CS3063 Theory of Computing

Semester 4 (21 Intake), Jan – May 2024

Lecture 5

Regular Languages & Finite Automata – Session 4

Sanath Jayasena

Announcements

- Assignment 1: due 04th March

- Week 6: Quiz 4 (based on L5, this lecture)
  - Thursday 29th Feb
  - 8.15am: Group 2
  - 9.15am: Group 1

Today’s Outline

Lecture 5

- FA with outputs (Moore, Mealy models)
- Pumping lemma for regular languages
- Applications of FA
- State Minimization

[Conclusion of “FA + Regular Languages”]

Overview of Topics Covered:

- Regular expressions/languages
- Finite automata (FA)
- Regular language $\leftrightarrow$ FA
- NFA
  - Given NFA $\rightarrow$ equivalent deterministic FA
- NFA-Λ
  - Given NFA-Λ $\rightarrow$ equivalent NFA
- Equivalency among DFA, NFA, NFA-Λ

PART 1

Today's Outline

Lecture 5

- FA with outputs (Moore, Mealy models)
- Pumping lemma for regular languages
- Applications of FA
- State Minimization

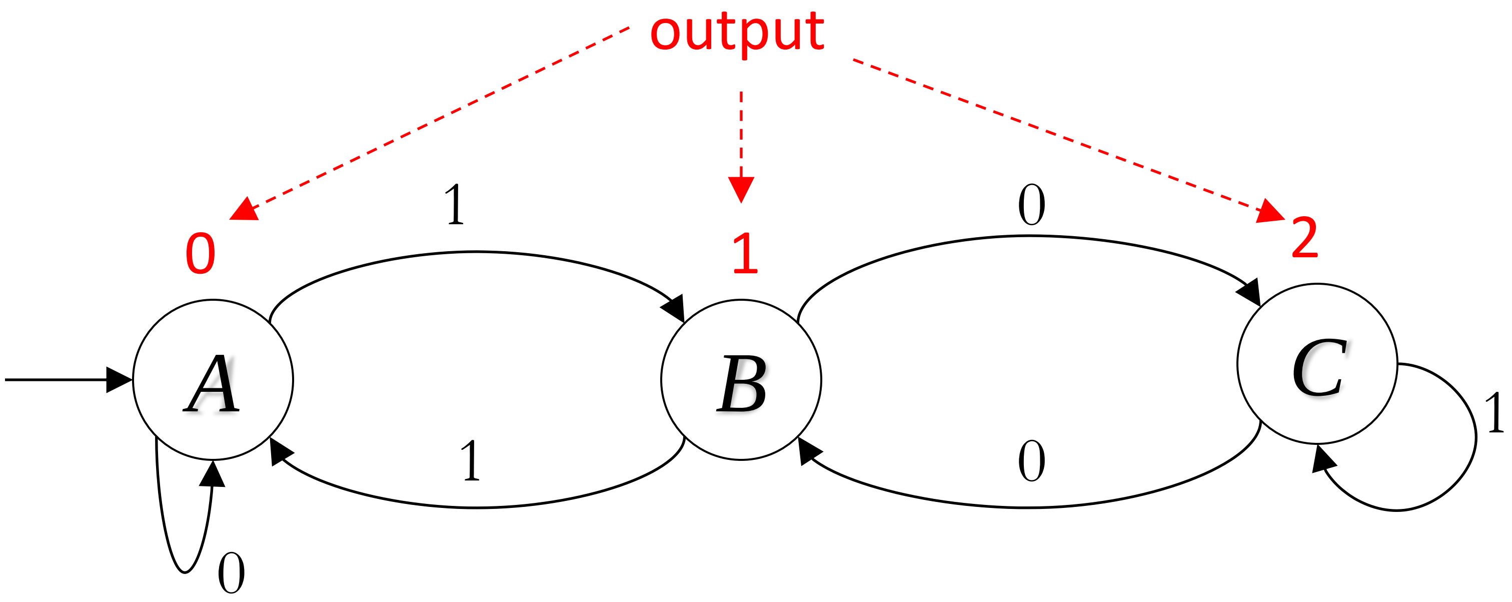
1. FA With Outputs

- So far we considered FA with a binary result: “accept” or “reject”
- Outputs from other alphabets are possible
- Two approaches
  - Moore model/machines
  - Mealy model/machines

Moore Machines

- The output is associated with the state
- Formally, a Moore machine is a 6-tuple $(Q, \Sigma, \Delta, q_0, \delta , \lambda)$ where,
  - Q, Σ, q_0, δ are as in FA we studied
  - Δ is the output alphabet
  - λ is a mapping from Q to Δ (gives the output associated with each state)

