Bab 4 Properties of Regular Languages

27-29 September 2021

General Discussion of "Properties"

The Pumping Lemma

Closure Properties

Decision Properties: Emptiness, Membership

Equivalence and Minimization of Automata

Informatika

Properties of Language Classes

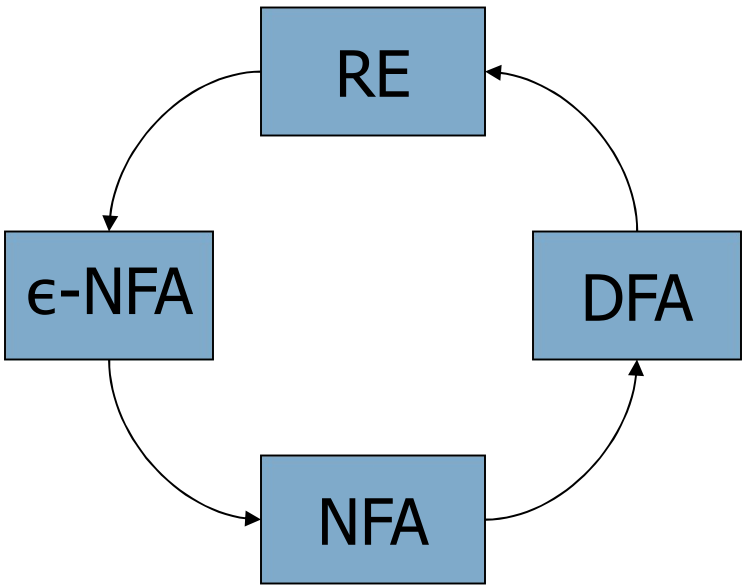
- **A *language class* is a set of languages.**
  - We have one example: the regular languages.
  - We'll see many more in this class.
- **Language classes have two important kinds of properties:**
  1. Decision properties.
  2. Closure properties.

Representation of Languages

- **Representations can be formal or informal.**
- **Example (formal): represent a language by a RE or DFA defining it.**
- **Example: (informal): a logical or prose statement about its strings:**
  - $\{0^n1^n \mid n \text{ is a nonnegative integer}\}$
  - "The set of strings consisting of some number of 0's followed by the same number of 1's."

What if the Regular Language Is not Represented by a DFA?

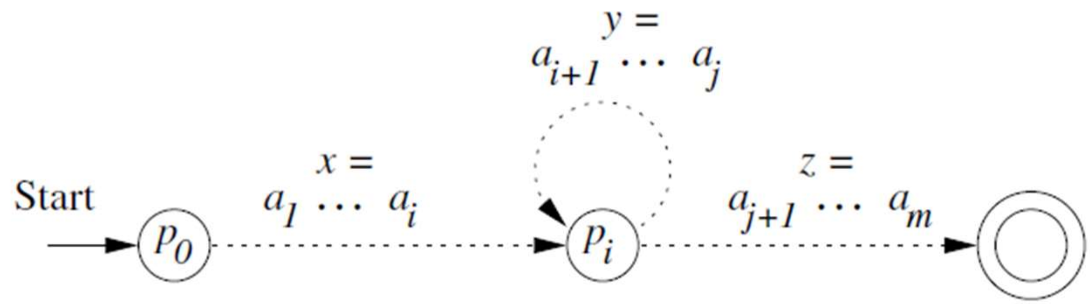
- **There is a circle of conversions from one form to another:**



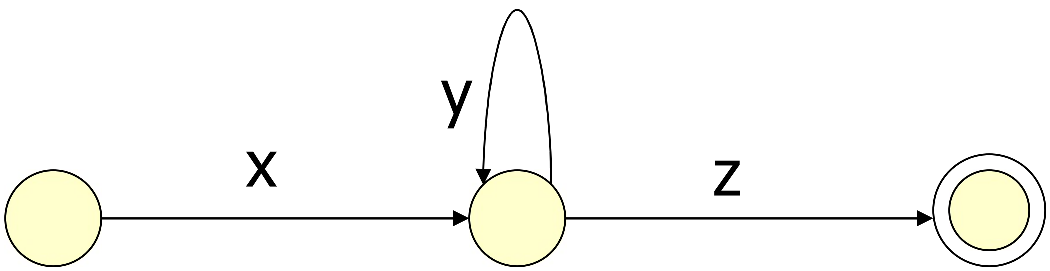
Pumping Lemma

- **If an n-state DFA accepts a string w of length *n* or more, then there must be a state that appears twice on the path labeled w from the start state to a final state.**
- **Because there are at least n+1 states along the path.**
- **Every regular language satisfies the pumping lemma.**

Pumping Lemma (2)



w = xyz

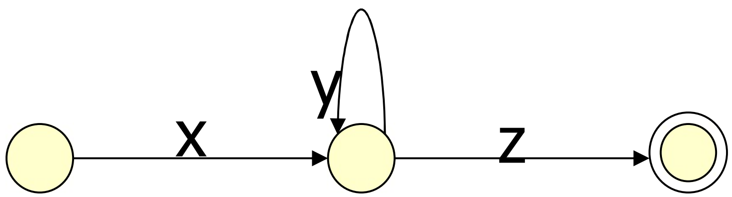


Then xy^iz is in the language for all $i \geq 0$.

Since y is not ϵ, we see an infinite number of strings in L.

Pumping Lemma Key Idea

- Remember:



- We can choose y to be the first cycle on the path.
- So $|xy| \leq n$; in particular, $1 \leq |y| \leq n$.
- Thus, if w is of length 2n or more, there is a shorter string in L that is still of length at least n.
- Keep shortening to reach [n, 2n-1].

The Pumping Lemma

- **Every regular language satisfies the pumping lemma.**
- **We have, almost accidentally, proved a statement that is quite useful for showing certain languages are not regular.**
- **Called the *pumping lemma for regular languages*.**

Statement of the Pumping Lemma

Number of states of DFA for L

For every regular language L

There is an integer n, such that

For every string w in L of length $\geq$ n

We can write w = xyz such that:

Labels along first cycle on path labeled w

1. **$|xy| \leq n$.**
2. **$|y| > 0$.**
3. **For all $i \geq 0$, xy^iz is in L.**

Example: Use of Pumping Lemma

- We have claimed $\{0^k1^k \mid k \geq 1\}$ is not a regular language.
- Proof by contradiction
- Suppose it were. Then there would be an associated n for the pumping lemma.
- Let $w = 0^n1^n$. We can write $w = xyz$, where x and y consist of 0's, and $y \neq \epsilon$.
- But then xyyz would be in L, and this string has more 0's than 1's.

Contoh Soal $L = \{wtw \mid w, t \in \{0,1\}^+\}$

To prove that L is not a regular language, we will use a proof by contradiction. Assume that L is a regular language. Then by the Pumping Lemma for Regular Languages, there exists a pumping length p for L such that for any sring $s \in L$ where $|s| \geq p$, $s = xyz$ subject to the following conditions:

(a) $|y| > 0$

(b) $|xy| \leq p$, and

(c) $\forall i > 0, xy^i z \in L$.

Choose $s = 0^p110^p1$. Clearly $s \in L$ with $w = 0^p1$ and $t = 1$, and $|s| \geq p$. By condition (b), it is obvious that xy is composed only of zeros, and further, by (a) and (b), it follows that $y = 0^k$ for some $k > 0$. By condition (c), we can take any i and xy^iz will be in L. Taking $i = 2$, then $xy^2z \in L$. $xy^2z = xyyz = 0^{(p+k)}110^p1$. There is no way that this string can be divided into wtw as required to be in L, thus $xy^2z \notin L$. This is a contradiction with condition (c) of the pumping lemma. Therefore the assumption that L is a regular language is incorrect and thus L is not a regular language.

Contoh

- **Perlihatkan dengan Pumping Lemma**

L_{pr} = { 1^p | p = prima}

Jawab: Ide pembuktian dengan kontradiksi.

Bilangan prima adalah bilangan yg merupakan perkalian 1 dan bilangan tersebut.

Suppose $L_{pr} = \{1^p : p \text{ is prime }\}$ were regular.

Let n be given by the pumping lemma.

Choose a prime $p \geq n + 2$.

$$w = \overbrace{\underbrace{111\cdots}_{x}\underbrace{\cdots\cdot 1}_{\substack{y \\ |y|=m}}\underbrace{1111\cdots 11}_{z}}^{p}$$

For i = p-m then xy^iz:

Now $xy^{p-m}z \in L_{pr}$

$|xy^{p-m}z| = |xz| + (p-m)|y| =$
$p - m + (p-m)m = (1+m)(p-m)$
which is not prime unless one of the factors is 1.

- $y \neq \epsilon \Rightarrow 1 + m > 1$

- $m = |y| \leq |xy| \leq n, \;\; p \geq n + 2$
$\Rightarrow p - m \geq n + 2 - n = 2.$

