# Properties of Regular Languages

# Topics

1) How to prove whether a given language is regular or not?
2) Closure properties of regular languages
3) Decision Properties
4) Minimization of DFAs

# Some languages are *not* regular

When is a language - regular?

if we are able to construct one of the following: DFA *or* NFA *or* $\varepsilon$ -NFA *or* regular expression

When is it not?

If we can show that no FA can be built for a language

# How to prove languages are ***not*** regular?

What if we cannot come up with any FA?

A) Can it be language that is not regular?

B) Or is it that we tried wrong approaches?

How do we *decisively* prove that a language is not regular?

# Example of a non-regular language

FA → non regular

01

0011

× 000111

Let L = {w | w is of the form $0^n1^n$ , for all n≥0}

- *Hypothesis:* *L is not regular*

00 11

- A formal rationale:
  - By contradiction, if L is regular then there should exist a DFA for L.
  - Let k = number of states in that DFA.
  - Consider the special word w= $0^k1^k$ => w $\in$ L
  - DFA is in some state $p_i$, after consuming the first i symbols in w

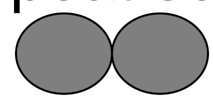
Uses Pigeon Hole Principle

0011 $0^n 1^n$

$n \geq 0$

$0 \xrightarrow{0} 0$

# Rationale…

- Let $\{p_0, p_1, \ldots p_k\}$ be the sequence of states that the DFA should have visited after consuming the first k symbols in w which is $0^k$
- But there are only k states in the DFA!
- ==> at least one state should repeat somewhere along the path (by  +  Principle)
- ==> Let the repeating state be $p_i=p_J$ for $i < j$
- ==> DFA accepts strings w/ unequal number of 0s and 1s, implying that the DFA is wrong!

# The Pumping Lemma for Regular Languages

L → PL → Pass → undecidable than L is regular

fail → L is not regular

- **What it is?**
  The Pumping Lemma is a property of all regular languages.
- **How is it used?**
  A technique that is used to show that a given language is not regular

If L is finite — always regular

L $\infty$ — then L is regular / non regular

# Pumping Lemma for Regular Languages

Let L be a regular language

Then *there exists* some constant ***N*** such that *for every* string $w \in L$ $|w| \geq N$, *there exists* a way to break $w$ into three parts, $w=xyz$, such that:

1. $y \neq \varepsilon$
2. $|xy| \leq N$
3. For all $k \geq 0$, all strings of the form $xy^kz \in L$

This property should hold for all regular languages.

**Definition:** *N* is called the "Pumping Lemma Constant"

# The Purpose of the Pumping Lemma for RL

- To prove that some languages *cannot be* regular.

$L = \{a^n b^n \mid n \geq 0\}$ is not regular using Pumping Lemma

any no. of a's followed by same number of b's

$N=3$

$w = a^n b^n \quad n=3$

$w_i = aaa\,bbb$

$w = xyz$

1. $y \neq \epsilon$
2. $|xy| \leq N$
3. for all $k \geq 0$ $\underline{xy^kz \in L}$

Case 1 'y' is in 'a' part

$$\underbrace{a}_{x}\ \underbrace{a\ a}_{y}\ \underbrace{b\ b\ b}_{z}$$

$k=2$

$w_0 = a\ aa\ aa\ bbb$

$5_a \neq 3_b$

$\therefore$ this L is not regular

Case 2: 'y' is in 'b' part

$k=2$

$N=3$

$$\underbrace{a\ a\ a}_{x}\ \underbrace{b\ b}_{y}\ \underbrace{b}_{z}$$

$aaa\ bb\ bb\ b$

$3_a \neq 5_b$

$\therefore$ this L is not regular

Case 3: 'y' is in 'ab' part

$k=2$

$N=3$

$$\underbrace{a\ a}_{x}\ \underbrace{a\ b}_{y}\ \underbrace{b\ b}_{z}$$

$\underline{a\,a\,ab\,ab\,b\,b}$ $\therefore$ L is not regular

$L = \{ \underline{w}\,\underline{w}^R \mid w \in \Sigma \}$ $\quad \Sigma = 0, 1$

$w = 0101$

$w^R = \underline{1010}$

$L = ww^R$

01011010 is not regular using Pumping Lemma

$w = \underbrace{010}_{x}\,\underbrace{11}_{y}\,\underbrace{010}_{z}$

$N = 3$

$k = 2$

$w = xyz$ $\quad xy^k z \in L$

1. $y \neq \epsilon$
2. $|xy| \leq N$
3. $xy^k z \in L$

$w_0 = \underline{010\ 1111\ 010}$

$w_0 \notin L$

$\therefore$ L is not regular

$L = \{0^n 1 0^n \mid n \geq 1\}$ is not regular using Pumping Lemma

# How to use the pumping lemma?

Think of playing a 2 person game

- Role 1: ***We*** claim that the language cannot be regular

- Role 2: An ***adversary*** who claims the language is regular

- We show that the adversary's statement will lead to a contradiction that implies pumping lemma *cannot* hold for the language.

- We win!!

# How to use the pumping lemma? (The Steps)

1. (we) L is not regular.
2. (adv.) Claims that L is regular and gives you a value for N as its PL constant
3. (we) Using N, choose a string $w \in L$ ,
    1. $|w| \geq N$,
    2. Using w as the template, construct other words $w_k$ of the form $xy^kz$ and show that at least one such $w_k \notin L$

        => this implies we have successfully broken the pumping lemma for the language, and hence that the adversary is wrong.

(Note: In this process, we may have to try many values of k, starting with k=0, and then 2, 3, .. so on, until $w_k \notin L$ )

Note: We don't have any control over N, except that it is positive.
We also don't have any control over how to split w=xyz,
but xyz should respect the PL conditions (1) and (2).

# Using the Pumping Lemma

- What WE do?

  3. Using $N$, we construct our template string $w$
  4. Demonstrate to the adversary, either through pumping up or down on $w$, that some string $w_k \notin L$
     (this should happen regardless of w=xyz)

- What the Adversary does?
  1. Claims L is regular
  2. Provides $N$

Note: This N can be anything (need not necessarily be the #states in the DFA. It's the adversary's choice.)