Example Moore Machine



Example Moore Machine

- Transition Table
  - For the transition diagram in previous slide

Present state	Next state		Output
	Input=0	Input=1	
→A	A	B	0
B	C	A	1
C	B	C	2

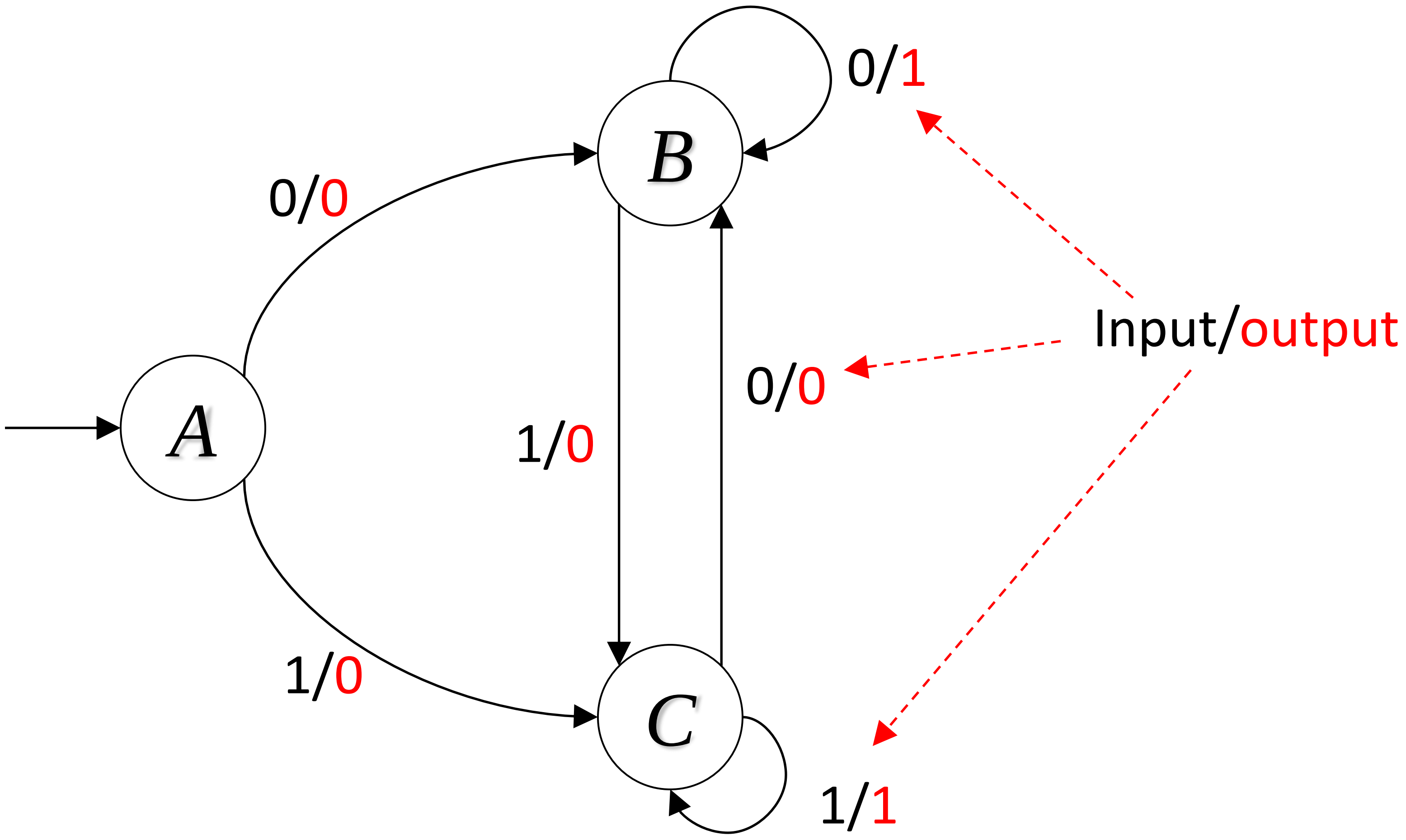
FA → Moore Machine?

- Given an FA, we can get an "equivalent Moore machine" as follows
  - $\Delta = \{0, 1\}$
  - $\lambda(q)=1$ if q is an accepting state
  - $\lambda(q)=0$ if q is not an accepting state

Mealy Machines

- The output is associated with the transition
- A Mealy machine is a 6-tuple $(Q, \Sigma, \Delta, q_0, \delta, \lambda)$ where,
  - All elements are as in the Moore machine, …
  - Except λ maps $Q \times \Sigma$ to Δ
  - That is, $\lambda(q, a)$ gives the output associated with the transition from state q on input a

Example Mealy Machine



Example Mealy Machine

- Transition Table
  - For the transition diagram in previous slide

Present State	Input=0		Input=1	
	Next State	Output	Next State	Output
→A	B	0	C	0
B	B	1	C	0
C	B	0	C	1

Moore vs. Mealy Models

- If the input string is of length ***n***, the length of the output string is:
  - For a Moore machine → ***n***+1
    - $\lambda(q_0)$ is the same for all cases
  - For a Mealy machine → ***n***

- What is the output for input Λ ?
  - Moore machine gives output $\lambda(q_0)$
  - Mealy machine gives output Λ

Moore-Mealy Equivalence

- Ignoring the output of a Moore machine for input Λ, *for a given Moore machine there is an equivalent Mealy machine* (and vice versa)
  - i.e., for a given input string, the output strings would be the same for the two machines
- Homework
  - Find how to convert between the two types

PART 2

Today's Outline

Lecture 5

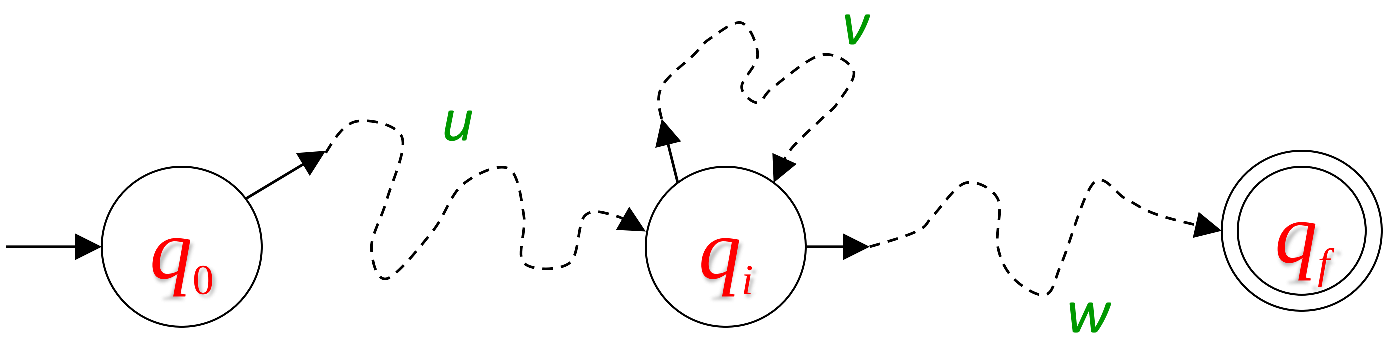
- FA with outputs (Moore, Mealy models)
- **Pumping lemma for regular languages**
- Applications of FA
- State Minimization

2. Pumping Lemma

- Allows us to prove non-regularity (i.e., that a language is not regular)
- A theorem that says all regular languages have a special property
  - Suppose $M=(Q, \Sigma, q_0, A, \delta)$ is an FA that recognizes a language L
  - Strings with sufficient length (pumping length) in the language can be "pumped" up
  - These strings correspond to "loops" in the path of transitions from start state to accepting state