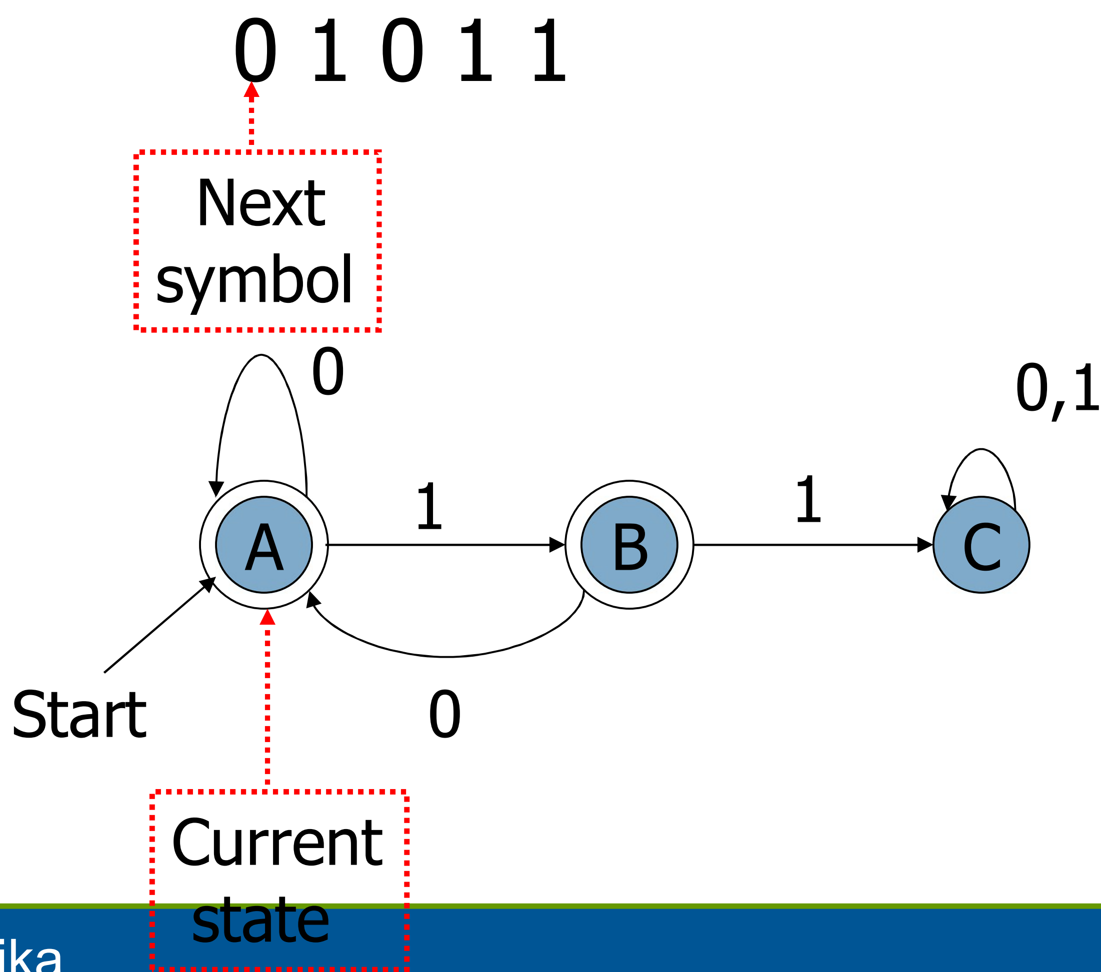
Decision Properties

- 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?
- Diketahui regular languages L dan M, apakah L = M?
- Diketahui regular languages L dan M, apakah $L \subseteq M$?

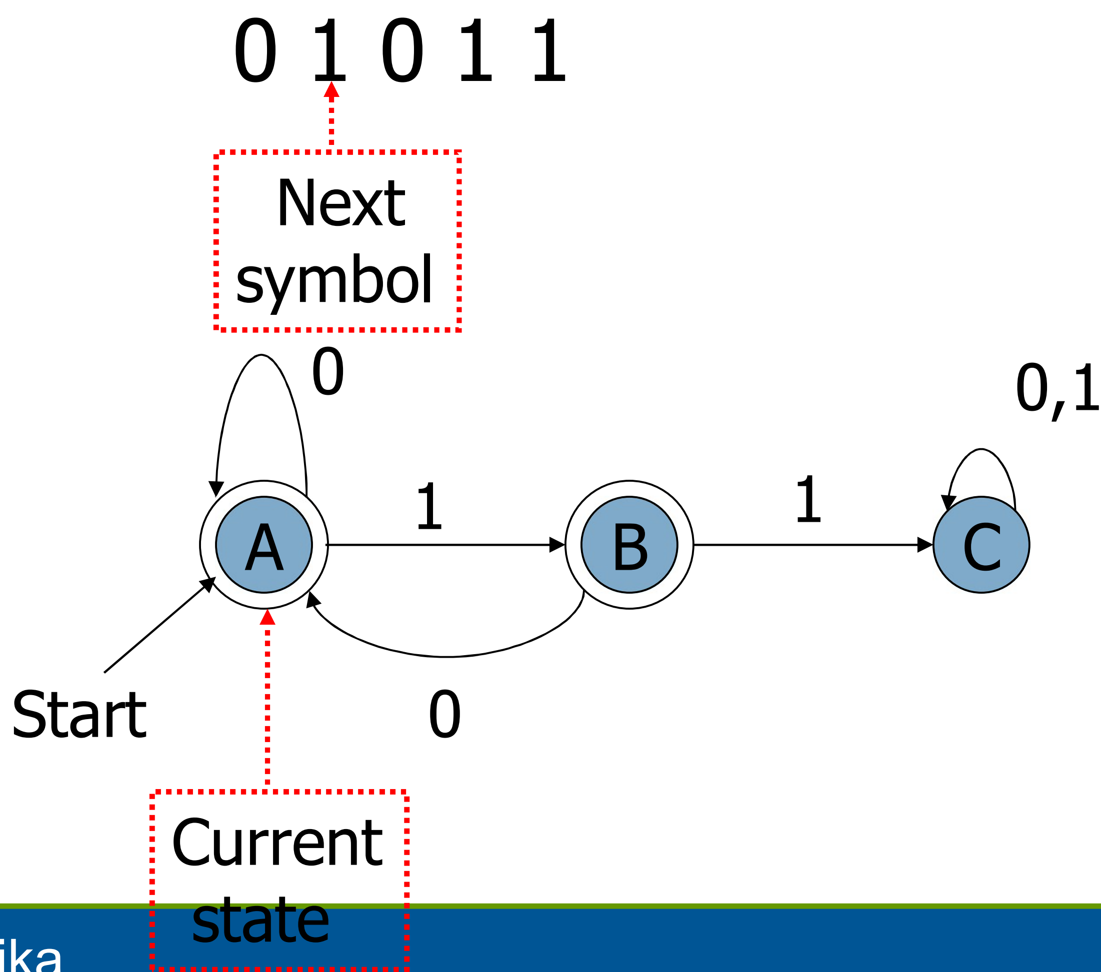
The Membership Question

- 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.