# Example of using the Pumping Lemma to prove that a language is not regular

**Let $L_{eq}$ = {w | w is a binary string with equal number of 1s and 0s}**

- Your Claim: $L_{eq}$ is not regular
- Proof:
  - By contradiction, let $L_{eq}$ be regular → adv.
  - PL constant should exist
    - Let *N* = that PL constant → adv.
  - Consider input w = $0^N1^N$
    *(your choice for the template string)* → you
  - By pumping lemma, we should be able to break w=xyz, such that: → you
    1) $y \neq \varepsilon$
    2) $|xy| \leq N$
    3) For all k≥0, the string $xy^kz$ is also in L

Template string $w = 0^N 1^N = \underbrace{00 \ldots 0}_{N}\underbrace{11 \ldots 1}_{N}$

# Proof…

- Because $|xy| \le N$, $xy$ should contain only 0s
  - (This and because $y \neq \varepsilon$, implies $y=0^+$)
- Therefore $x$ can contain *at most* N-1 0s
- Also, all the N 1s must be inside $z$
- By (3), any string of the form $xy^kz \in L_{eq}$ for all $k \geq 0$
- Case k=0: $xz$ has at most N-1 0s but has N 1s
- Therefore, $xy^0z \notin L_{eq}$
- This violates the PL (a contradiction) ↯

→ you

Setting k=0 is referred to as **"pumping down"**

Setting k>1 is referred to as **"pumping up"**

Another way of proving this will be to show that if the #0s is arbitrarily pumped up (e.g., k=2), then the #0s will exceed the #1s

- Template string $w = 0^N 1^N$ = 00 …. 011 … 1

  0011
  x=0
  y= 01
  z=1
  Using PL condition $xy^kz \in L$

# Exercise 2

*Prove $L = \{0^n10^n \mid n \geq 1\}$ is not regular*

Note: This n is not to be confused with the pumping lemma constant N. That *can* be different.

In other words, the above question is same as proving:

- $L = \{0^m10^m \mid m \geq 1\}$ is not regular

# Example 3: Pumping Lemma

**Claim: $L = \{ 0^i \mid i$ is a perfect square$\}$ is not regular**

- Proof:
  - By contradiction, let L be regular.
  - PL should apply
  - Let $N$ = PL constant
  - Choose $w=0^{N^2}$
  - By pumping lemma, w=xyz satisfying all three rules
  - By rules (1) & (2), y has between 1 and N 0s
  - By rule (3), any string of the form $xy^kz$ is also in L for all $k\geq 0$
  - Case k=0:
    - #zeros ($xy^0z$) = #zeros (xyz) - #zeros (y)
    - $N^2 - N \leq$ #zeros ($xy^0z$) $\leq N^2 - 1$
    - $\mathbf{(N-1)^2} < N^2 - N \leq$ #zeros ($xy^0z$) $\leq N^2 - 1 < \mathbf{N^2}$
    - $xy^0z \notin L$
    - But the above will complete the proof ONLY IF N>1.
    - … (proof contd.. Next slide)

# Example 3: Pumping Lemma

- (proof contd…)
  - If the adversary pick N=1, then $\mathbf{(N-1)^2} \leq N^2 - N$, and therefore the #zeros($xy^0z$) could end up being a perfect square!
  - This means that pumping down (i.e., setting k=0) is not giving us the proof!
  - So lets try pumping up next…
- Case k=2:
  - #zeros ($xy^2z$) $=$ #zeros ($xyz$) + #zeros ($y$)
  - $N^2 + 1 \leq$ #zeros ($xy^2z$) $\leq N^2 + N$
  - $\mathbf{N^2} < N^2 + 1 \leq$ #zeros ($xy^2z$) $\leq N^2 + N < \mathbf{(N+1)^2}$
  - $xy^2z \notin L$ ↯
  - (Notice that the above should hold for all possible N values of N>0. Therefore, this completes the proof.)

# Closure properties of Regular Languages

Set = $\{1 \ldots n\}$

$1 + 1 = 2$

$1 \times 3 = 3$

$1 \div 2 = 0.5$

Division is not closed

Addition is closed for this set

~~Subtract is closed~~

multiplication

# Closure properties for Regular Languages (RL)

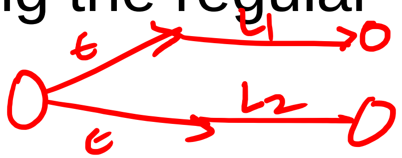
This is different from Kleene closure

- *Closure property:*
  - If a set of regular languages are combined using an operator, then the resulting language is also regular
- Regular languages are *closed* under:
  - Union, intersection, complement, difference
  - Reversal
  - Kleene closure
  - Concatenation
  - Homomorphism
  - Inverse homomorphism

# RLs are closed under union

- IF L and M are two RLs THEN:
  - they both have two corresponding regular expressions, R and S respectively
  - (L U M) can be represented using the regular expression R+S

