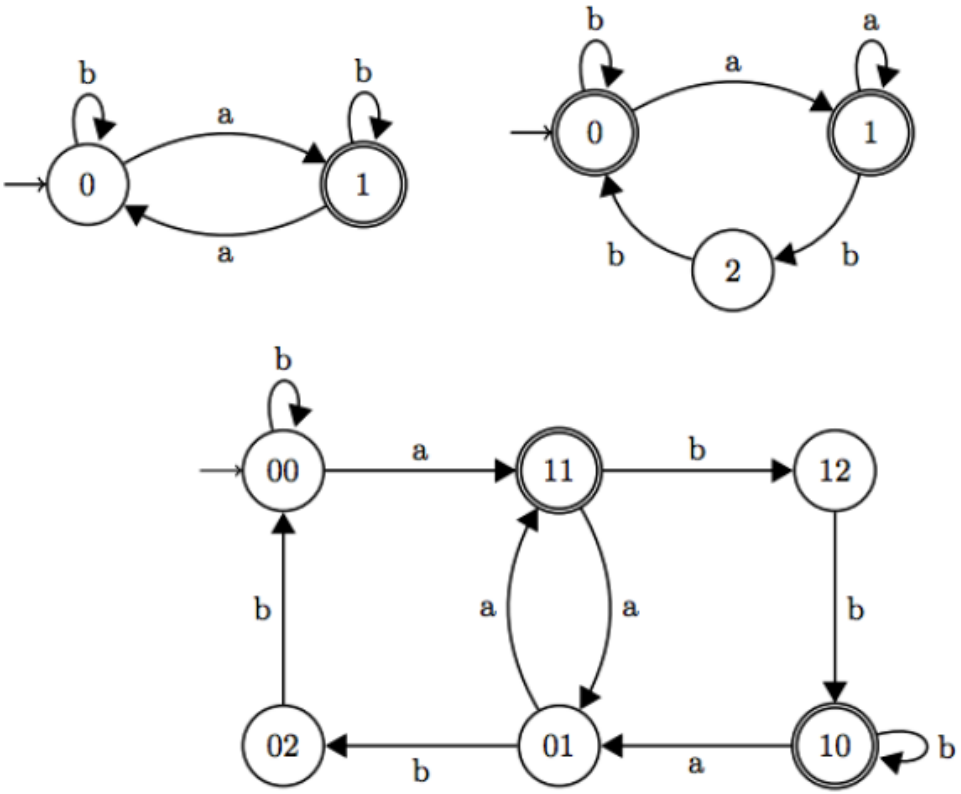
- Is a given regular language infinite?
- Start with a DFA for the language.
- Key idea: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- Second key idea: if there is a string of length $> n$ (= number of states) in L, then there is a string of length between n and $2n - 1$.

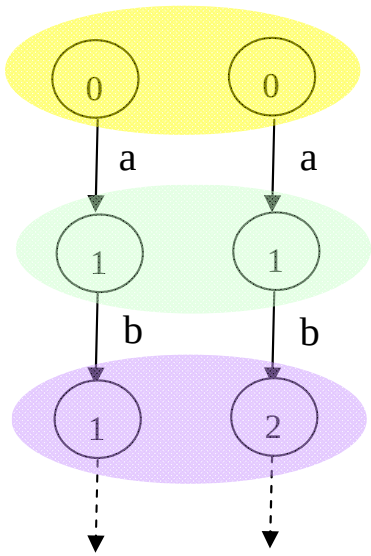


- Test for membership all strings of length between n and $2n - 1$. If any are accepted, then infinite, else finite.

The Product Automaton $M \times N$

Idea: Running two automata M and N in parallel.