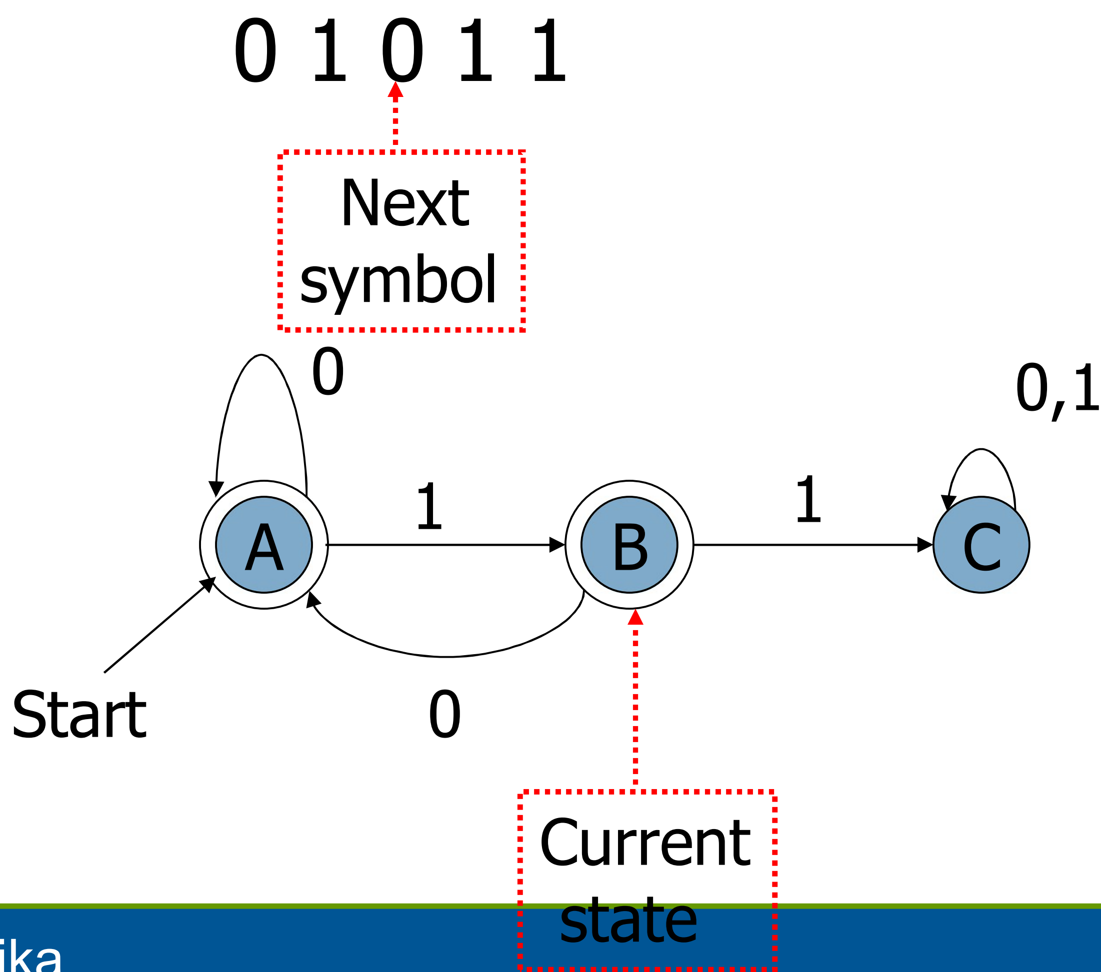
Example: Testing Membership



Example: Testing Membership

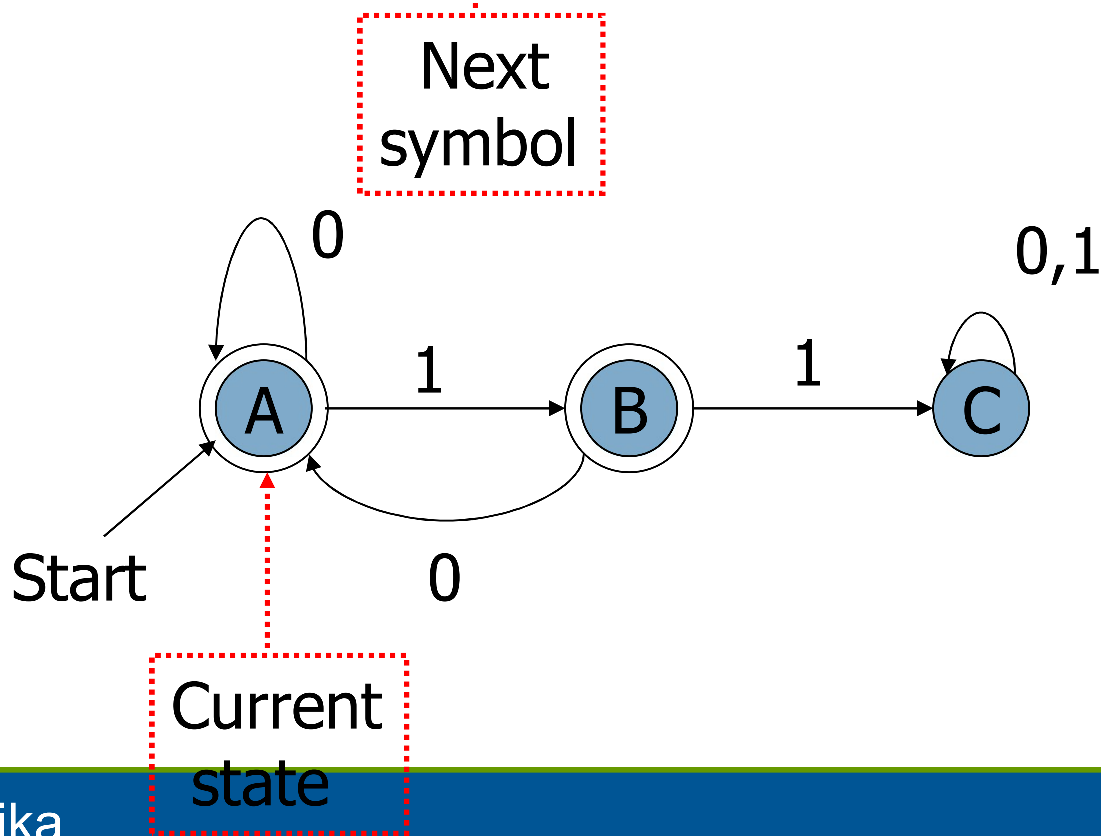


Example: Testing Membership

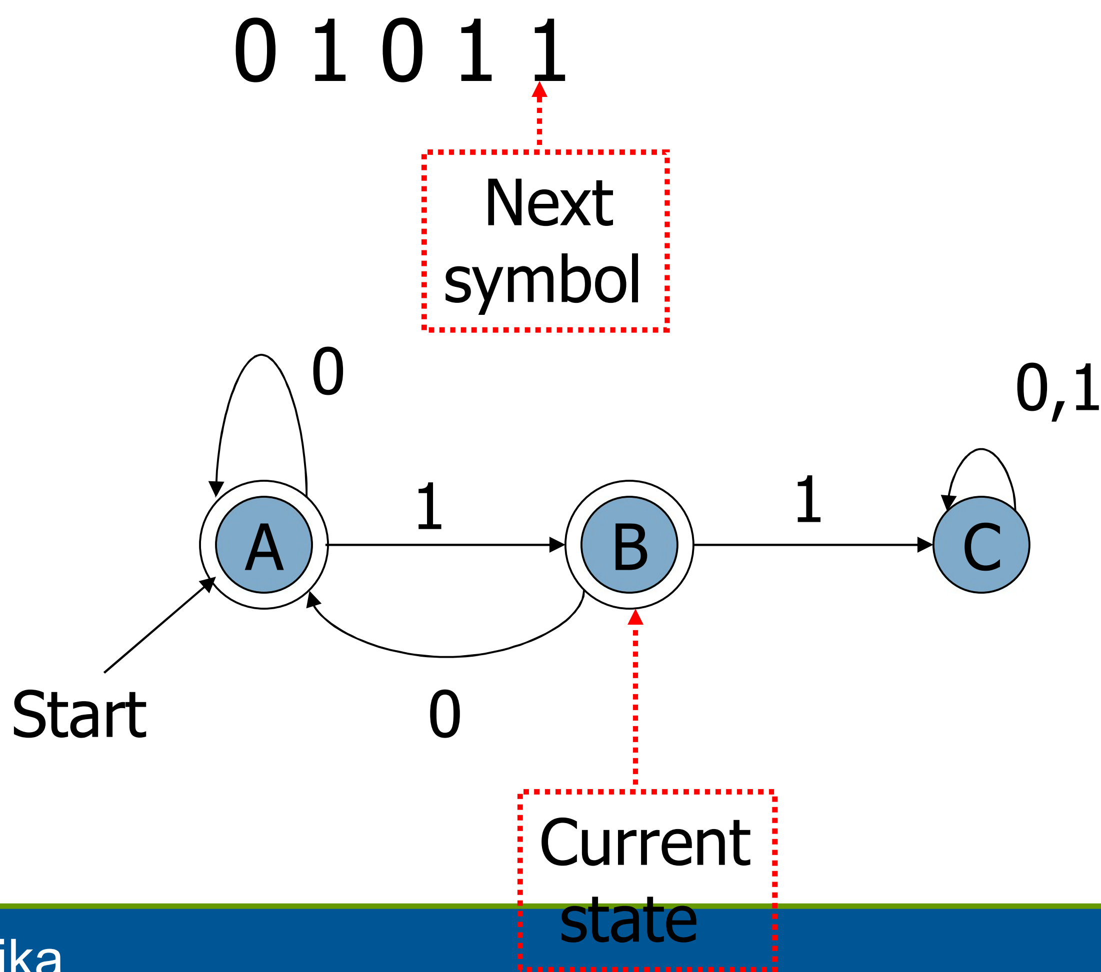


Example: Testing Membership

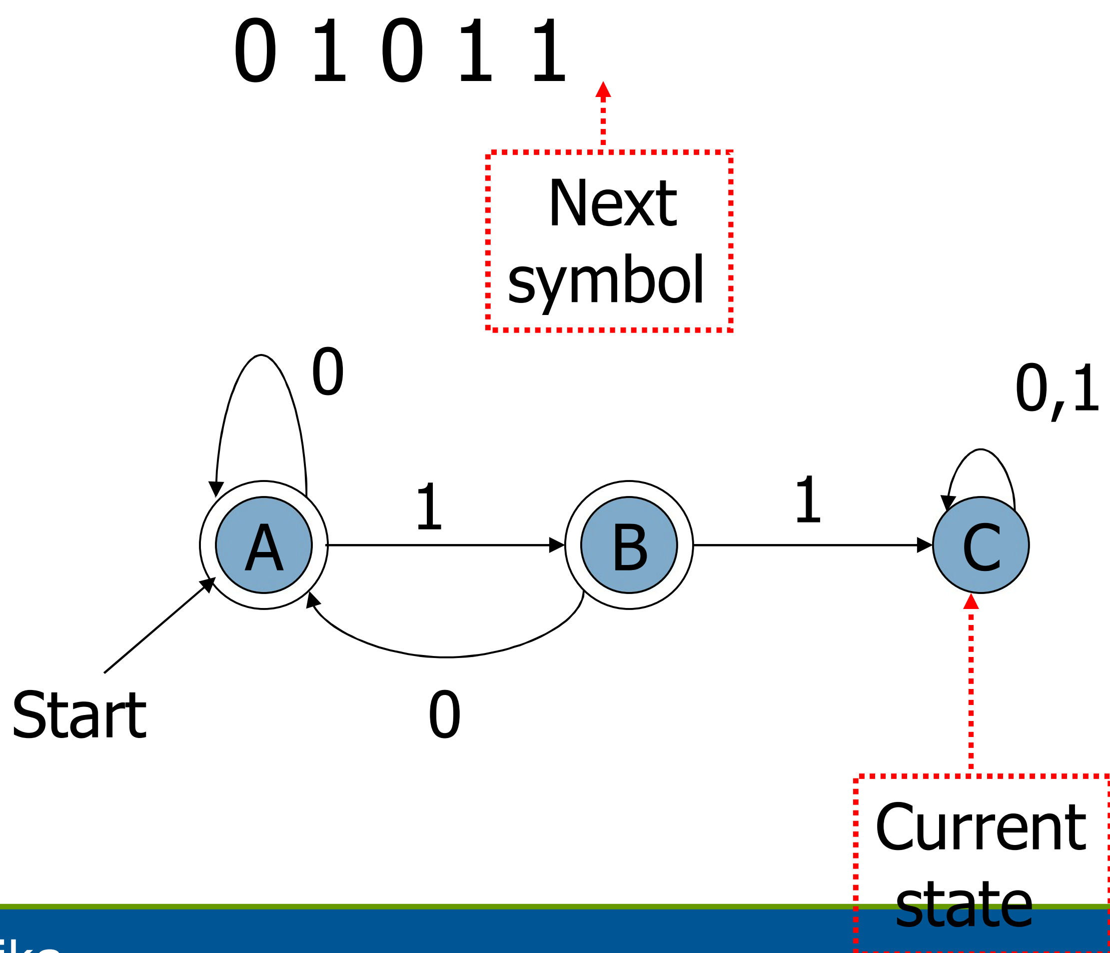
0 1 0 1 1



Example: Testing Membership



Example: Testing Membership



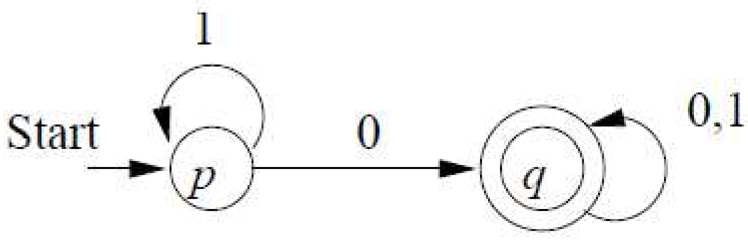
The Emptiness Problem

- 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.