    $L_1 \cup L_2$



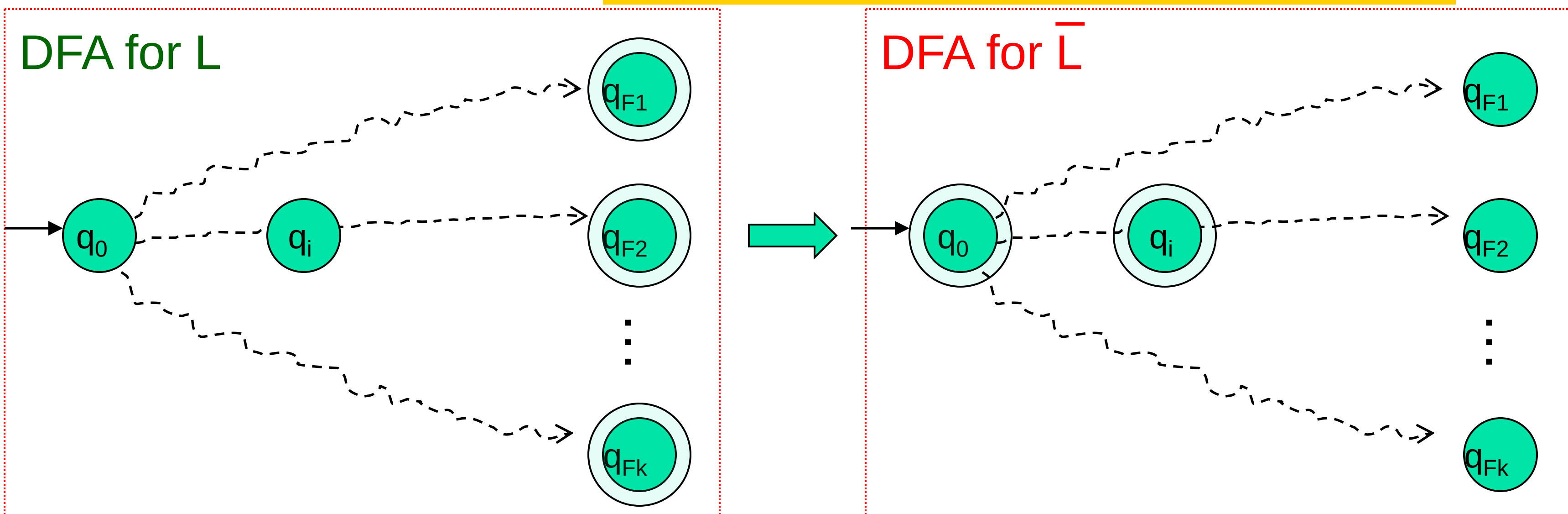

  - Therefore, (L U M) is also regular □

How can this be proved using FAs?

# RLs are closed under complementation

- If L is an RL over $\Sigma$, then $\overline{L}=\Sigma^*-L$
  - To show $\overline{L}$ is also regular, make the following construction

Convert every final state into non-final, and every non-final state into a final state



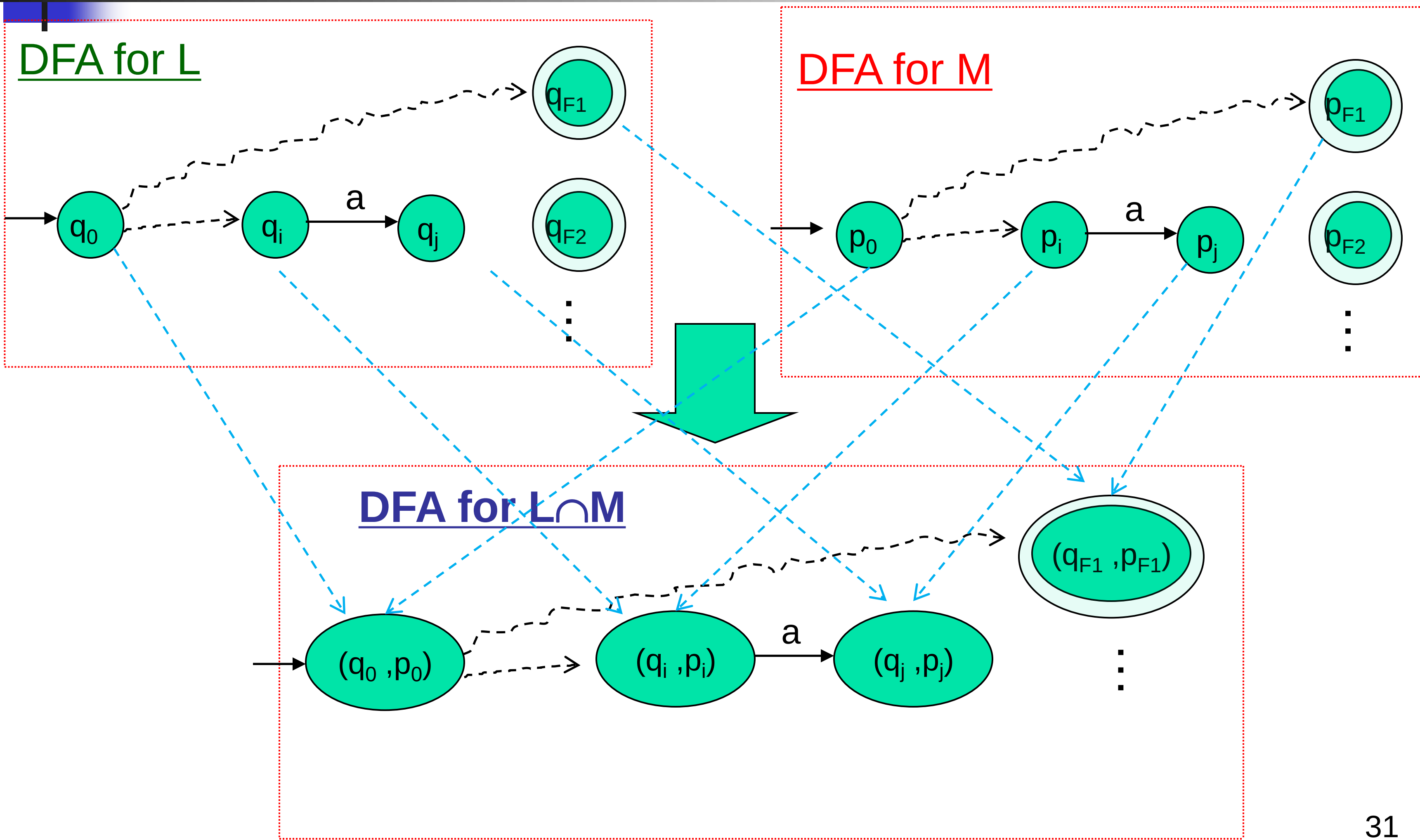
# RLs are closed under intersection

- A quick, indirect way to prove:
  - By DeMorgan's law:
  - $L \cap M = \overline{(\overline{L} \cup \overline{M})}$
  - Since we know RLs are closed under union and complementation, they are also closed under intersection
- A more direct way would be construct a finite automaton for $L \cap M$

# DFA construction for L ∩ M

- $A_L$ = DFA for L = $\{Q_L, \Sigma, q_L, F_L, \delta_L\}$
- $A_M$ = DFA for M = $\{Q_M, \Sigma, q_M, F_M, \delta_M\}$
- Build $A_{L \cap M} = \{Q_L x\ Q_M, \Sigma, (q_L, q_M), F_L x\ F_M, \delta\}$ such that:
  - $\delta((p,q),a) = (\delta_L(p,a), \delta_M(q,a))$, where p in $Q_L$, and q in $Q_M$
- This construction ensures that a string w will be accepted if and only if w reaches an accepting state in <u>both</u> input DFAs.