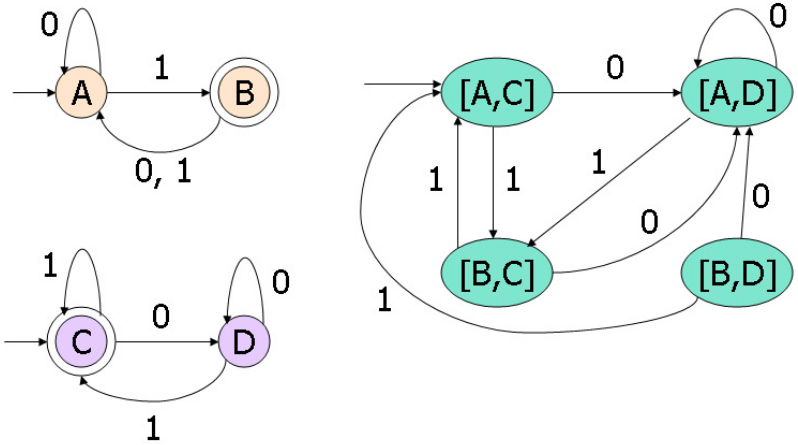


Running the two FAs in parallel

The Equivalence Problem

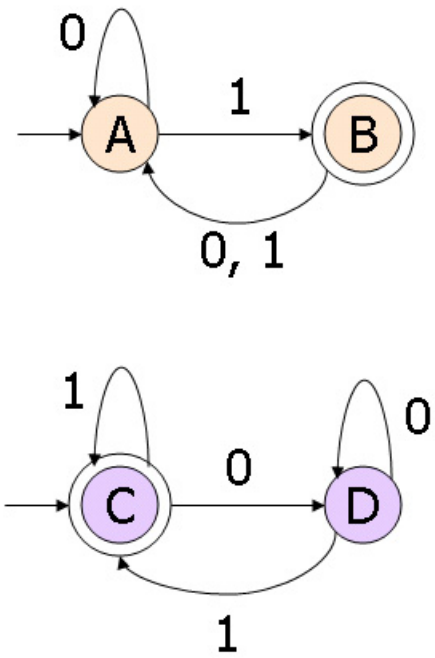
- Given regular languages L and M, is $L = M$?
- Algorithm involves constructing the product DFA from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states $Q \times R$. I.e., pairs $[q, r]$ with q in Q, r in R.

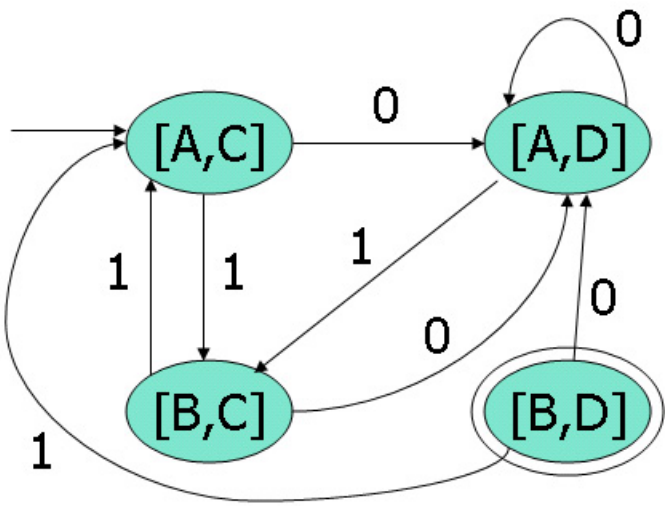
- Make the final states of the product DFA be those states $[q, r]$ such that exactly one of q and r is a final state of its own DFA. Thus, the product accepts w iff w is in exactly one of L and M.
- The product DFA's language is empty iff L = M.



The Containment Problem

- Given regular languages L and M, is $L \subseteq M$?
- Algorithm also uses the product automaton.
- How do you define the final states $[q, r]$ of the product so its language is empty iff $L \subseteq M$?
  - Answer: q is final; r is not.





Note: the only final state is unreachable, so containment holds.

The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA's we can, given a DFA A find the DFA with the fewest states accepting $L(A)$.
- Test all smaller DFA's for equivalence with A.
- But that's a terrible algorithm.

– **Efficient State Minimization**

- Construct a table with all pairs of states.
- If you find a string that distinguishes two states (takes exactly one to an accepting state), mark that pair.
- Algorithm is a recursion on the length of the shortest distinguishing string.

Equivalence Relation

Definition 12

A binary relation R on a set S is a subset of $S \times S$. An equivalence relation on a set satisfies

1. Reflexivity: For all x in S, xRx
2. Symmetry: For $x, y \in S$ $xRy \Leftrightarrow yRx$
3. Transitivity: For $x, y, z \in S$ $xRy \wedge yRz \Rightarrow xRz$

- Every equivalence relation on S partitions S into equivalence classes.
- The number of equivalence classes is called the index of the relation.
- An equivalence class containing x is written as $[x]$.