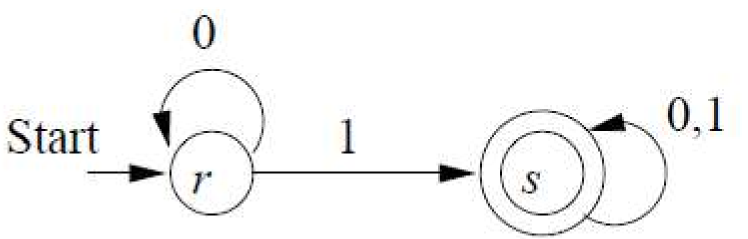
Teknik Untuk Memeriksa Kesamaan 2 Finite Automata

- **Teknik Perkalian Finite Automata**
  - Buatlah sebuah FA baru yang merupakan hasil perkalian 2 FA lama
  - Tetapkan state akhir pada FA baru yaitu state baru merupakan gabungan dari 2 state lama dimana hanya salah satu state pada FA lama merupakan state akhir. State final merepresentasikan status perbedaan string diterima dari 2 FA lama tsb
  - Hubungkan setiap state (tetapkan transition function) pada FA baru berdasar transition function FA lama
  - Jika pada FA baru, tidak ada string yang bisa diterima, maka dapat dinyatakan bahwa 2 FA tsb adalah sama

Teknik dgn Perkalian FA

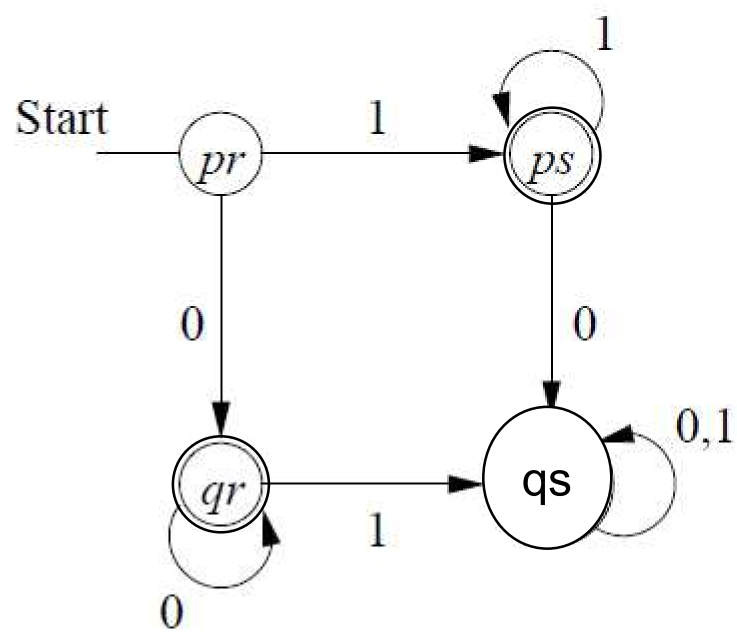


(a)



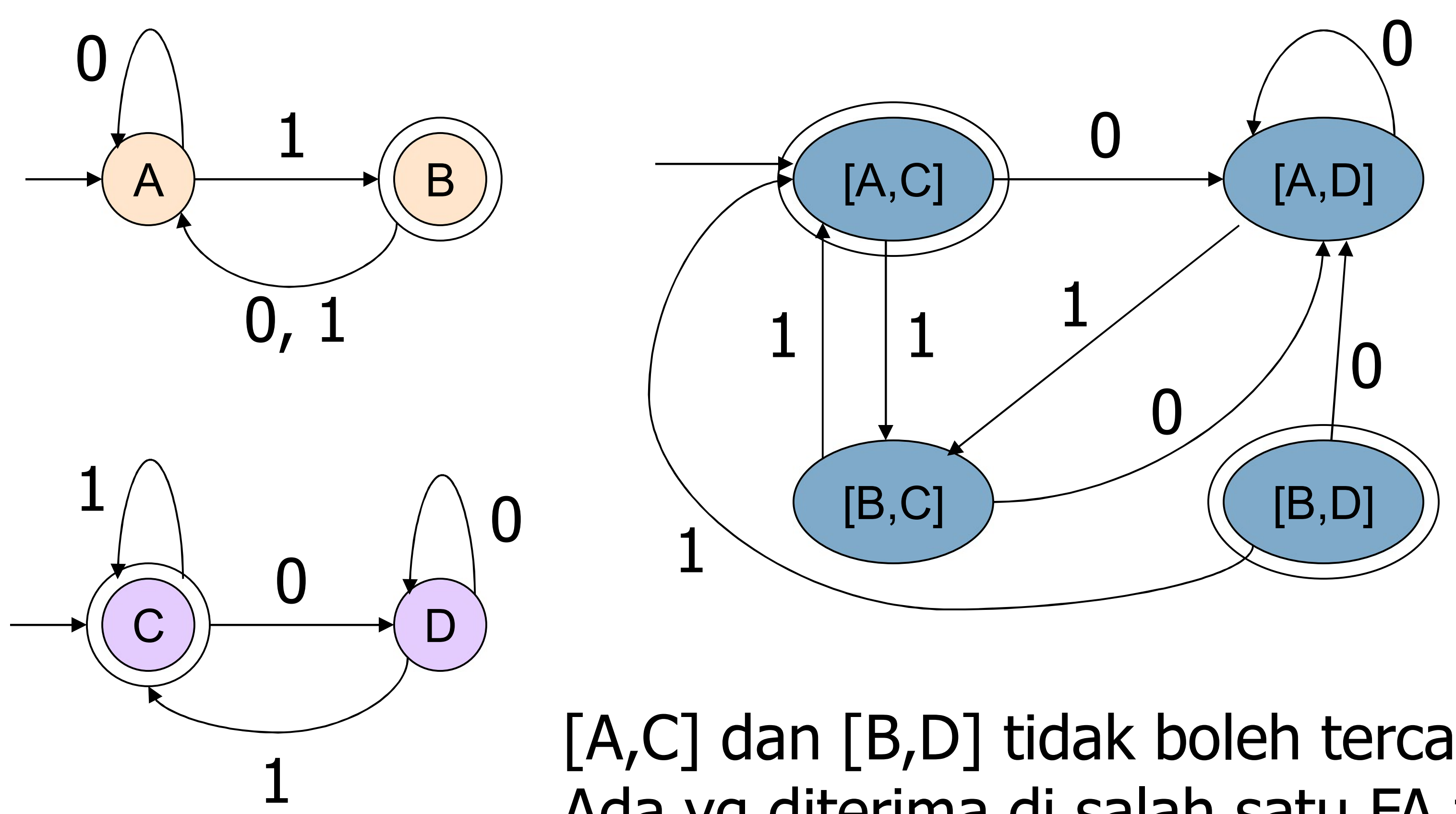
(b)

- **Hasil perkalian:**



- **Karena FA hasil perkalian di atas dapat menerima string (cth: 1,11,0,00,dst; dgn final state berupa ps dan qr) maka 2 FA ini tidak ekuivalen**

Product DFA utk Equivalence

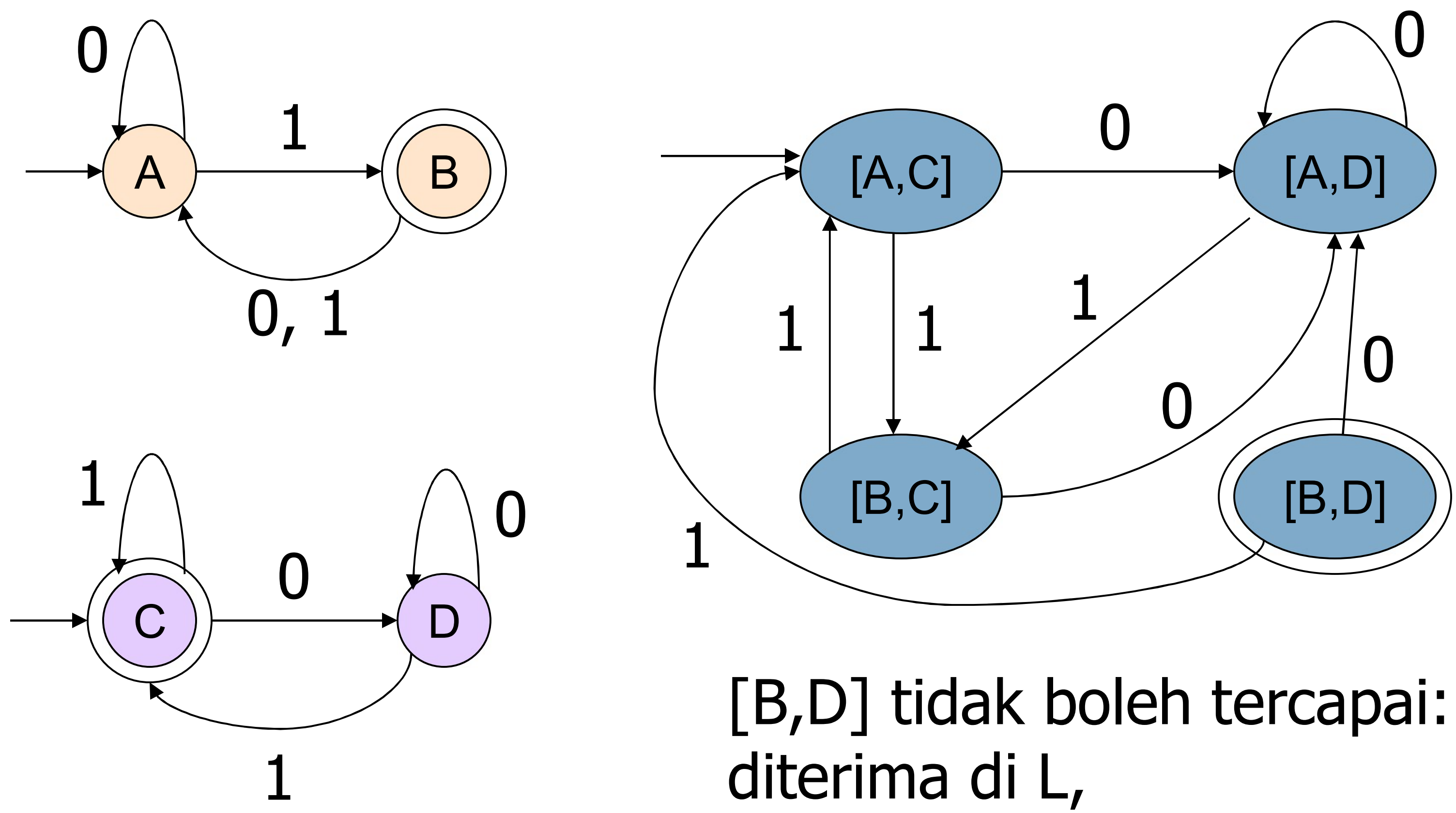


[A,C] dan [B,D] tidak boleh tercapai: Ada yg diterima di salah satu FA tp tdk diterima di FA lainnya

Decision Property: Containment

- **Diketahui regular languages L dan M, apakah L $\subseteq$ M?**
- **Menggunakan product automaton**