# DFA construction for L ∩ M



# RLs are closed under set difference

- We observe:
  - $L - M = L \cap \overline{M}$

Closed under intersection

Closed under complementation

- Therefore, L - M is also regular

# RLs are closed under reversal

Reversal of a string w is denoted by $w^R$

- E.g., w=00111, $w^R$=11100

DFA DFA

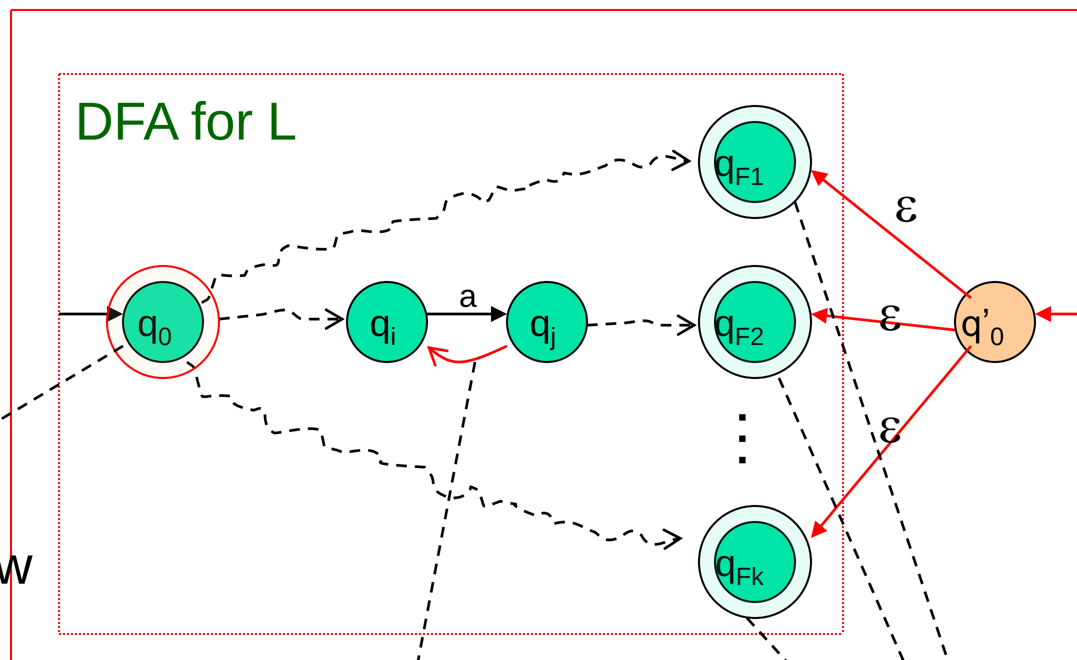
Reversal of a language:

- $L^R$ = The language generated by reversing all strings in L

Theorem: If L is regular then $L^R$ is also regular

# $\varepsilon$ -NFA Construction for $L^R$

New $\varepsilon$-NFA for $L^R$




New start state

Make the old start state as the only new final state

Reverse all transitions

Convert the old set of final states into <u>non-final</u> states

What to do if $q_0$ was one of the final states in the input DFA?

# If L is regular, $L^R$ is regular (proof using regular expressions)

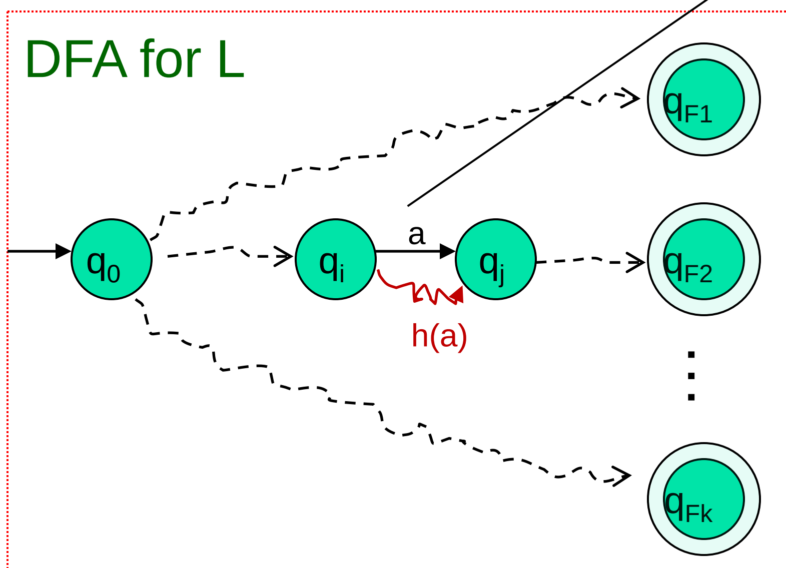
- Let E be a regular expression for L
- Given E, how to build $E^R$?
- <u>Basis:</u> If E= ε, Ø, or a, then $E^R$=E
- <u>Induction:</u> Every part of E (refer to the part as "F") can be in only *one* of the three following forms:
  1. $F = F_1+F_2$
     - $F^R = F_1^R+F_2^R$
  2. $F = F_1F_2$
     - $F^R = F_2^RF_1^R$
  3. $F = (F_1)^*$
     - $(F^R)^* = (F_1^R)^*$

# Homomorphisms

- Substitute each symbol in ∑ (main alphabet) by a corresponding string in T (another alphabet)
  - h: ∑--->T*
- Example:
  - Let ∑={0,1} and T={a,b}
  - Let a homomorphic function h on ∑ be:
    - h(0)=ab, h(1)=ε
  - If w=10110, then h(w) = εabεεab = abab
- In general,
  - $h(w) = h(a_1)\ h(a_2)\dots\ h(a_n)$

Given a DFA for L, how to convert it into an FA for h(L)?

# FA Construction for h(L)




Replace every edge "a" by a path labeled h(a) in the new DFA

- Build a new FA that simulates h(a) for every symbol a transition in the above DFA
- The resulting FA may or may not be a DFA, but will be a FA for h(L)

Given a DFA for M, how to convert it into an FA for $h^{-1}(M)$?