Right Invariant

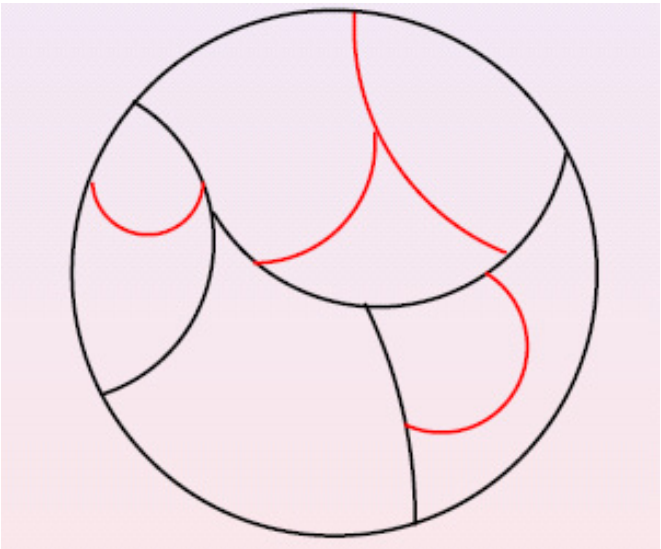
Definition 13

An equivalence relation on Σ^* is said to be right invariant with respect to concatenation if $\forall x, y \in \Sigma^*$ and $a \in \Sigma$, xRy implies that $xaRya$.

Refinement

Definition 14

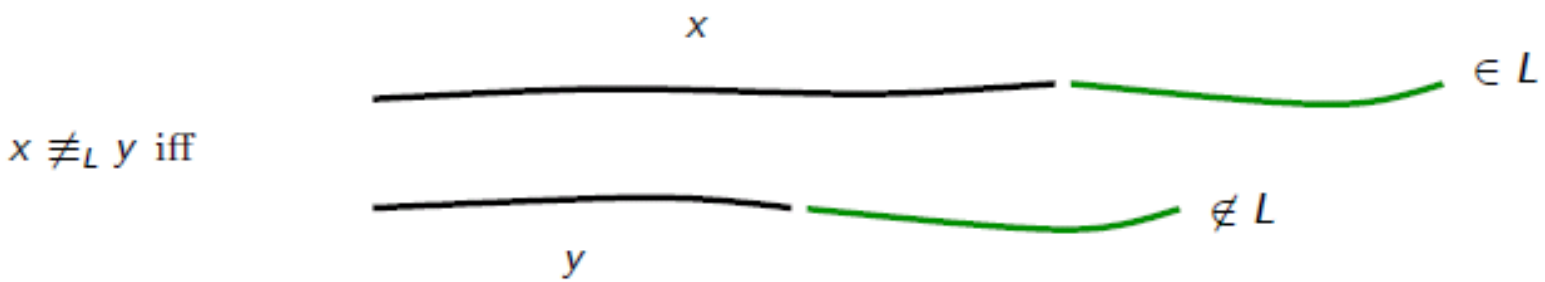
An equivalence relation R_1 is a refinement of R_2 if $R_1 \subseteq R_2$, i.e. $(x, y) \in R_1 \Rightarrow (x, y) \in R_2$



Myhill-Nerode Theorem – Algebraic View of Languages

- Equivalence relation $\equiv_L$ on A^* induced by $L \subseteq A^*$:

$$x \equiv_L y \Leftrightarrow (z \in A^*, xz \in L \Leftrightarrow yz \in L)$$

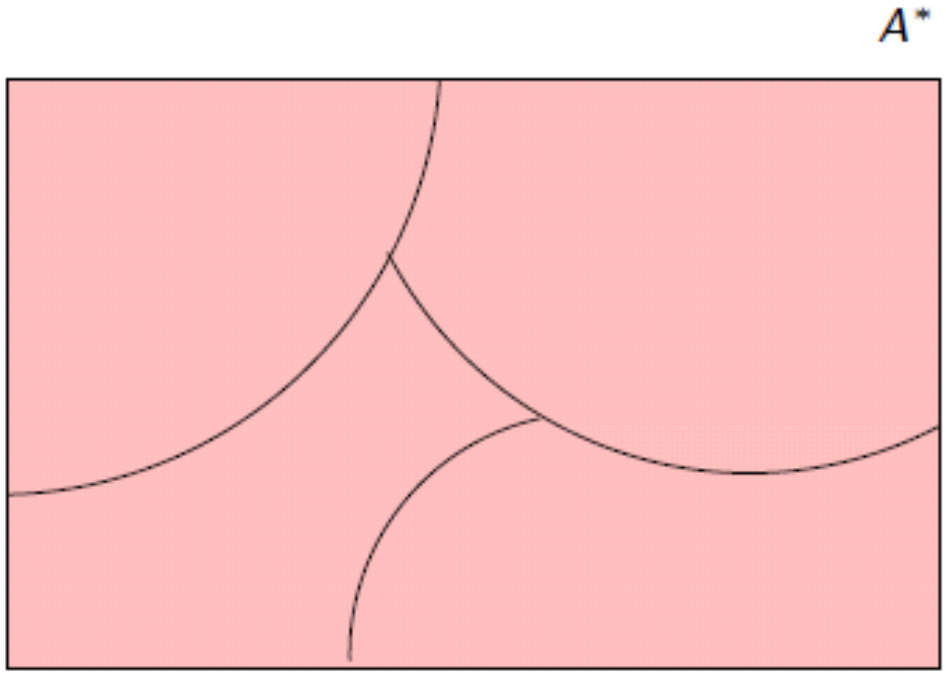


Theorem 15 (Myhill-Nerode)

L is regular iff $\equiv_L$ is of finite index (i.e., having a finite number of equivalence classes).