Product DFA utk Containment



[B,D] tidak boleh tercapai:
diterima di L,
tp tdk diterima di M

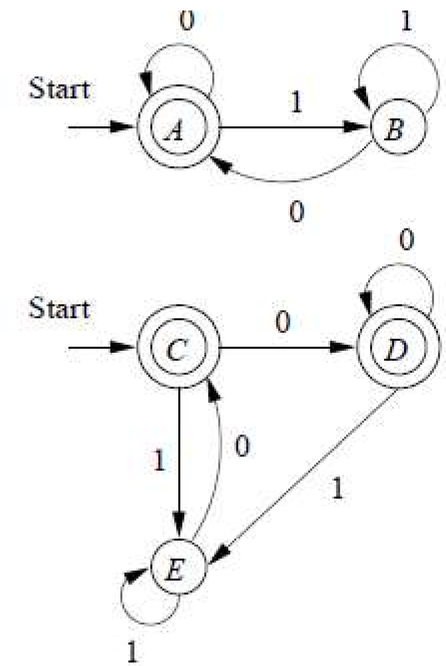
Teknik Untuk Memeriksa Kesamaan 2 Finite Automata

- **Teknik Minimisasi State pada Finite Automata dgn Distinguishable Table (Table Filling Algorithm)**
  - Buatlah sebuah *distinguishable table* yang jumlah kolom dan barisnya mewakili gabungan semua state pada 2 FA yang lama.
  - Jika state awal dan state akhir pada 2 FA bernilai indistinguishable (tidak bisa dibedakan) maka 2 FA merupakan FA yang sama.
  - Untuk mengisi x pada table filling sbb

Basis: If $p \in F$ and $q \notin F$, then $p \not\equiv q$.

Induction: If $\exists a \in \Sigma : \delta(p,a) \not\equiv \delta(q,a)$, then $p \not\equiv q$.

Teknik dgn Distinguishable Table (Table Filling Algorithm)



B	x			
C		x		
D		x		
E	x		x	x
	A	B	C	D

- **Karena start state tidak bisa dibedakan maka 2 FA ini adalah sama**

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.**

Constructing the Minimum-State DFA

- **Suppose $q_1,\ldots,q_k$ are indistinguishable states.**
- **Replace them by one state q.**
- **Then $\delta(q_1, a),\ldots, \delta(q_k, a)$ are all indistinguishable states.**
  - Key point: otherwise, we should have marked at least one more pair.
- **Let $\delta(q, a)$ = the representative state for that group.**

Example: State Minimization

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}	{2,4,6,8}	{1,3,5,7,9}

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

Here it is with more convenient state names

Remember this DFA? It was constructed for the chessboard NFA by the subset construction.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X				
B	X	X				
C	X	X				
D	X	X				
E	X	X				
F						

Start with marks for the pairs with one of the final states F or G.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X				
B	X	X				
C	X	X				
D	X	X				
E	X	X				
F						

Input r gives no help, because the pair [B, D] is not marked.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X	X	X	X	
B	X	X	X	X	X	
C	X	X				
D	X	X				
E	X	X				
F	X					

But input b distinguishes {A,B,F} from {C,D,E,G}. For example, [A, C] gets marked because [C, F] is marked.

Example – Continued

	r	b
→ A	B	C
B	D	E
C	D	F
D	D	G
E	D	G
* F	D	C
* G	D	G

	G	F	E	D	C	B
A	x	x	x	x	x	
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

[C, D] and [C, E] are marked because of transitions on b to marked pair [F, G].

Example – Continued

	r	b
→ A	B	C
B	D	E
C	D	F
D	D	G
E	D	G
* F	D	C
* G	D	G

	G	F	E	D	C	B
A	x	x	x	x	x	x
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

[A, B] is marked because of transitions on r to marked pair [B, D].

[D, E] can never be marked, because on both inputs they go to the same state.

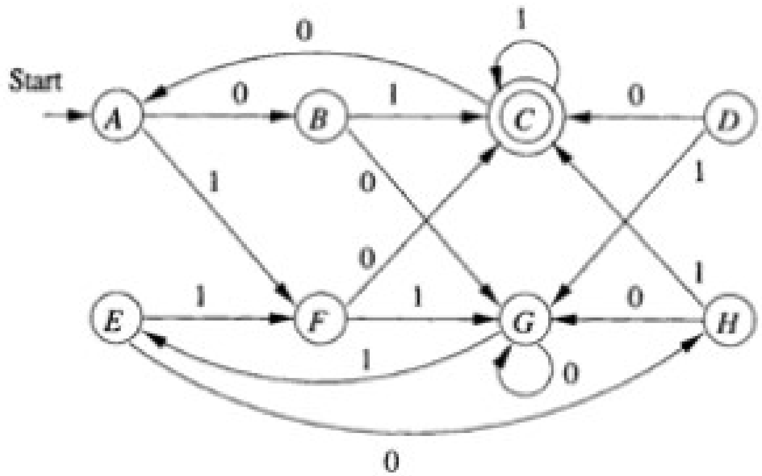
Example – Concluded

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

		r	b
→	A	B	C
	B	H	H
	C	H	F
	H	H	G
*	F	H	C
*	G	H	G

	G	F	E	D	C	B
A	x	x	x	x	x	x
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

Replace D and E by H.
Result is the minimum-state DFA.

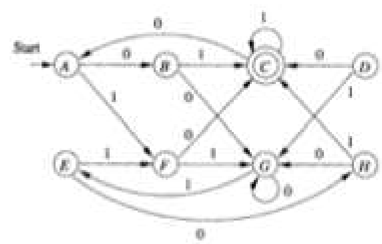


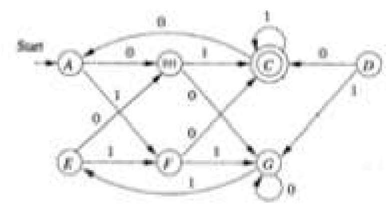
Distinguishable table

B							
C							
D							
E							
F							
G							
H							
	A	B	C	D	E	F	G

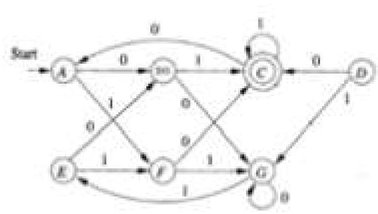
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

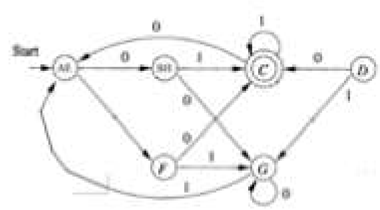
- Combine H and B



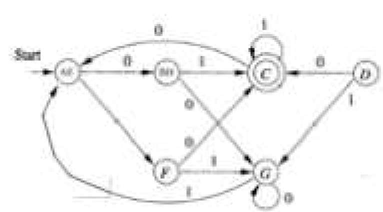


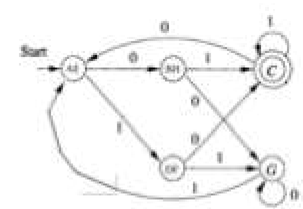
Combine E and A



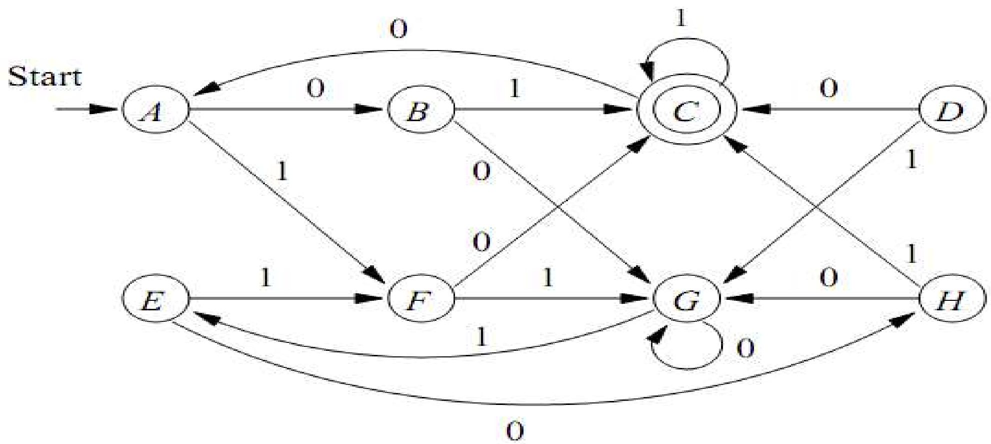


Combine D and F





What about A and E?



$\widehat{\delta}(A, \epsilon) = A \notin F, \widehat{\delta}(E, \epsilon) = E \notin F$

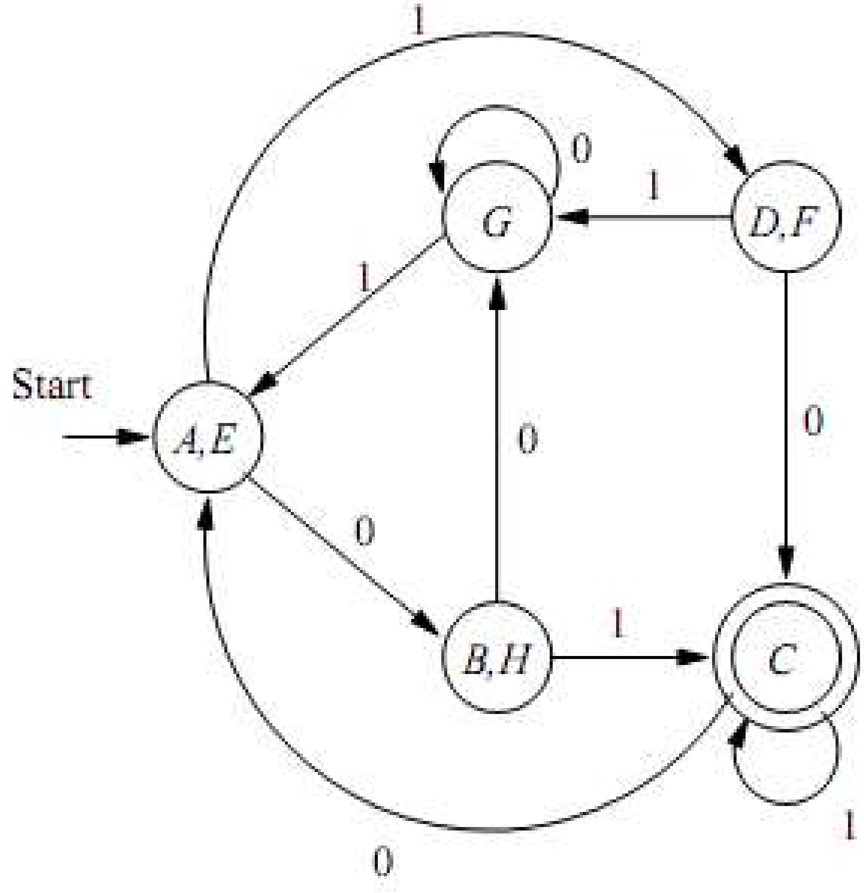
$\widehat{\delta}(A, 1) = F = \widehat{\delta}(E, 1)$