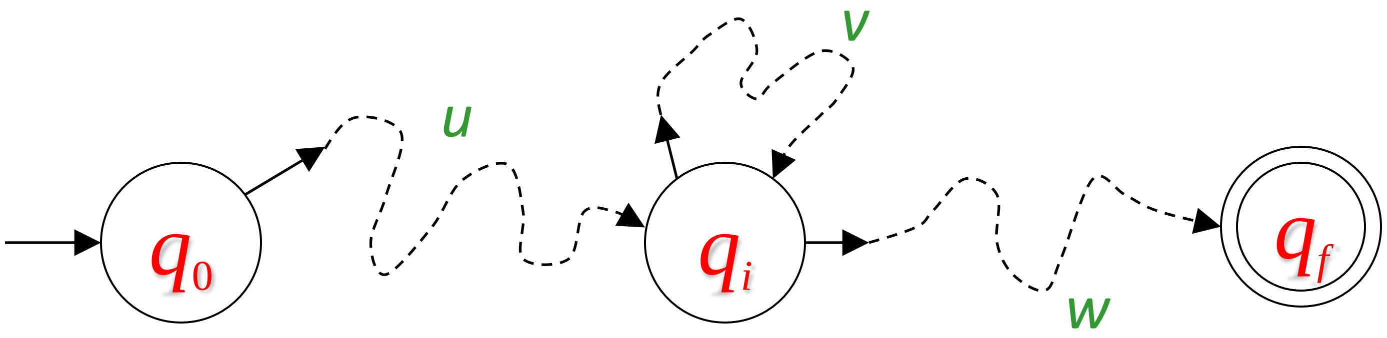
Pumping Lemma ...contd

- For a string $x \in L$, if M enters a state twice then we have a path with a loop
  – x is of the form uvw where v corresponds to the loop



Pumping Lemma ...contd

- If $|Q|=n$, for a string x in L with length at least n
  - We can write, $x=a_1a_2\ldots a_ny$
  - The sequence of $n+1$ states $q_0=\delta^*(q_0,\Lambda)$, $q_1=\delta^*(q_0, a_1)$, $q_2=\delta^*(q_0, a_1a_2)$,…, $q_n=\delta^*(q_0, a_1a_2\ldots a_n)$ must contain some state at least twice (where loop exists)
  - $\delta^*(q_i, v) = q_i$ means $\delta^*(q_i, v^m) = q_i$ for every $m \geq 0$
  - So, $\delta^*(q_0, uv^mw) = q_f$ for every $m \geq 0$



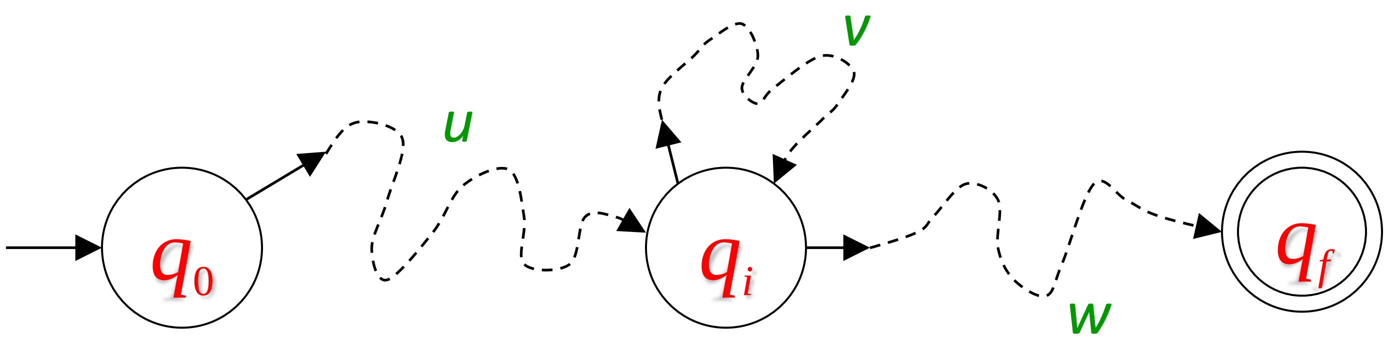
Pumping Lemma ...contd

- Pumping Lemma: Version 1
  – Suppose L is a regular language recognized by an FA with n states. For any string x in L with $|x| \geq n$, x may be written as $x=uvw$ for some strings u, v and w satisfying

    $|uv| \leq n$

    $|v| > 0$

    for any $m \geq 0$, uv^mw is in L



Pumping Lemma ...contd

- Pumping Lemma: Version 2 (more common)
  - Suppose L is a regular language. Then there is an integer n so that for any x in L with $|x| \geq n$, there are strings u, v and w so that

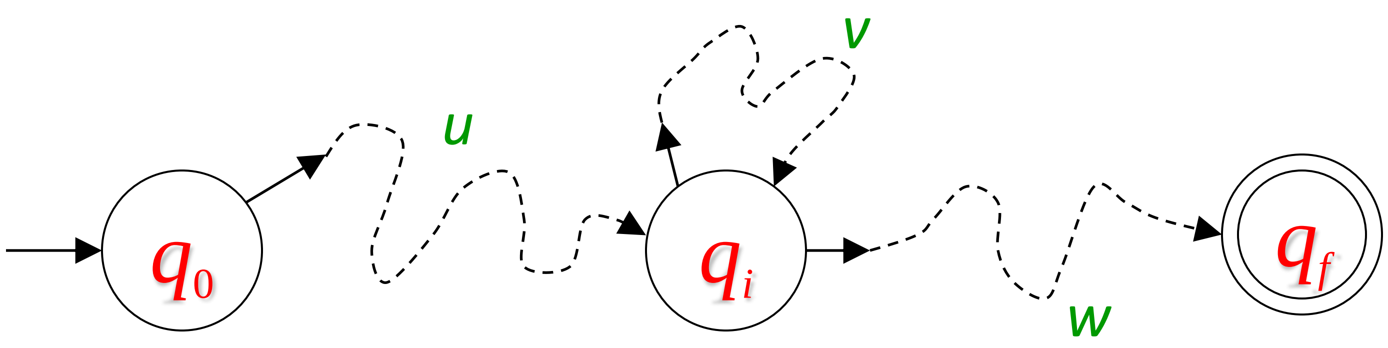
$x = uvw$

$|uv| \leq n$

$|v| > 0$

for any $m \geq 0$, $uv^m w$ is in L

Either ***u*** or ***w*** may be Λ, but ***v*** can't be Λ



Pumping Lemma ...contd

- Idea: for an arbitrary string of sufficient length in *L*, a portion of it can be pumped up
  - Lemma gives a *necessary condition* to be regular