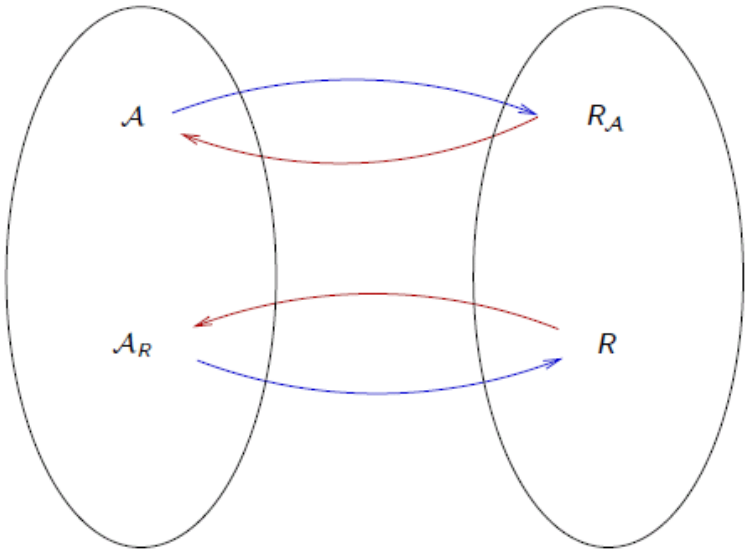
Myhill-Nerode Relation for a Language

- An MN relation for a language L on an alphabet A is an equivalence relation R on A^* satisfying
  - R is right-invariant (i.e. $xRy \Rightarrow xaRya, a \in A$)
- R respects L (i.e., x R y $\Rightarrow (x, y \in L)$ or$(x, y \notin L)$).



Deterministic Finite Automata for L and MN relations for L

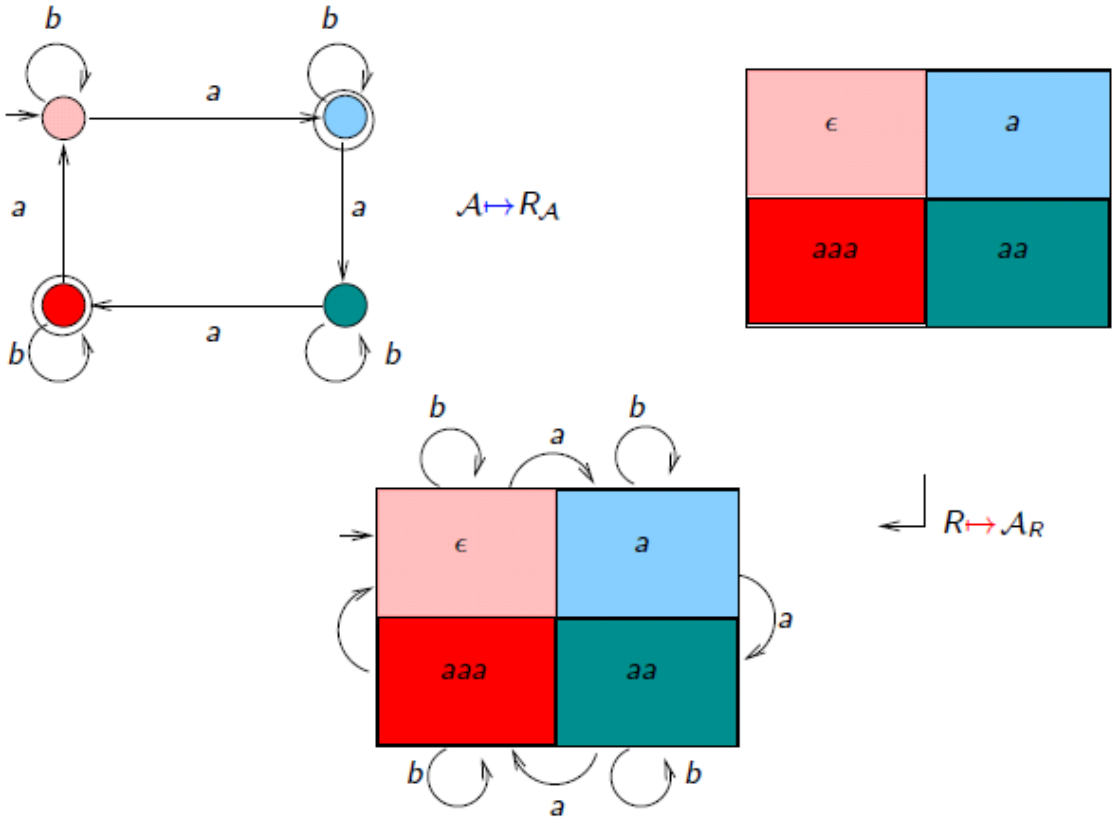
DFA (with no unreachable states) for L and MN relations for L are in 1-1 correspondence (they represent each other).



- Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
- Define a relation $R_{\mathcal{A}}$ as follows:
  - For $x, y \in \Sigma^*, xR_{\mathcal{A}}y \Leftrightarrow \delta(q_0, x) = \delta(q_0, y)$.

Example

L is "Odd number of a's":



$[\epsilon] = \{\epsilon, b, bb, aaaa, ...\}$; $[a] = \{a, ab, abb, ...\}$; $[aa] = \{aa, aab, aabb....\}$;
$[aaa] = \{aaa, aaab, aaabb, ...\}$.

Refinements of $\equiv_L$

Lemma 16

Let R be any MN-relation for a language L over A. Then R refines $\equiv_L$.

Proof.

To prove that xRy implies $x \equiv_L y$. Suppose $x \not\equiv_L y$. Then there exists z such that (WLOG) $xz \in L$ and $yz \notin L$. Suppose xRy. Since its an MN relation for L, it must be right invariant; and hence $xzRyz$. But this contradicts the assumption that R respects L. □

Theorem 17 (Myhill-Nerode)

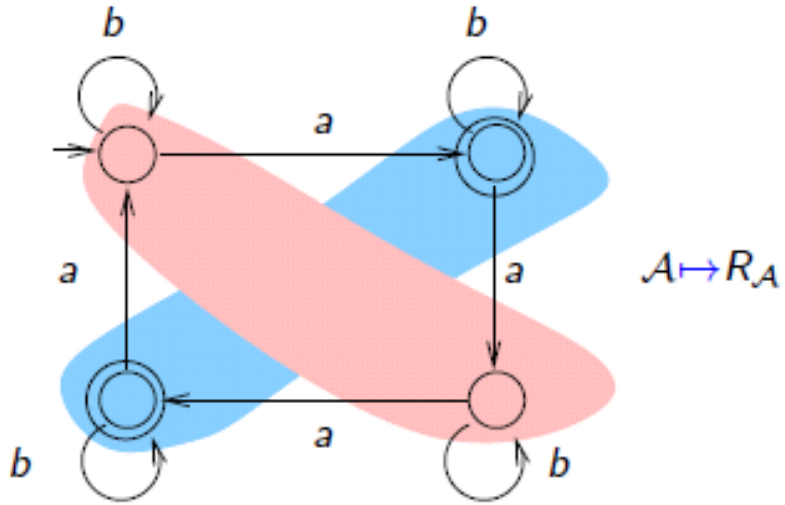
L is regular iff $\equiv_L$ is of finite index (i.e., having a finite number of equivalence classes).

Canonical DFA for L

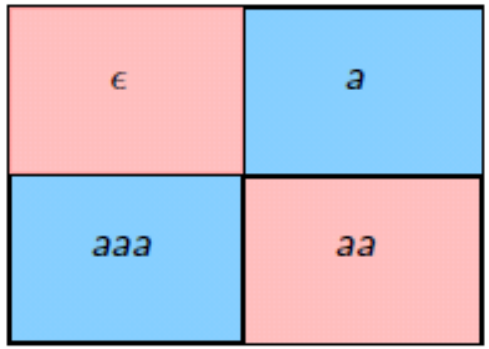
- We call $A_{\equiv_L}$ the "canonical" DFA for L in the following sense.
- Given a DFA $\mathcal{A} = (Q, A, q_0, \delta, F)$, let $\approx$ be an equivalence relation on Q (called a partitioning of $\mathcal{A}$) s.t.
  - $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a), \forall a \in A$,
  - $(p \approx q \text{ and } p \in F) \Rightarrow q \in F$
- The quotient automaton $\mathcal{A}/\approx$ is defined as $\mathcal{A}/\approx = (Q_\approx, A, [q_0], \delta_\approx, F_\approx)$, where
  - $Q_\approx = \{[q] \mid q \in Q\}$
  - $\delta_\approx([q], a) = [\delta(q, a)]$
  - $F_\approx = \{[q_f] \mid q_f \in F\}$
- DFA $\mathcal{A}$ is a refinement of a DFA $\mathcal{B}$ if there is a partition $\approx$ of $\mathcal{A}$, such that $\mathcal{A}/\approx$ is isomorphic to $\mathcal{B}$.
- Every other DFA for L is a "refinement" of $\mathcal{A}_{\equiv_L}$.