Therefore $\widehat{\delta}(A, 1x) = \widehat{\delta}(E, 1x) = \widehat{\delta}(F, x)$

$\widehat{\delta}(A, 00) = G = \widehat{\delta}(E, 00)$

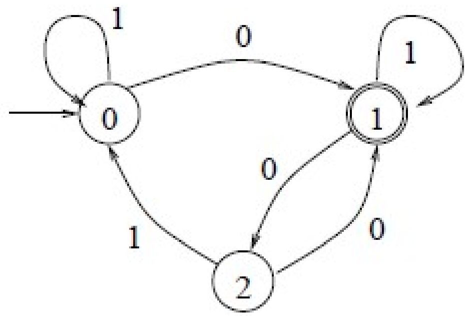
$\widehat{\delta}(A, 01) = C = \widehat{\delta}(E, 01)$

Conclusion: $A \equiv E$

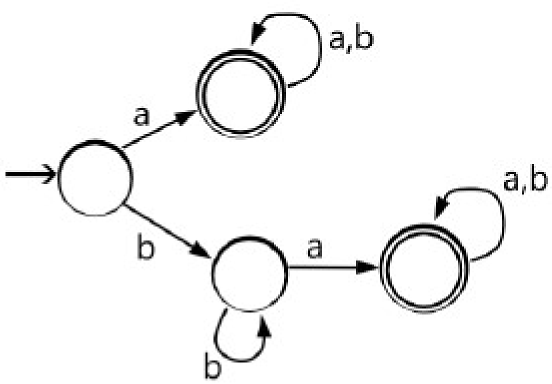

Start
A,E
G
D,F
B,H
C
1
0
1
1
0
0
0
1
1
0

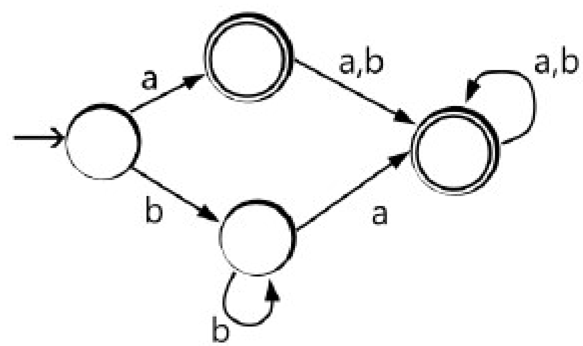
Soal Latihan

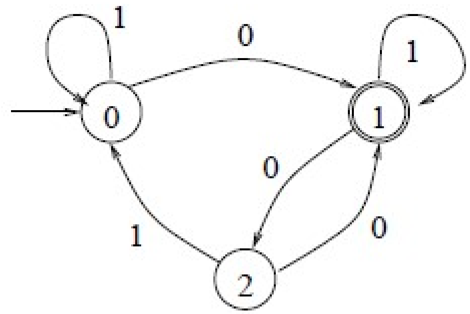
- **Minimalkan state pada FA**

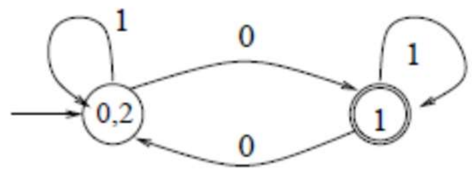


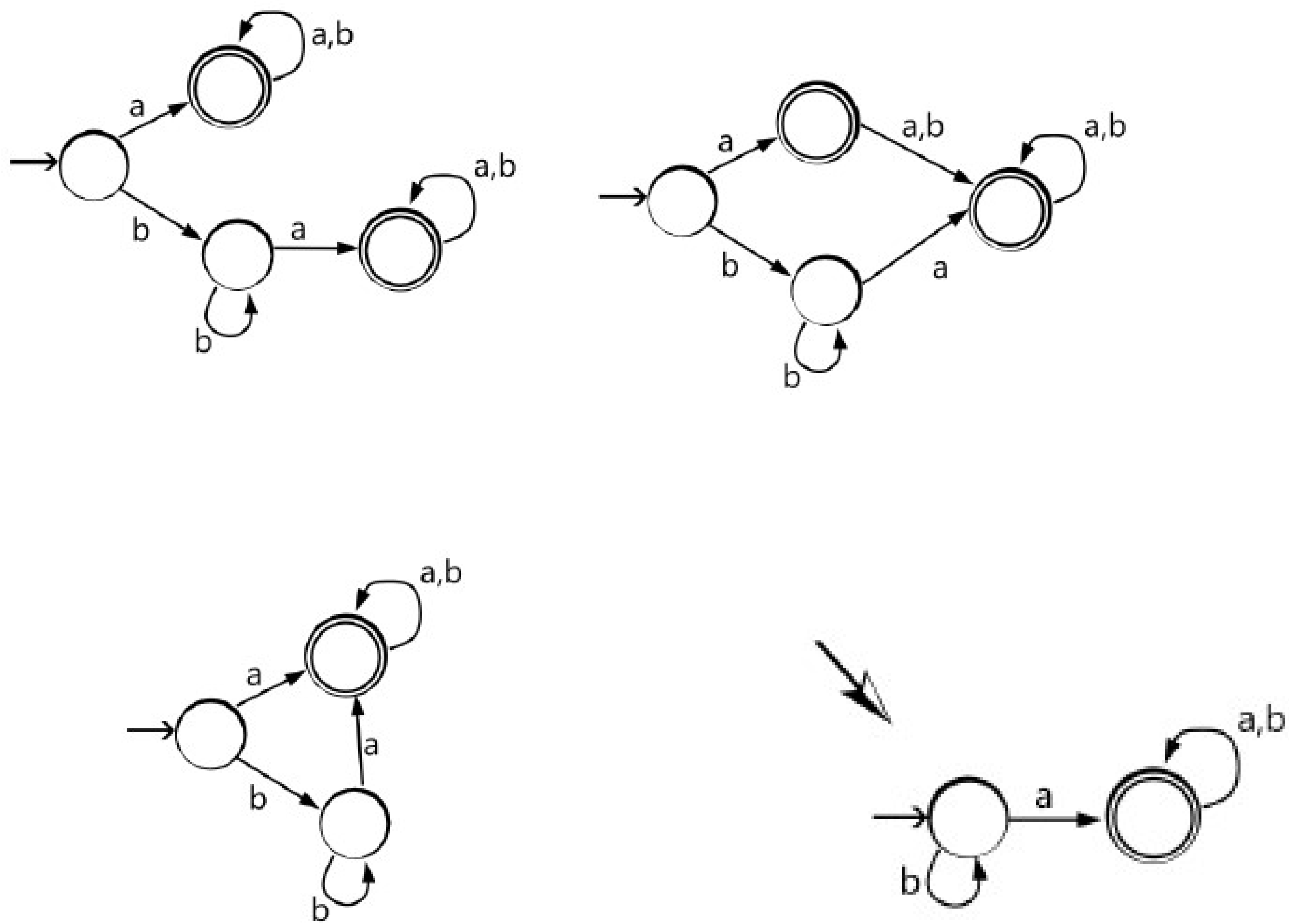
- **Apakah dua FA di bawah ini ekuivalen? Bagaimanakah bentuk FA dgn minimal statenya?**






1
0
1
0
1
0
1
0
2


1
0
1
0,2
1
0



a,b
a
b
a
a,b
b
a,b
a,b
a
a,b
b
a
b
a,b
a
a
b
b
a,b
a
b

Closure Properties of Regular Languages

Union, Intersection, Difference, Concatenation, Kleene Closure, Reversal, Homomorphism, Inverse Homomorphism

Informatika

Closure Properties

- **Recall a closure property is a statement that a certain operation on languages, when applied to languages in a class (e.g., the regular languages), produces a result that is also in that class.**
- **For regular languages, we can use any of its representations to prove a closure property.**

Closure Under Union

- **If L and M are regular languages, so is L ∪ M.**
- **Proof: Let L and M be the languages of regular expressions R and S, respectively.**
- **Then R+S is a regular expression whose language is L ∪ M.**