The set of strings in $\Sigma^*$ whose homomorphic translation results in the strings of M

# Inverse homomorphism

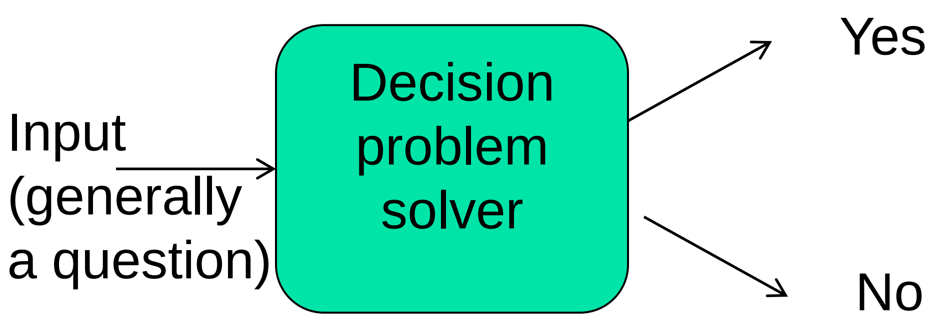
- Let $h: \Sigma \text{--->} T^*$
- Let M be a language over alphabet T
- $h^{-1}(M) = \{w \mid w \in \Sigma^* \text{ s.t., } h(w) \in M\}$

*Claim: If M is regular, then so is $h^{-1}(M)$*

- Proof:
  - Let A be a DFA for M
  - Construct another DFA A' which encodes $h^{-1}(M)$
  - A' is an exact replica of A, except that its transition functions are s.t. for any input symbol *a* in $\Sigma$, A' will simulate *h(a)* in A.
    - $\delta(p,a) = \hat{\delta}(p,h(a))$

# Decision properties of regular languages

<u>Any “decision problem” looks like this:</u>



# Decision properties of regular languages

- Emptiness
- Finiteness
- Equivalence
- Membership

# Emptiness test

- Decision Problem: Is L=Ø ?
- FA accepts empty language or not.
- It is a decidable problem as algorithm exist
- Approach:

On a DFA for L:

1. From the start state, run a *reachability* test, which returns:
   1. success: if there is at least one final state that is reachable from the start state
   2. failure: otherwise
2. L=Ø if and only if the reachability test fails

# Membership question

- Decision Problem: Given L, is w in L?
- Possible answers: Yes or No
- Approach:
  1. Build a DFA for L
  2. Input w to the DFA
  3. If the DFA ends in an accepting state, then yes; otherwise no.

# Finiteness

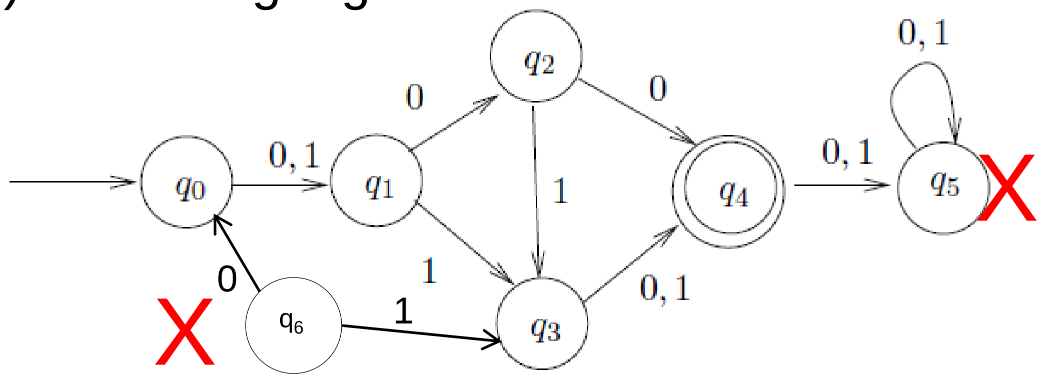
- <u>Decision Problem:</u> Is L finite or infinite?
- <u>Approach:</u>

  On a DFA for L:

  1. Remove all states unreachable from the start state
  2. Remove all states that cannot lead to any accepting state.
  3. After removal, check for cycles in the resulting FA
  4. L is finite if there are no cycles; otherwise it is infinite

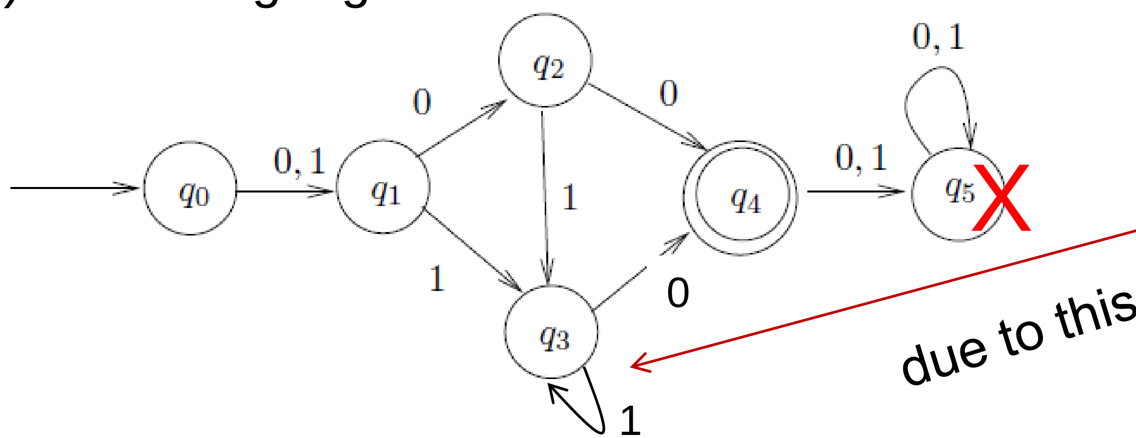
# Finiteness test - examples

Ex 1) Is the language of this DFA finite or infinite?




FINITE

Ex 2) Is the language of this DFA finite or infinite?