- To prove that a language is not regular using this lemma, we must show that the language does not have the property described in it
  - Can assume property holds and show contradiction
  - E.g., assume there is an ***n*** (although we do not know it), then find a string ***x***, with $|x| \geq n$, that will lead to a contradiction

Example

- Show that $L = \{0^i 1^i \mid i \geq 0\}$ is not regular
  - Assume properties in pumping lemma hold for L
  - Choose ***x*** with |***x***|$\geq n$; a reasonable choice is ***x***= $0^n 1^n$
  - Lemma says ***x*** can be split into 3 as ***x***=***uvw*** for some ***u***, ***v***, ***w*** and for any ***m*** $\geq$ 0, ***uv^{m}w*** is in L
    - We can show this is not possible, as follows
    - Note: either ***u*** or ***w*** may be Λ, but ***v*** can't be Λ
- **Case 1**: The string ***v*** with only 0s
  - For any ***u***, ***w***, the string ***uvvw*** has more 0s than 1s; → ***uvvw*** is not in L
  - Similarly, for any ***m*** $\geq$ 0, **uv^{m}w** is not in L
  - This case is a contradiction

Example ...contd

- Show that $L = \{0^i 1^i \mid i \geq 0\}$ is not regular
- Case 2: The string ***v*** with only 1s
  - For whatever ***u***, ***w***, for any ***m*** $\geq$ 0, the string **uv**$^{\mathbf{m}}$**w** has more 1s than 0s; so **uv**$^{\mathbf{m}}$**w** is not in L
  - This case is a contradiction
- Case 3: String ***v*** consists of both 0s and 1s
  - In this case, the string ***uvvw*** may have the same number of 0s and 1s, but they will be out of order (some 1s before 0s)
  - A contradiction

Example ...contd

- Show that $L = \{0^i 1^i \mid i \geq 0\}$ is not regular
  - [Cases 2 and 3 can be eliminated by considering the condition $|\boldsymbol{uv}| \leq n$]
  - Contradictions for all cases of ***v***
  - L cannot be regular

- Programming languages are not regular
  - E.g., main( ) $\{^n\ \}^n$

PART 3

Today's Outline

Lecture 5

- FA with outputs (Moore, Mealy models)
- Pumping lemma for regular languages
- Applications of FA
- State Minimization

3. Applications of FA

- Modeling of *reactive systems*
  - Reactive system
    - A system that changes its actions, outputs and status in response to stimuli from within or outside
    - Maintains an ongoing interaction with the environment rather than produce some final value upon termination
  - Examples
    - Vending machines, ATMs, communication protocols
    - Systems for air-traffic control
    - Control systems for trains, planes, nuclear plants

Applications of FA

- Some software design problems simplified by using regular expressions or converting regular expressions to FA

- Though programming languages are not regular, *tokens* (identifiers, literals, operators, reserved words, punctuation) can be described by regular expressions

Applications of FA

- Lexical analysis/analyzers
  - First phase in compiling a program
  - Identifying and classifying the tokens
  - Lexical-analyzer generator
    - Input: sequence of regular expressions (for tokens)
    - Output: a lexical analyzer (an FA) to recognize any token
    - E.g., *lex* and *flex*

Applications of FA ...contd

- Text editors
  - Operations based on regular expressions
  - For searching, substitution
  - E.g., **vi** editor
    - `%s/\s\s\s*/\s/` → substitute two or more spaces by a single space
- "grep": utility to search for reg. expressions
- Other similar tools, situations…

PART 4

Today’s Outline

Lecture 5

- FA with outputs (Moore, Mealy models)
- Pumping lemma for regular languages
- Applications of FA
- State Minimization

4. (State) Minimization of DFA

- Minimization of DFA means *minimizing the number of states* of an DFA
- Detailed discussion on this requires understanding of equivalence relations and equivalence classes of states
- *Myhill-Nerode Theorem*
  - ***Reading assignment***
  - Provides a necessary and sufficient condition for a language to be regular

Minimization ...contd

- Myhill-Nerode theorem implies that there is a unique minimum-state DFA for every regular language

- Idea is to identify pairs of *equivalent states*
  - Two states q_i and q_j are equivalent if some language L takes the DFA from either state to an accepting state (same or different)

Minimization ...contd

- In practice, rather than looking for pairs of equivalent states we find pairs (p, q) of *distinguishable states*, which is easier
  - i.e., $\delta^*(p, x)$ is an accepting state and $\delta^*(q, x)$ is not, or vice versa, for some string x
- **If two states are not *equivalent*, they are *distinguishable***
  - All pairs of states are presumed equivalent until they are proved distinguishable

State Minimization ...contd

- Initially we have *two equivalence classes* or *two distinguishable sets of states*
  - The *set of accepting states*, and
  - The *set of non-accepting states*
- But we initially don't know the equivalence relation between 2 states in one class
  - So, next we consider pairs of states presumed equivalent (not yet distinguishable)
  - For this, consider transitions from states