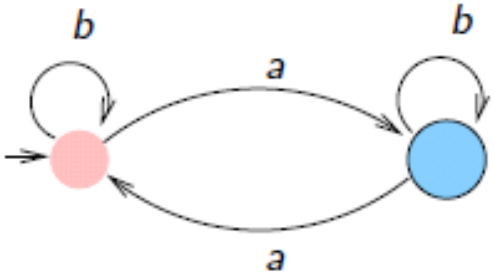
Canonicity of $A_{\equiv_L}$

Let $\mathcal{A}$ be a DFA for L with no unreachable states. Then $A_{\equiv_L}$ represents a partitioning of $\mathcal{A}$ (i.e., $\mathcal{A}$ refines $A_{\equiv_L}$).

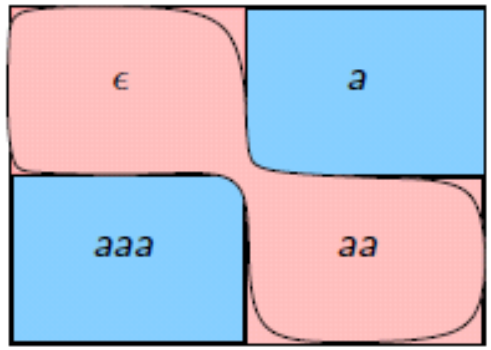


$\mathcal{A} \mapsto R_{\mathcal{A}}$

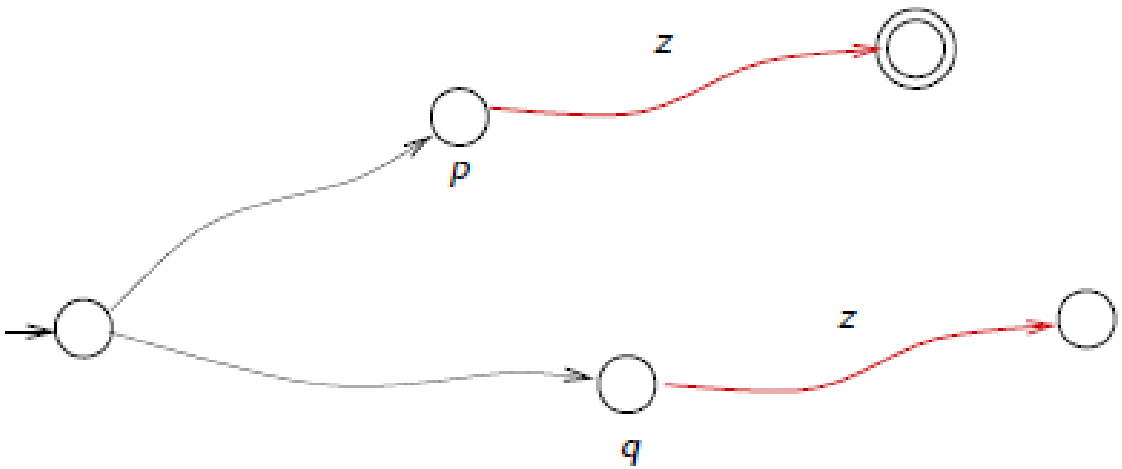




$\mathcal{A}_{\equiv_L} \mapsto \equiv_L$



- Let $\mathcal{A} = (Q, A, q_0, \delta, F)$ be a DFA for L with no unreach. states.
- The canonical MN relation for L (i.e. $\equiv_L$) induces a "coarsest" partitioning $\approx_L$ of $\mathcal{A}$ given by
  - $p \approx_L q$ iff $\exists x, y \in A^*$, such that $\delta(q_0, x) = p, \delta(q_0, y) = q$ with $x \equiv_L y$, or equivalently,
  - $p \approx_L q$ iff $\forall z \in A^*, \delta(p, z) \in F$ iff $\delta(q, z) \in F$