INFINITE

# Equivalence & Minimization of DFAs

# Applications of interest

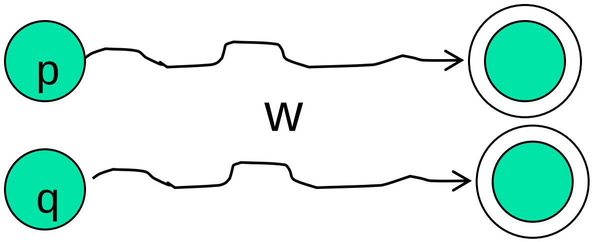
- Comparing two DFAs:
  - $L(DFA_1) == L(DFA_2)$?

- How to minimize a DFA?
  1. Remove unreachable states
  2. Identify & condense equivalent states into one

# When to call two states in a DFA “equivalent”?

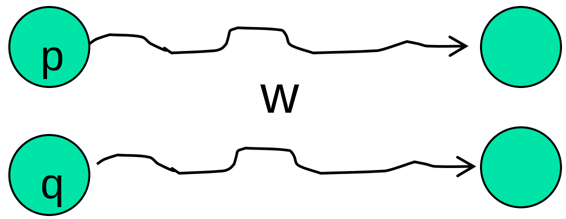
Two states p and q are said to be *equivalent* iff:

i) Any string w accepted by starting at p is also accepted by starting at q;




AND

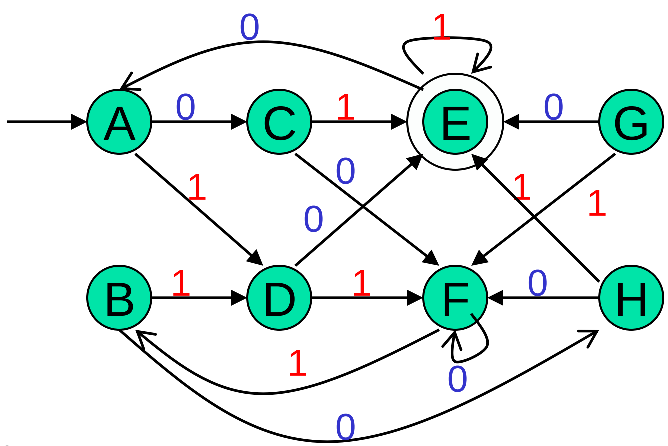
i) Any string w rejected by starting at p is also rejected by starting at q.




➔ p≡q

# Computing equivalent states in a DFA

Table Filling Algorithm




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | = | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | X | X | X | X | X | = | | |
| G | X | X | X | = | X | X | = | |
| H | X | X | = | X | X | X | X | = |

Pass #0
1. Mark accepting states ≠ non-accepting states

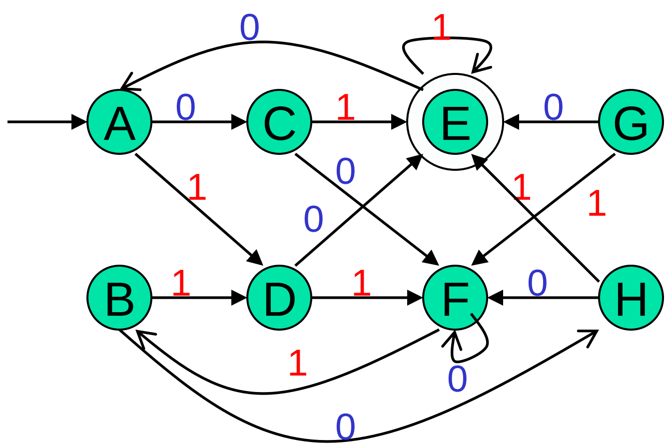
Pass #1
1. Compare every pair of states
2. Distinguish by one symbol transition
3. Mark = or ≠ or blank(tbd)

Pass #2
1. Compare every pair of states
2. Distinguish by up to two symbol transitions (until different or same or tbd)

….

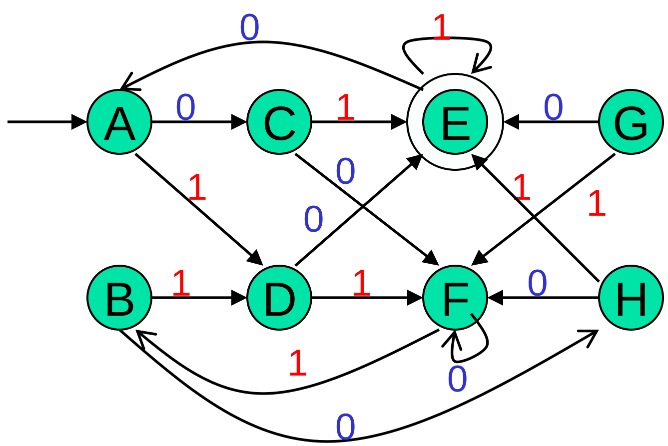
(keep repeating until table complete)

# Table Filling Algorithm - step by step




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | | | = | | | | | |
| D | | | | = | | | | |
| E | | | | | = | | | |
| F | | | | | | = | | |
| G | | | | | | | = | |
| H | | | | | | | | = |
| | A | B | C | D | E | F | G | H |

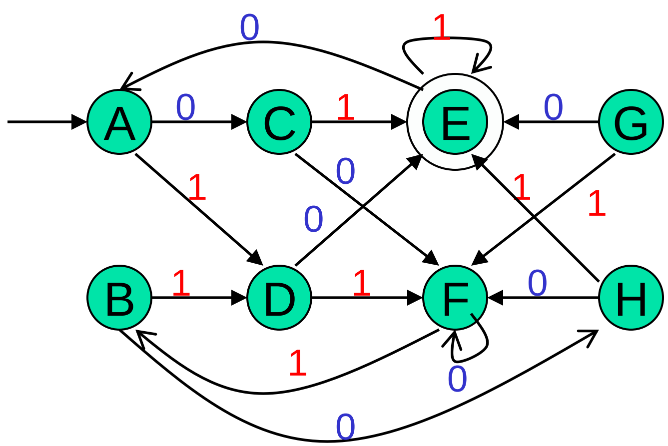
# Table Filling Algorithm - step by step




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | | | = | | | | | |
| D | | | | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | | | X | = | | |
| G | | | | | X | | = | |
| H | | | | | X | | | = |