State Minimization ...contd

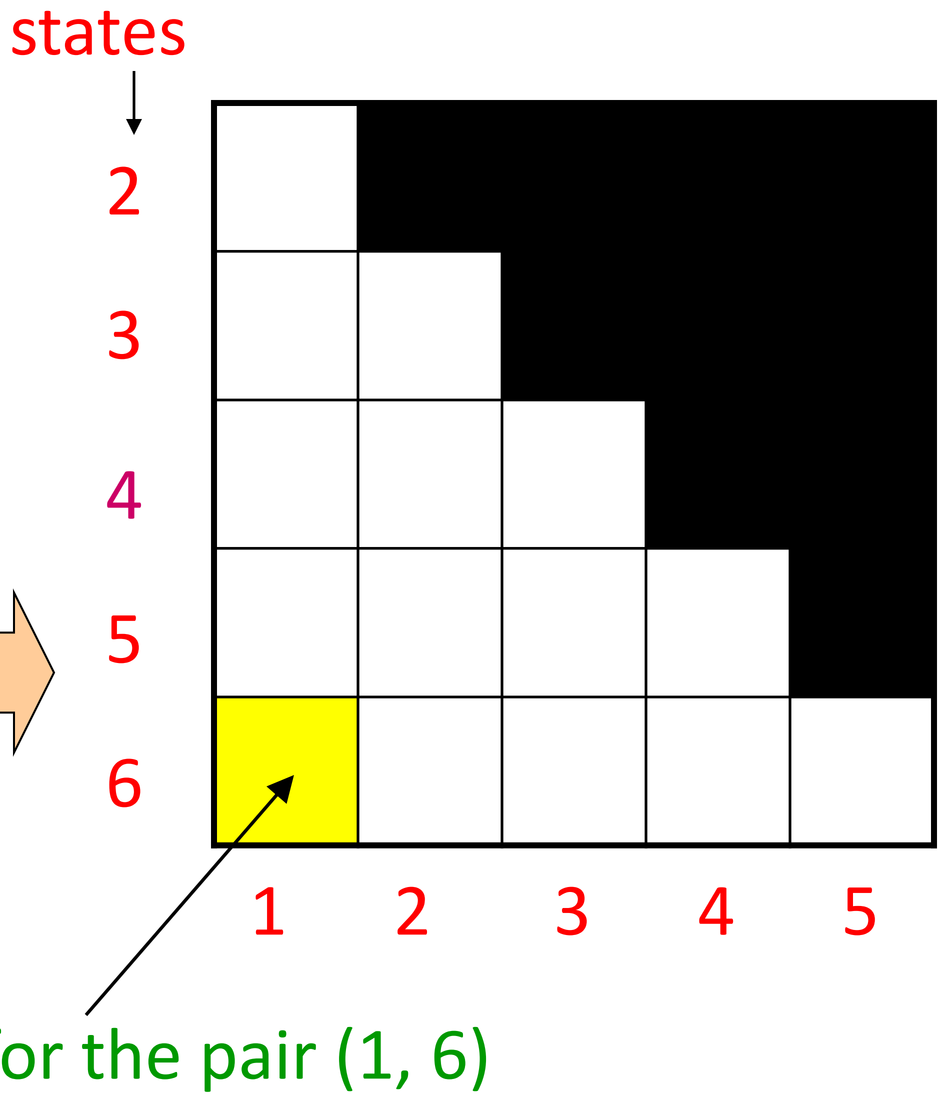
- We look at single symbols from Σ to check the transitions from pairs of states
  - If all symbols in Σ take a DFA from states p and q to accepting states, then p and q are equivalent
  - Even if one symbol in Σ takes a DFA from states p and q to a pair of states already known to be distinguishable, then p and q are also distinguishable

Minimization Algorithm

- To identify distinguishable pairs of states
  - List all (unordered) pairs of states
  - Make a sequence of passes through these
  - **1st pass**: mark each pair of which exactly one is an accepting state
  - **Next passes**: mark any pair (p, q) if there is an a in Σ for which $\delta(p, a)=r$, $\delta(q, a)=s$ and also (r, s) is already marked
  - After a pass with no new pair marked, ***stop***
  - Marked states → distinguishable, else → equivalent