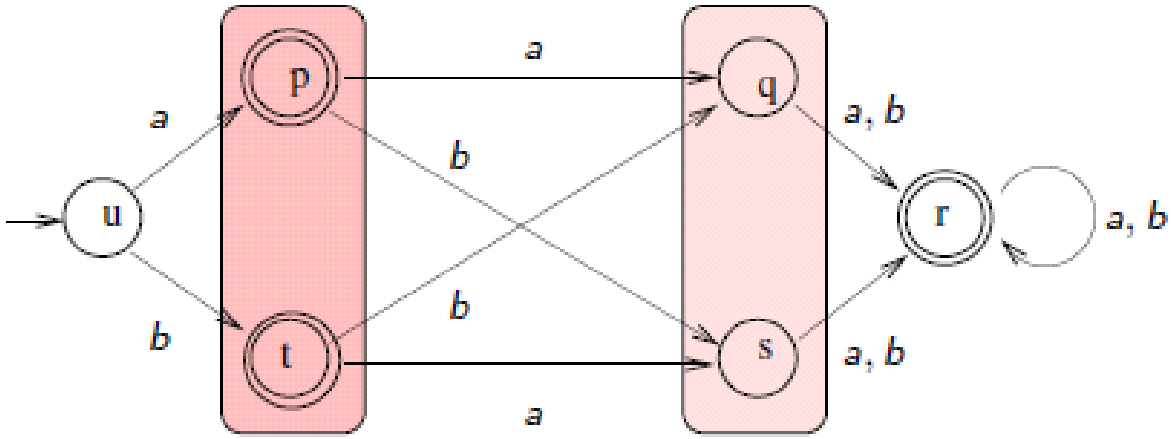


Algorithm for Computing $\approx_L$ for a DFA $\mathcal{A}$

- Input: DFA $\mathcal{A} = (Q, A, q_0, \delta, F)$
- Output $\approx_L$ for $\mathcal{A}$
  1. Initialize entry for each pair in table to "unmarked".
  2. Mark (p, q) if $p \in F$ and $q \notin F$ or vice-versa.
  3. Scan table entries and repeat till no more marks can be added:
     - ★ If there exists unmarked (p, q) with $a \in A$ such that $\delta(p, a)$ and $\delta(q, a)$ are marked, then mark (p, q).
  4. Return $\approx_L$ as: $p \approx_L q$ iff (p, q) is left unmarked in table.

Example

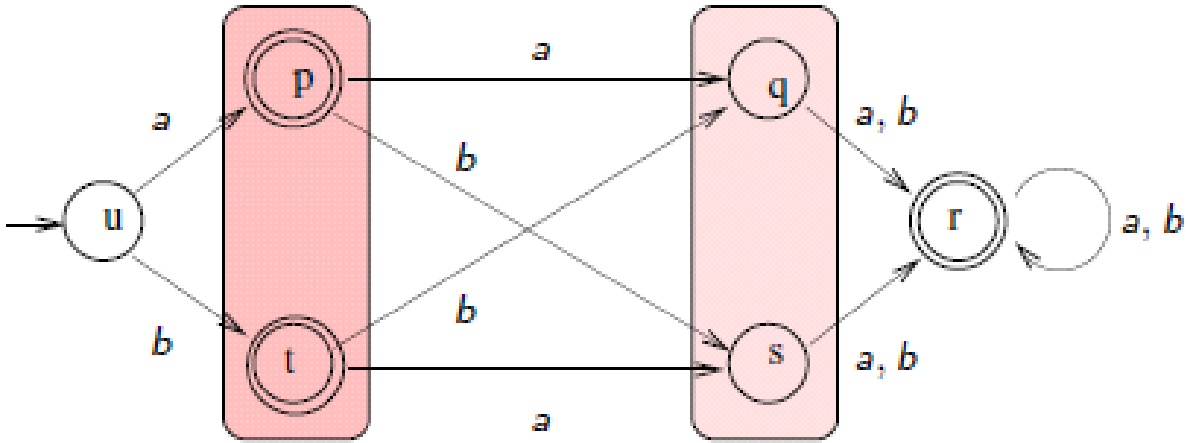
Run minimization algorithm on DFA below:



	u	p	t	q	s	r
u	·					
p		·				
t			·			
q				·		
s					·	
r						·

Example (Cont'd)

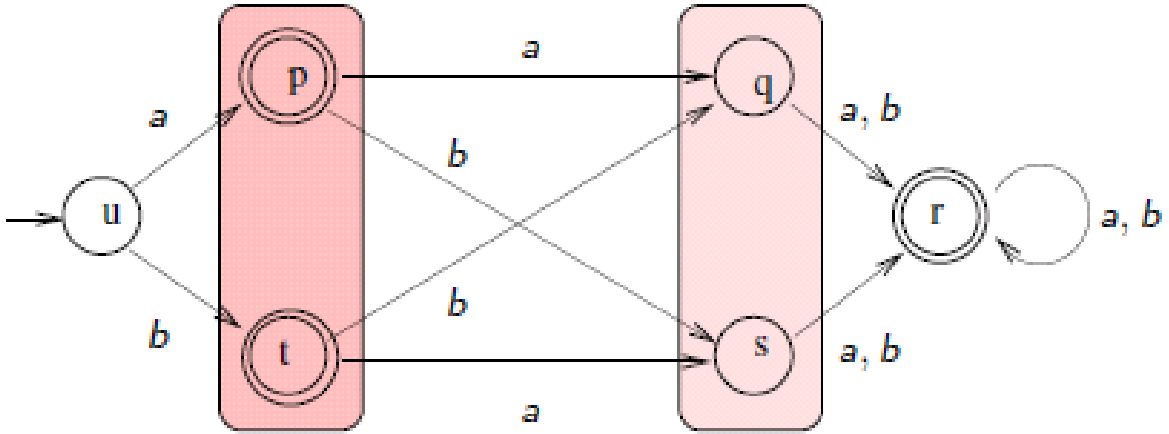
Run minimization algorithm on DFA below:



	u	p	t	q	s	r
u	·					
p	√	·				
t	√		·			
q		√	√	·		
s		√	√		·	
r	√			√	√	·

Example (Cont'd)

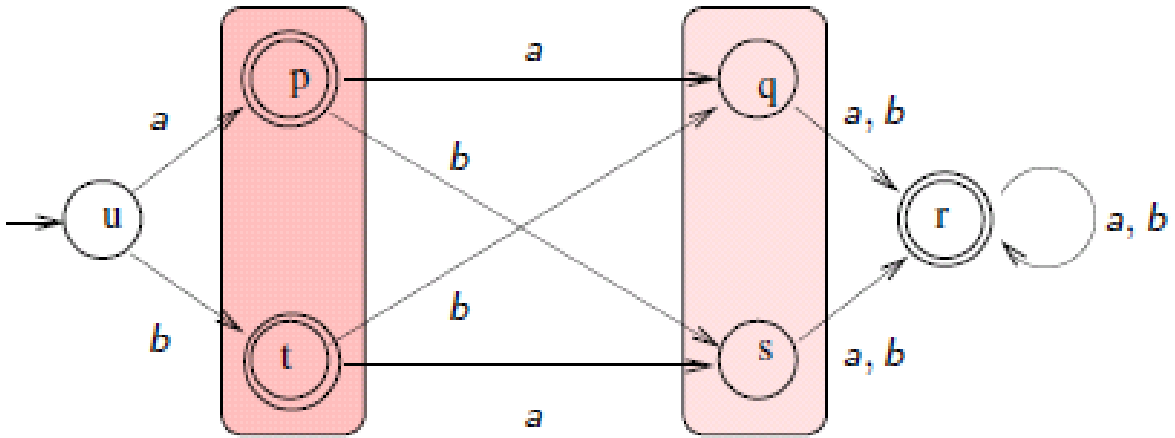
Run minimization algorithm on DFA below:



	u	p	t	q	s	r
u	·					
p	√	·				
t	√		·			
q		√	√	·		
s		√	√		·	
r	√	√	√	√	√	·

Example (Cont'd)

Run minimization algorithm on DFA below:



	u	p	t	q	s	r
u	·					
p	√	·				
t	√		·			
q	√	√	√	·		
s	√	√	√		·	
r	√	√	√	√	√	·

Applications of the Myhill-Nerode Theorem

The MN theorem can be used to show that a particular language is regular without actually constructing the automaton or to show conclusively that a language is not regular.
Example. Is the following language regular

1. $L_1 = \{xy : \ |x| = |y|, x, y \in \Sigma^*\}$?
2. Example. What about the language
   $L_2 = \{xy : \ |x| = |y|, x, y \in \Sigma^*$ and y ends with a 1 $\}$?
3. Example. What about the language
   $L_3 = \{xy : \ |x| = |y|, x, y \in \Sigma^*$ and y contains a 1$\}$?

Applications of the Myhill-Nerode Theorem (cont'd)

1. For the language L_1 there are two equivalence classes of R_{L_1} . The first C_1 contains all strings of even length and the second C_2 all strings of odd length.
2. For L_2 we have the additional constraint that y ends with a 1. Class C_2 remains the same as that for L_1. Class C_1 is refined into classes C_1' which contains all strings of even length that end in a 1 and C_1'' which contains all strings of even length which end in a 0. Thus L_1 and L_2 are both regular.
3. For L_3 we have to distinguish for example, between the even length strings in the sequence 01, 0001, 000001,..., as 00 distinguishes the first string from all the others after it in the sequence (concatenation of 00 to 01 gives a string not in L_3 but concatenation of 00 to all the others gives a string in L_3), 0000 distinguishes the second from all the others ...

The $\equiv_L$ for $L = \{a^n b^n \ n \geq 0\}$

Describe the equivalence classes of $\equiv_L$ for $L = \{a^n b^n \ n \geq 0\}$