1. Mark X between accepting vs. non-accepting state

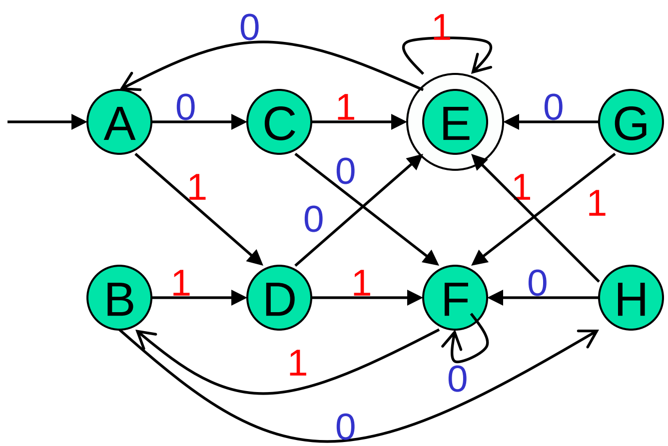
# Table Filling Algorithm - step by step




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | X | | = | | | | | |
| D | X | | | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | | | X | = | | |
| G | X | | | | X | | = | |
| H | X | | | | X | | | = |

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

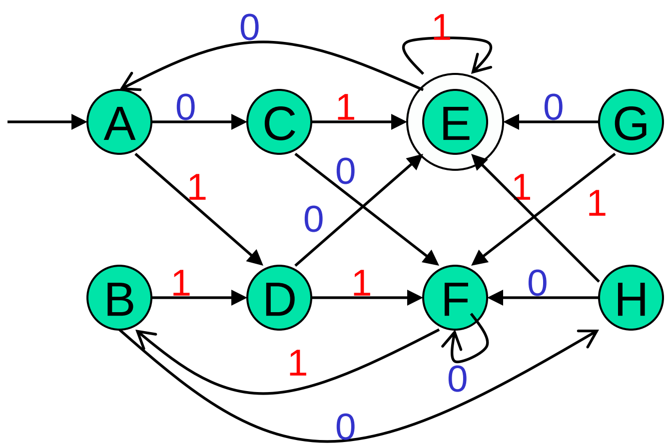
# Table Filling Algorithm - step by step




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | | | X | = | | |
| G | X | X | | | X | | = | |
| H | X | X | | | X | | | = |
| | A | B | C | D | E | F | G | H |

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

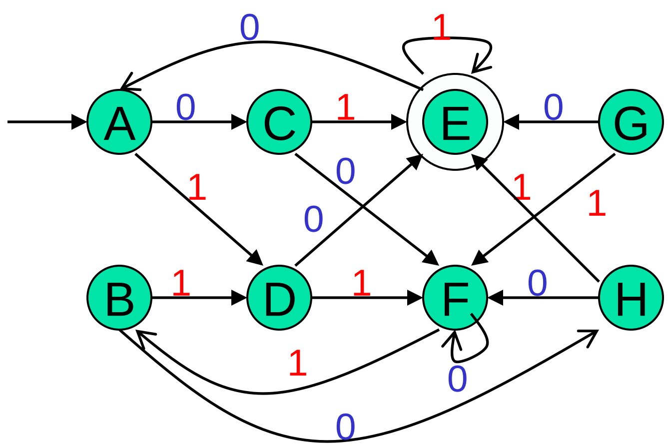
# Table Filling Algorithm - step by step




1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | X | | X | = | | |
| G | X | X | X | | X | | = | |
| H | X | X | = | | X | | | = |
| | A | B | C | D | E | F | G | H |

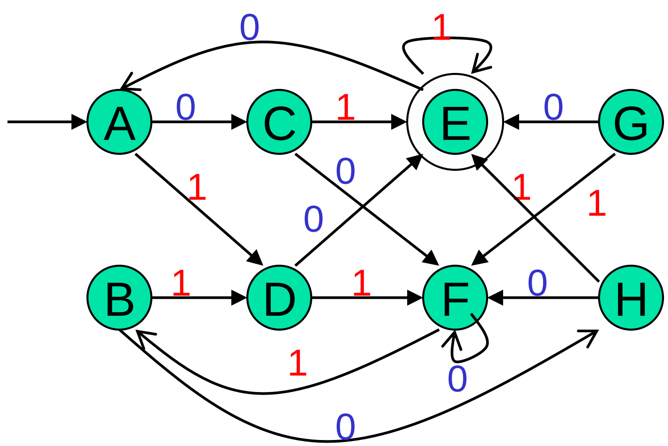
# Table Filling Algorithm - step by step




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | X | X | X | = | | |
| G | X | X | X | = | X | | = | |
| H | X | X | = | X | X | | | = |

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

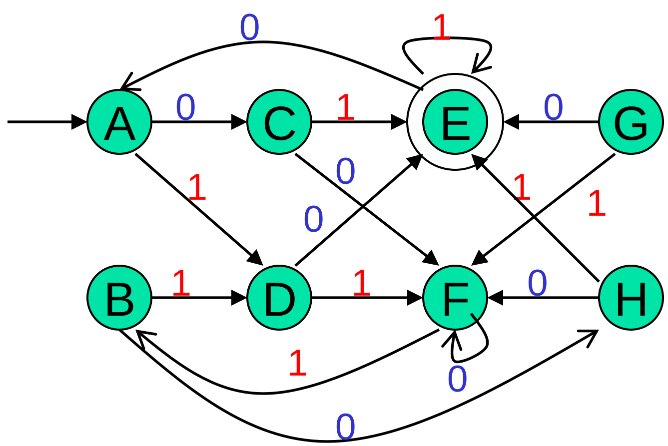
# Table Filling Algorithm - step by step




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | X | X | X | = | | |
| G | X | X | X | = | X | X | = | |
| H | X | X | = | X | X | X | | = |