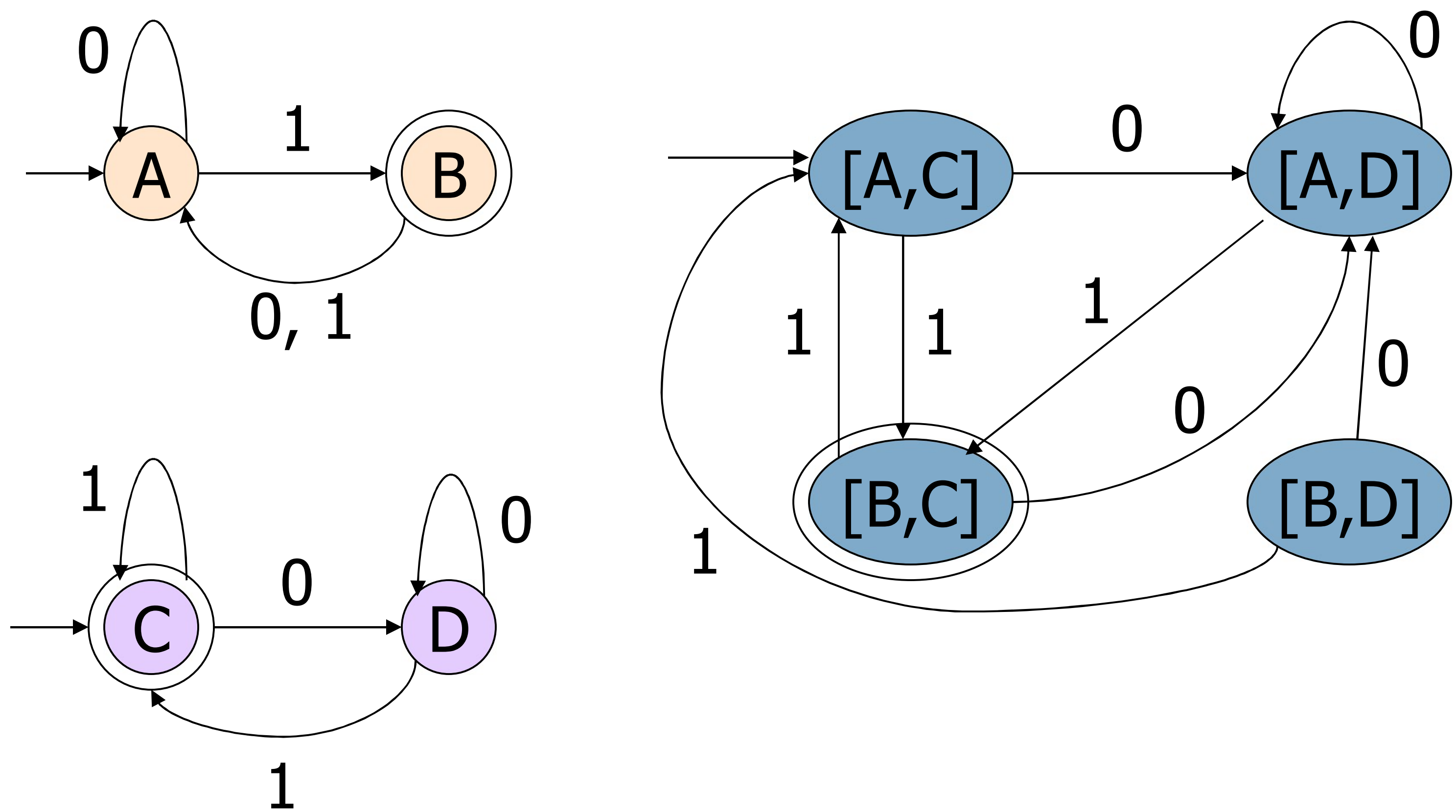
Closure Under Concatenation and Kleene Closure

- **Same idea:**
  - RS is a regular expression whose language is LM.
  - R* is a regular expression whose language is L*.

Closure Under Intersection

- If L and M are regular languages, then so is L $\cap$ M.
- Proof: Let A and B be DFA's whose languages are L and M, respectively.
- Construct C, the product automaton of A and B.
- Make the final states of C be the pairs consisting of final states of both A and B.

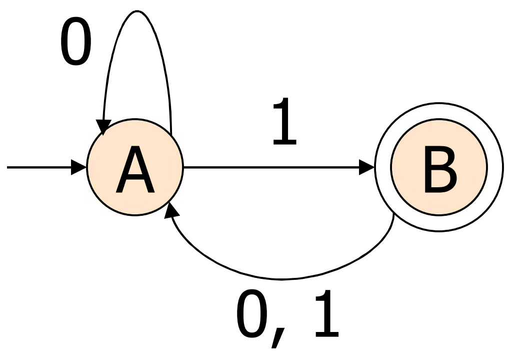
Example: Product DFA for Intersection

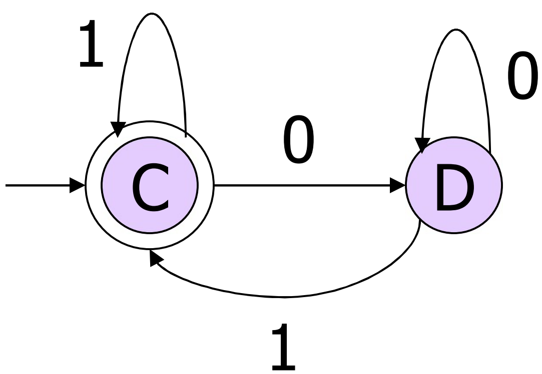


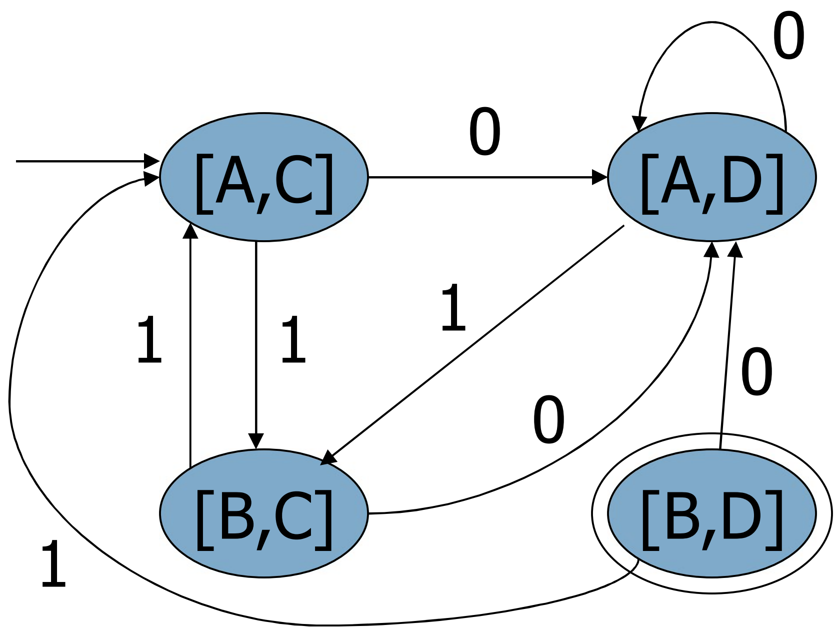
Closure Under Difference

- If L and M are regular languages, then so is ***L – M*** = strings in L but not M.
- Proof: Let A and B be DFA's whose languages are L and M, respectively.
- Construct C, the product automaton of A and B.
- Make the final states of C be the pairs where A-state is final but B-state is not.

Example: Product DFA for Difference







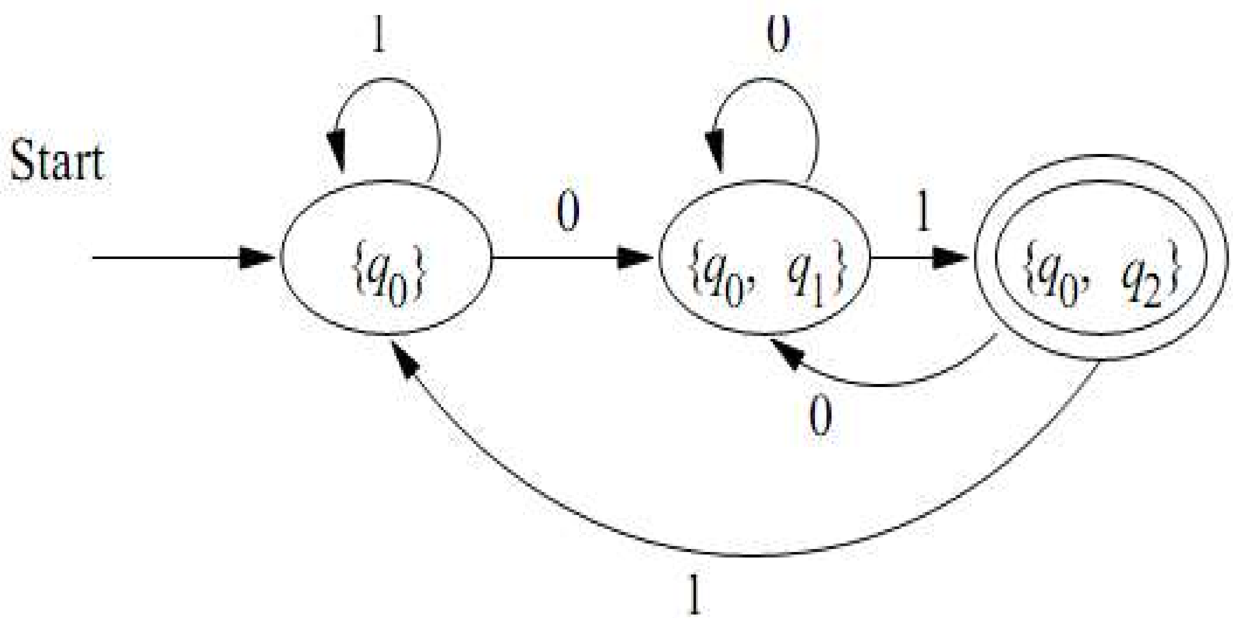
Notice: difference
is the empty language

Closure Under Complementation

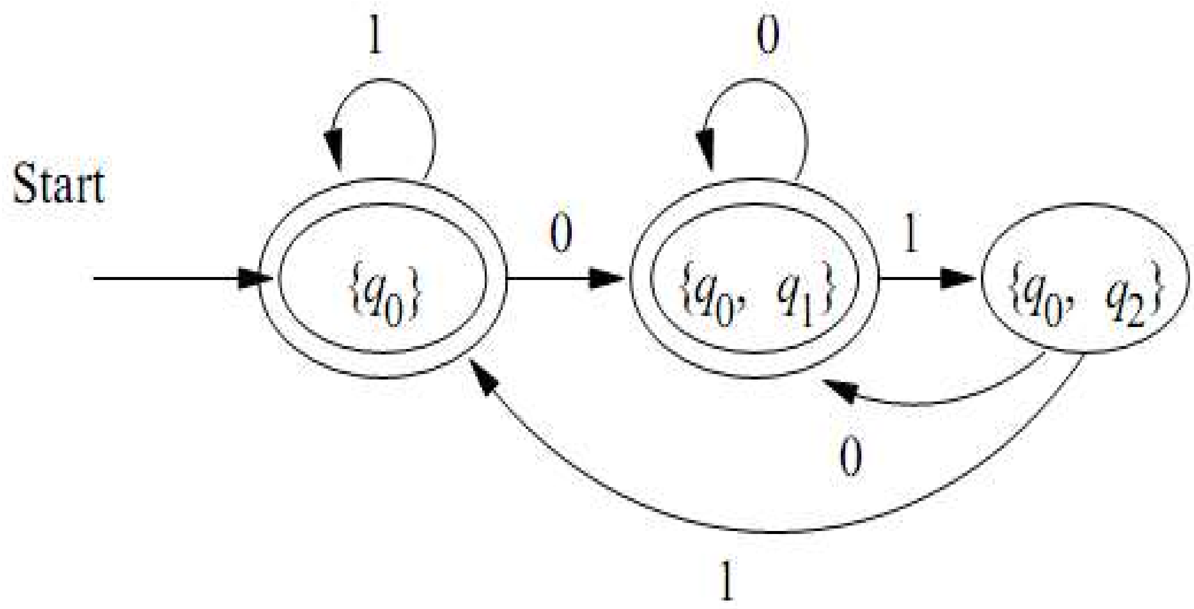
- **The *complement* of a language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* - L$.**
- **Since Σ^* is surely regular, the complement of a regular language is always regular.**