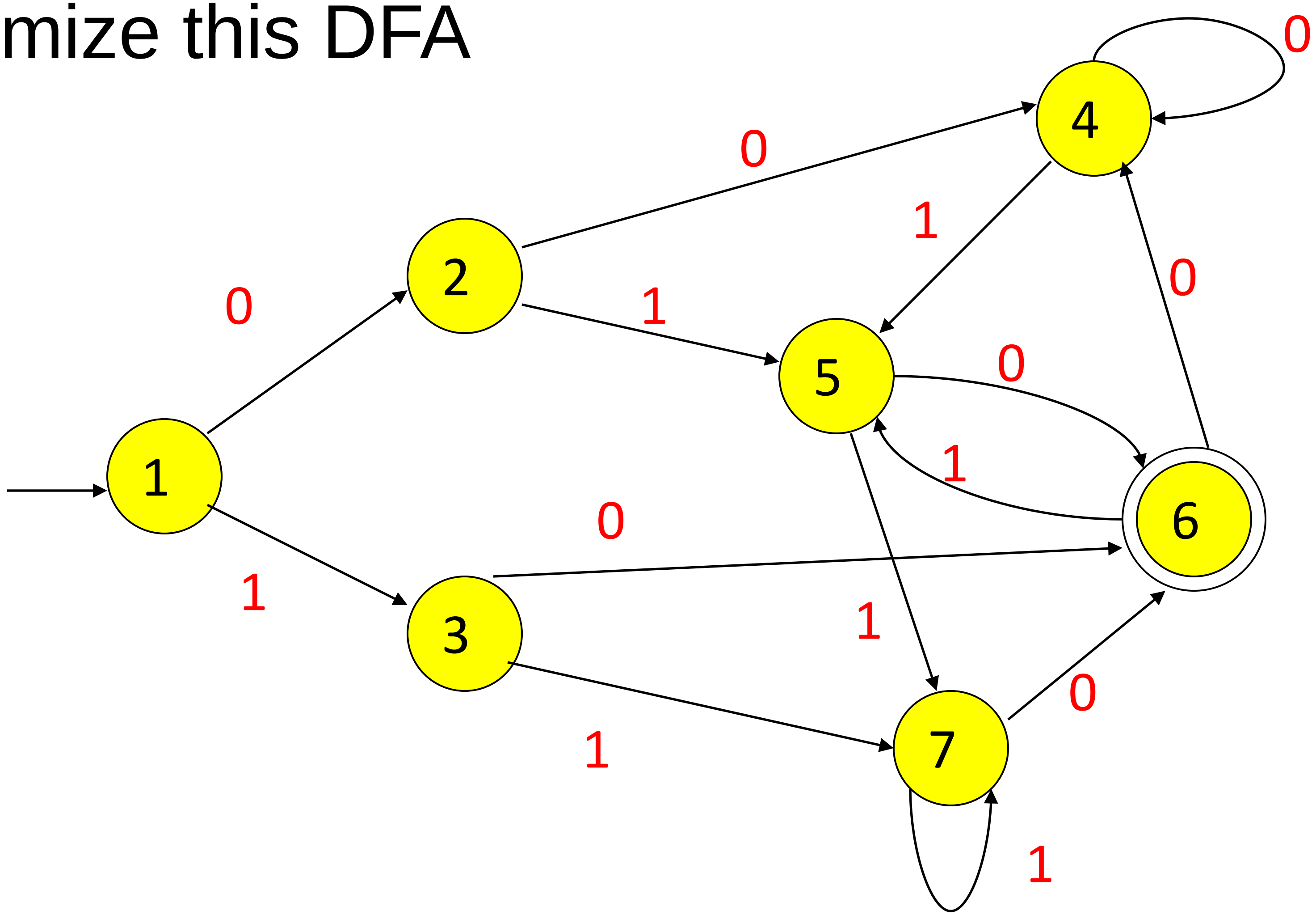
Minimization Algorithm

- Can use a lower (or upper) triangular matrix to mark the pairs in passes
- This is the ***distinguishability matrix***
  - Example is for a DFA with 6 states



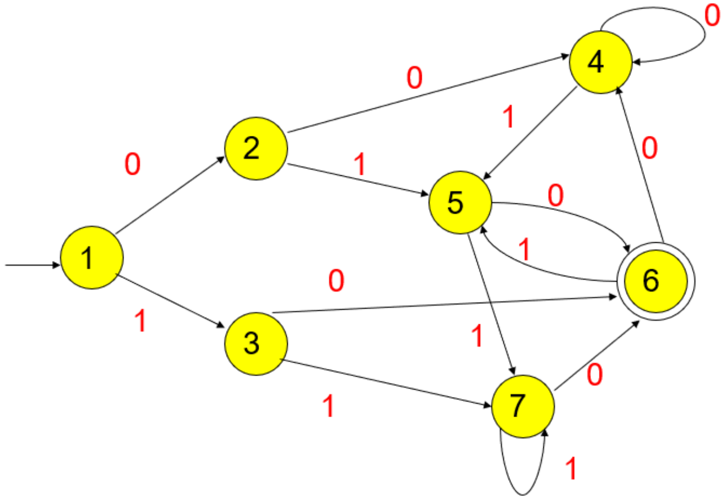
Example 5.6 in Book (p. 179)

- Minimize this DFA



Solution – Step 1

- First pass
  - Pairs marked as “1” are those with exactly one element being an (the only) accepting state

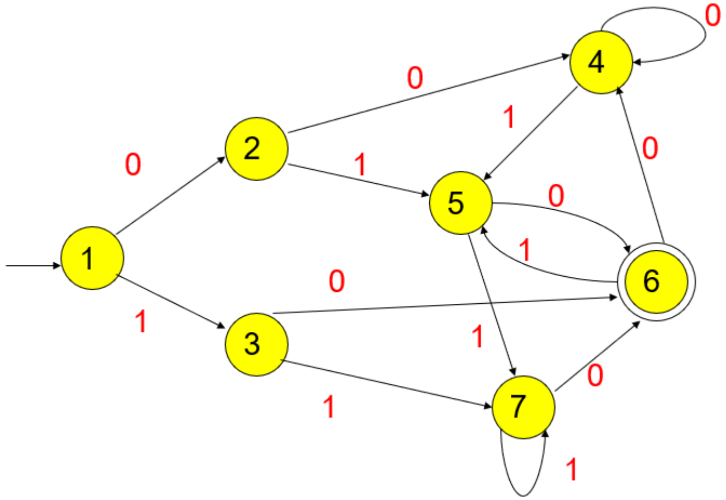


Distinguishability Matrix

	1	2	3	4	5	6
2						
3						
4						
5						
6	1	1	1	1	1	
7						1

Solution – Step 2

- 2^{nd} pass
  - Pairs marked as "2"
  - (2,5) is marked because $\delta(2,0)=4$, $\delta(5,0)=6$ and (4, 6) is already marked
  - Similarly for other cases



Distinguishability Matrix

	1	2	3	4	5	6
2						
3	2	2				
4			2			
5	2	2		2		
6	1	1	1	1	1	
7	2	2		2		1

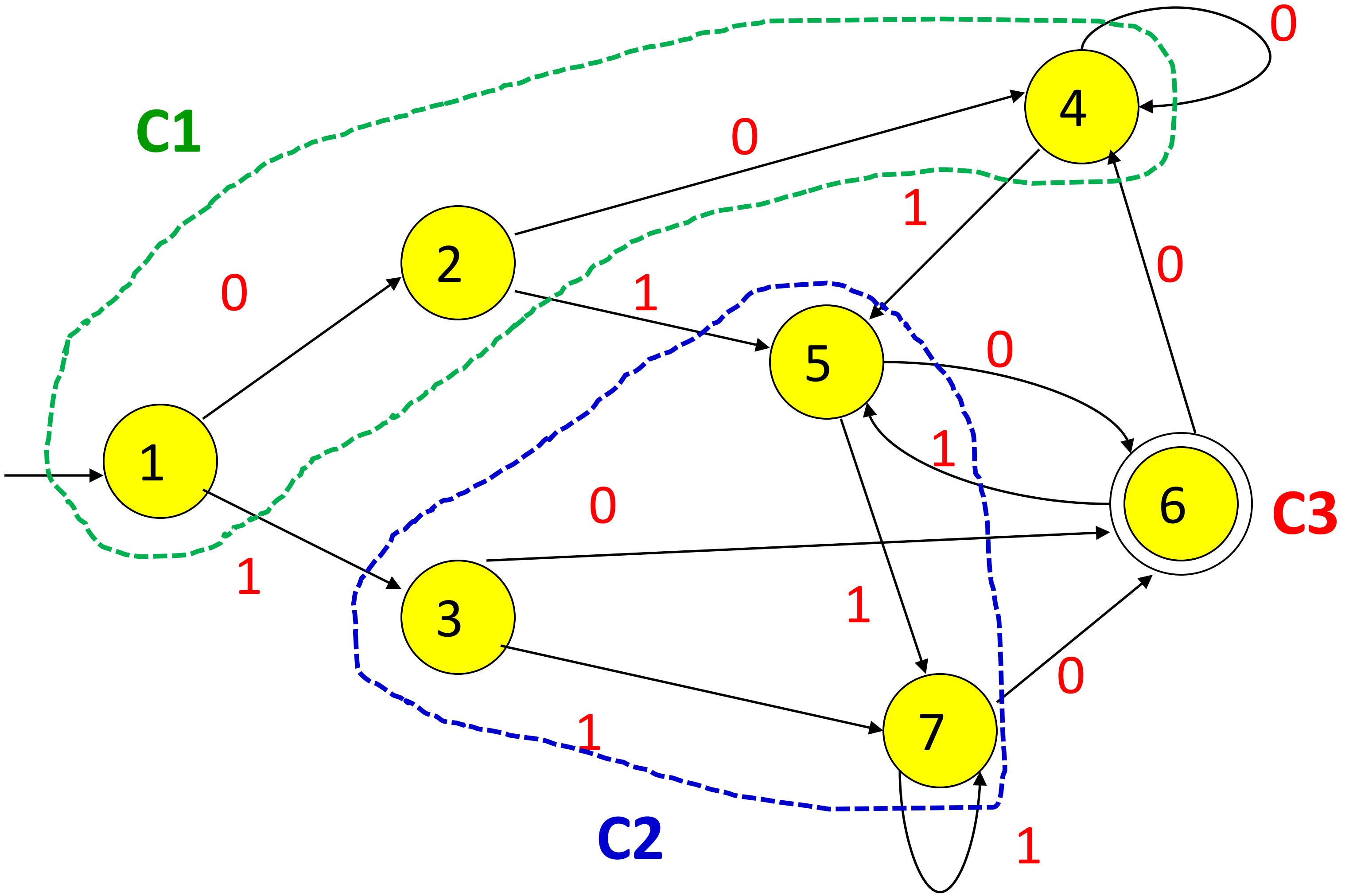
Solution – Step 3

- 3rd pass
  - No new pairs marked
- Stop !!
- Equivalence classes
  - {1, 2, 4}
  - {3, 5, 7}
  - {6}

Distinguishability Matrix

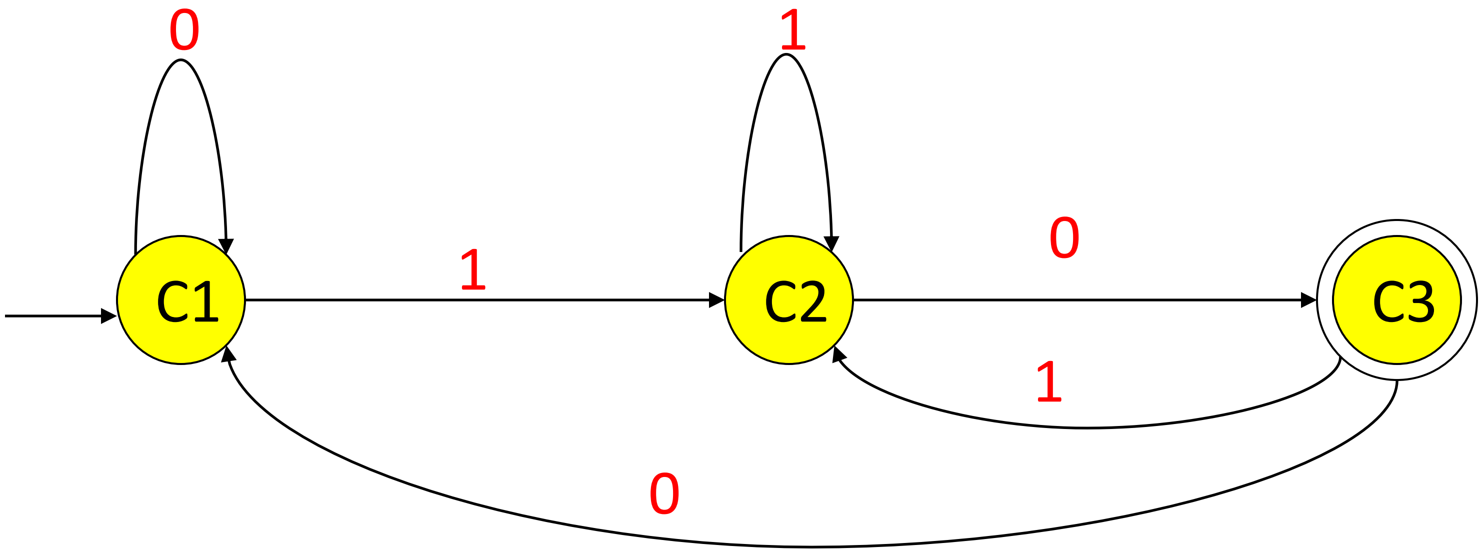
2						
3	2	2				
4			2			
5	2	2		2		
6	1	1	1	1	1	
7	2	2		2		1
	1	2	3	4	5	6