Example:

Let L be recognized by the DFA below



Then $\overline{L}$ is recognized by



Question: What are the regex's for L and $\overline{L}$

Closure Under Reversal

- **Recall example of a DFA that accepted the binary strings that, as integers were divisible by 23.**
- **We said that the language of binary strings whose reversal was divisible by 23 was also regular, but the DFA construction was very tricky.**
- **Good application of reversal-closure.**

Closure Under Reversal – (2)

- **Given language L, L^R is the set of strings whose reversal is in L.**
- **Example: L = {0, 01, 100}; L^R = {0, 10, 001}.**
- **Proof: Let E be a regular expression for L.**
- **We show how to reverse E, to provide a regular expression E^R for L^R.**

Reversal of a Regular Expression

- **Basis: If E is a symbol a, ϵ, or $\emptyset$, then $E^R = E$.**
- **Induction: If E is**
  - F+G, then $E^R = F^R + G^R$.
  - FG, then $E^R = G^R F^R$
  - F*, then $E^R = (F^R)^*$.

Example: Reversal of a RE

- Let $E = 01^* + 10^*$.
- $E^R = (01^* + 10^*)^R = (01^*)^R + (10^*)^R$
- $= (1^*)^R 0^R + (0^*)^R 1^R$
- $= (1^R)^* 0 + (0^R)^* 1$
- $= 1^* 0 + 0^* 1$.

Homomorphisms

- **A *homomorphism* on an alphabet is a function that gives a string for each symbol in that alphabet.**
- **Example: h(0) = ab; h(1) = ϵ.**
- **Extend to strings by $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$.**
- **Example: h(01010) = ababab.**

Closure Under Homomorphism

- If L is a regular language, and h is a homomorphism on its alphabet, then h(L) = {h(w) | w is in L} is also a regular language.
- Proof: Let E be a regular expression for L.
- Apply h to each symbol in E.
- Language of resulting RE is h(L).

Example: Closure under Homomorphism

- **Let h(0) = ab; h(1) = ϵ.**
- **Let L be the language of regular expression 01* + 10*.**
- **Then h(L) is the language of regular expression abϵ* + ϵ(ab)*.**

Note: use parentheses to enforce the proper grouping.

Example – Continued

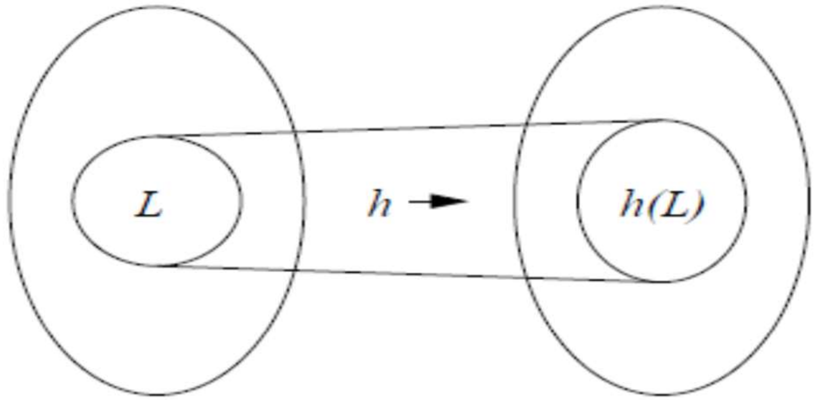
- **abϵ* + ϵ(ab)* can be simplified.**
- **ϵ* = ϵ, so abϵ* = abϵ.**
- **ϵ is the identity under concatenation.**
  - That is, ϵE = Eϵ = E for any RE *E*.
- **Thus, abϵ* + ϵ(ab)* = abϵ + ϵ(ab)* = ab + (ab)*.**
- **Finally, L(ab) is contained in L((ab)*), so a RE for h(L) is (ab)*.**

Inverse Homomorphisms

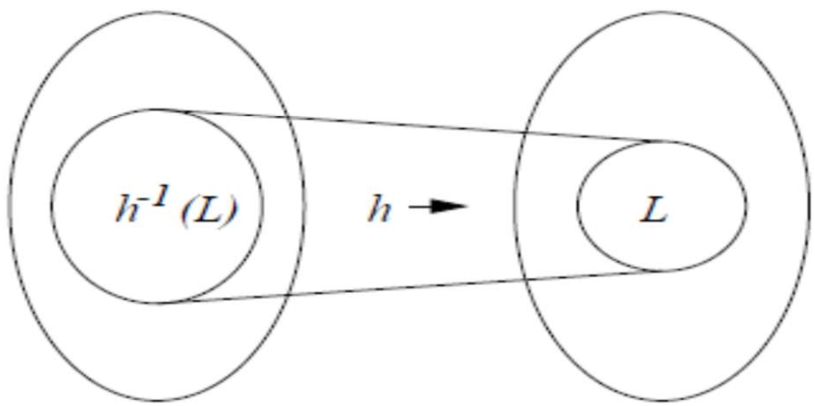
- **Let h be a homomorphism and L a language whose alphabet is the output language of h.**
- **$h^{-1}(L)$ = {w | h(w) is in L}.**

Let $h : \Sigma^* \to \Theta^*$ be a homom. Let $L \subseteq \Theta^*$, and define

$$h^{-1}(L) = \{w \in \Sigma^* : h(w) \in L\}$$



(a)



(b)

Example: Inverse Homomorphism

- Let $h(0) = ab$; $h(1) = \epsilon$.
- Let $L = \{abab, baba\}$.
- $h^{-1}(L)$ = the language with two 0's and any number of 1's = $L(1^*01^*01^*)$.

Notice: no string maps to baba; any string with exactly two 0's maps to abab.

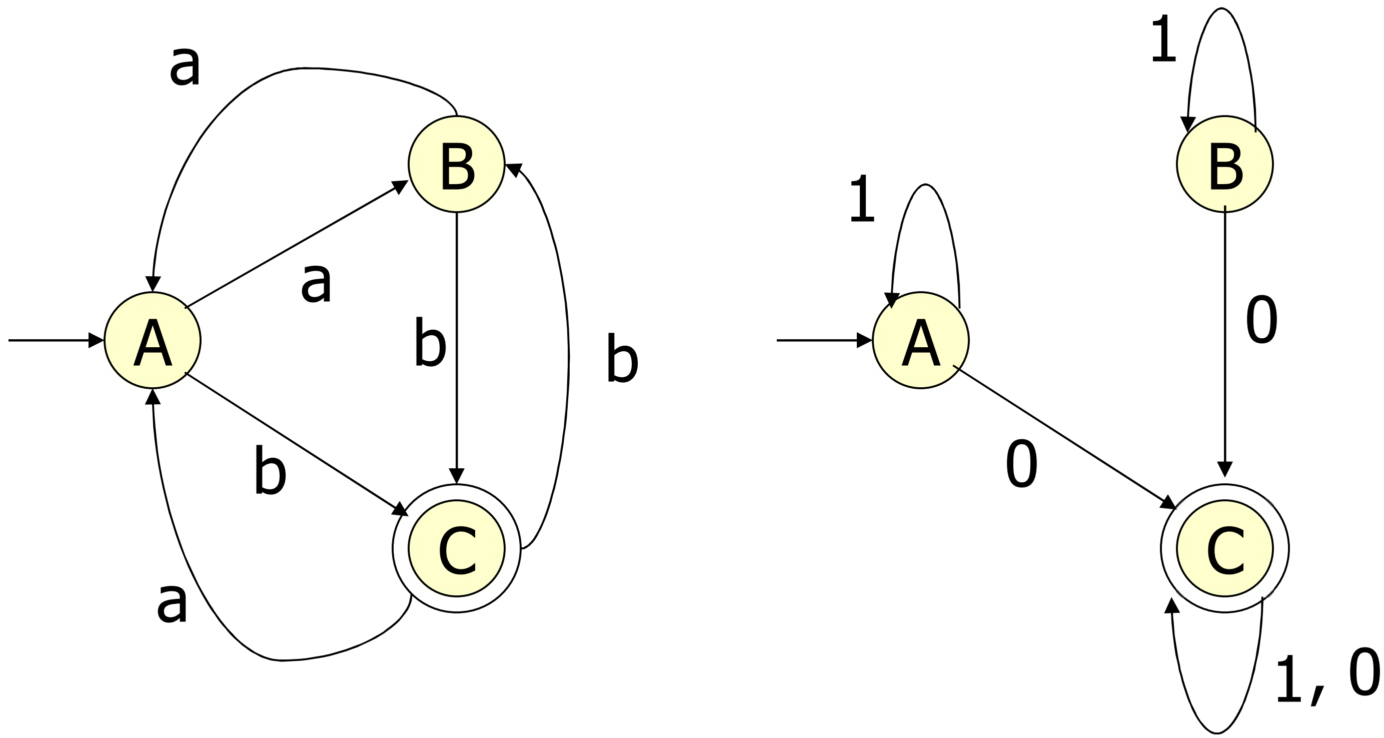
Closure Proof for Inverse Homomorphism

- **Start with a DFA A for L.**
- **Construct a DFA B for $h^{-1}(L)$ with:**
  - The same set of states.
  - The same start state.
  - The same final states.
  - Input alphabet = the symbols to which homomorphism h applies.

Proof – (2)

- **The transitions for B are computed by applying h to an input symbol *a* and seeing where A would go on sequence of input symbols h(a).**
- **Formally, $\delta_B(q, a) = \delta_A(q, h(a))$.**

Example: Inverse Homomorphism Construction



Since
h(1) = ϵ

Since
h(0) = ab

h(0) = ab
h(1) = ϵ