Solution – Step 4



Solution – Final Answer

- Minimum state DFA

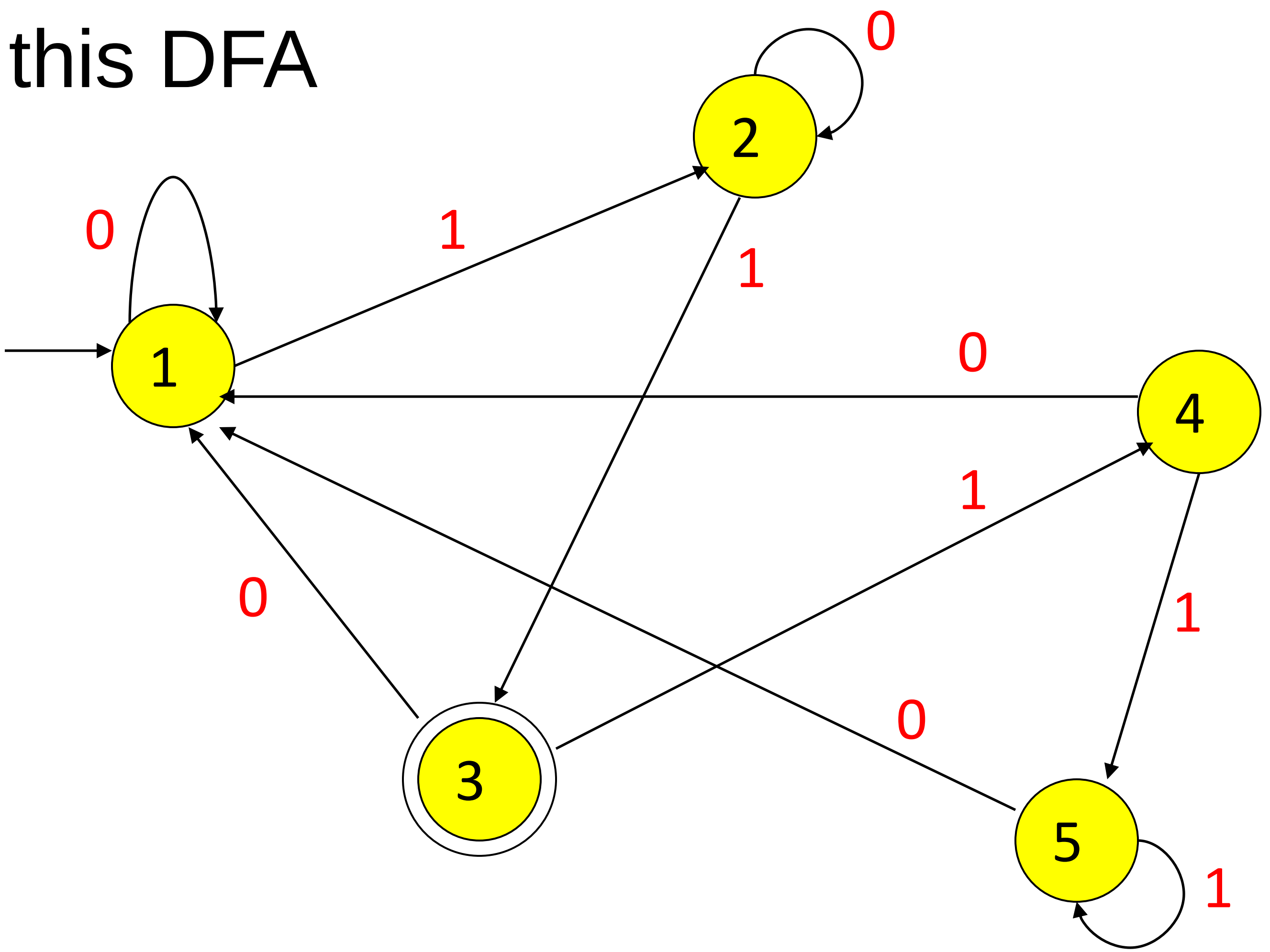


More on Minimization

- Within a pass, the following is possible
  - A pair (p,q) is unmarked while every pair (r, s) such that $\delta(p\ a)=r$, $\delta(q,\ a)=s$ for every a in Σ is also unmarked
  - Add (p,q) to a linked list for each (r, s); if later (r, s) is marked, then mark (p,q) also
- At the end
  - an unmarked-pair means the 2 states are equivalent and can be merged
  - # of equivalent classes = # of minimum states

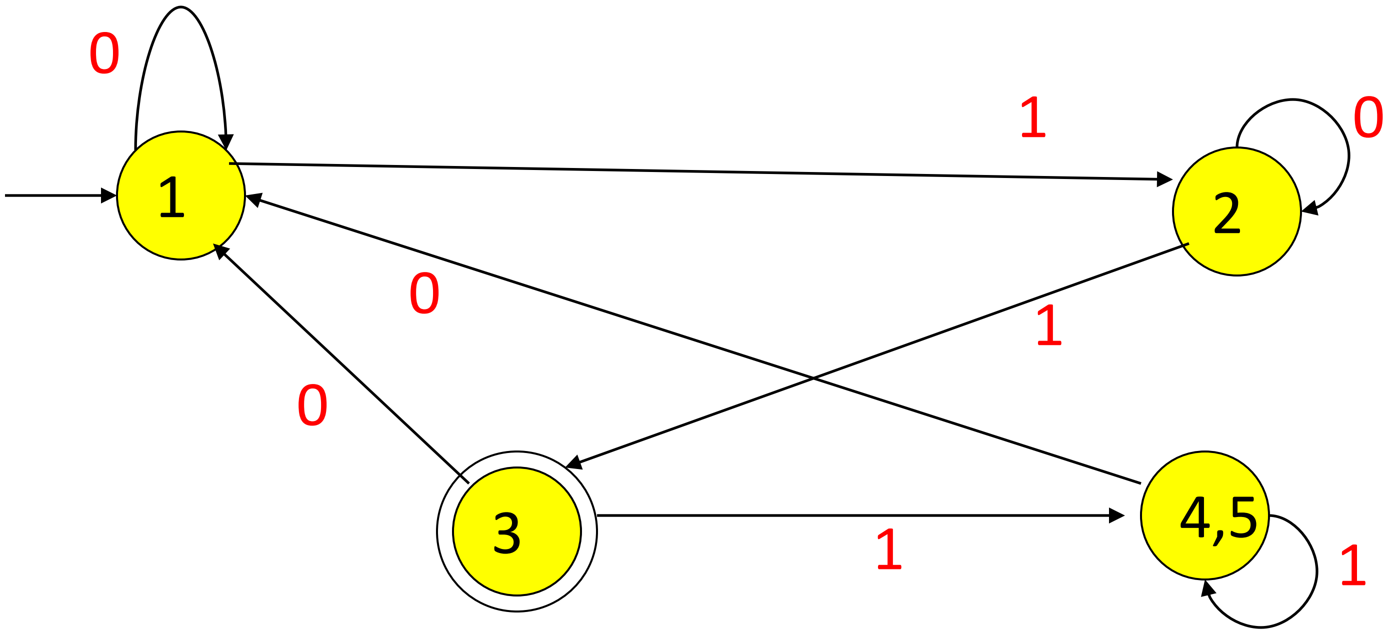
Exercise

- Minimize this DFA



Solution

- States 4 and 5 are equivalent (in the same equivalence class, indistinguishable)
  - Can merge 4 and 5



Conclusion

- Today we discussed
  - FA with output
  - Pumping lemma
  - Applications of FA
  - State Minimization
- We conclude “FA+Regular Languages”

- Next topic: Context-free Languages