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

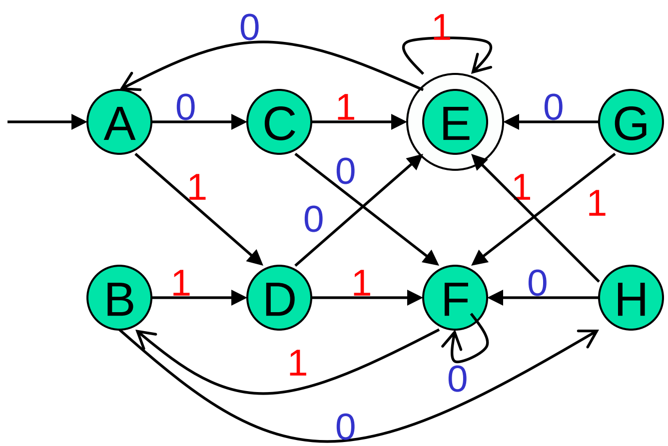
# Table Filling Algorithm - step by step




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | | | X | X | X | = | | |
| G | X | X | X | = | X | X | = | |
| H | X | X | = | X | X | X | X | = |

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

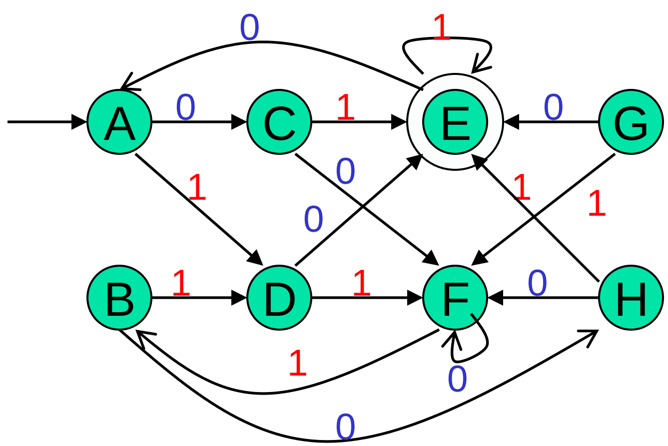
# Table Filling Algorithm - step by step




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | **=** | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | **X** | **X** | X | X | X | = | | |
| G | X | X | X | **=** | X | X | = | |
| H | X | X | **=** | X | X | X | X | = |
| | A | B | C | D | E | F | G | H |

1. Mark X between accepting vs. non-accepting state
2. Pass 1:
   Look 1- hop away for distinguishing states or strings
3. Pass 2:
   Look 1-hop away again for distinguishing states or strings
   continue….

# Table Filling Algorithm - step by step

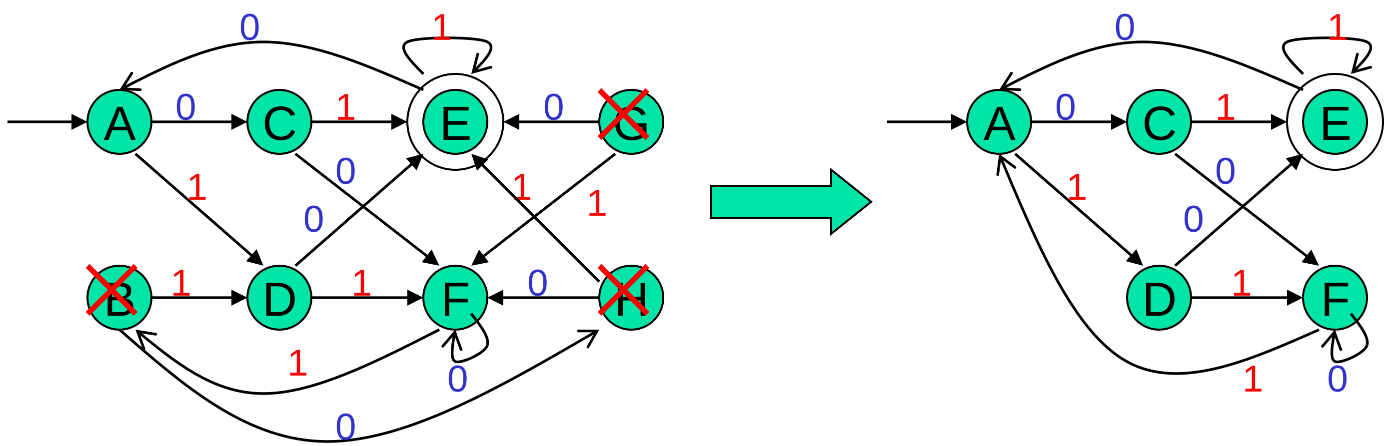



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | = | = | | | | | | |
| C | X | X | = | | | | | |
| D | X | X | X | = | | | | |
| E | X | X | X | X | = | | | |
| F | X | X | X | X | X | = | | |
| G | X | X | X | = | X | X | = | |
| H | X | X | = | X | X | X | X | = |

1. Mark X between accepting vs. non-accepting state
2. Pass 1:
   Look 1- hop away for distinguishing states or strings
3. Pass 2:
   Look 1-hop away again for distinguishing states or strings
   continue….

Equivalences:
- A=B
- C=H
- D=G

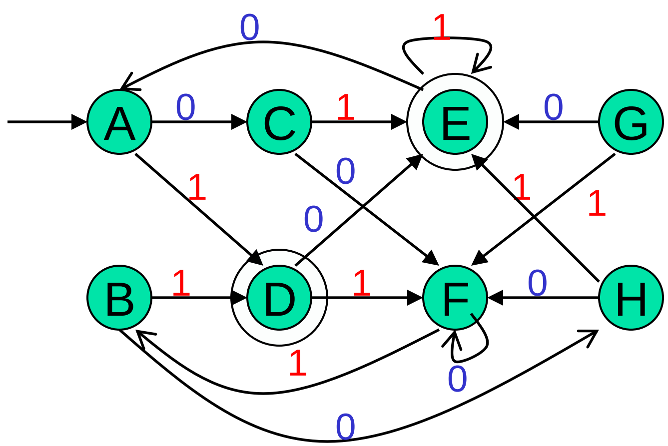
# Table Filling Algorithm - step by step




Retrain only one copy for
each equivalence set of states

Equivalences:
- A=B
- C=H
- D=G

# Table Filling Algorithm – special case




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | = | | | | | | | |
| B | | = | | | | | | |
| C | | | = | | | | | |
| D | | | | = | | | | |
| E | | | | ? | = | | | |
| F | | | | | | = | | |
| G | | | | | | | = | |
| H | | | | | | | | = |

Q) What happens if the input DFA has more than one final state?
Can all final states initially be treated as equivalent to one another?

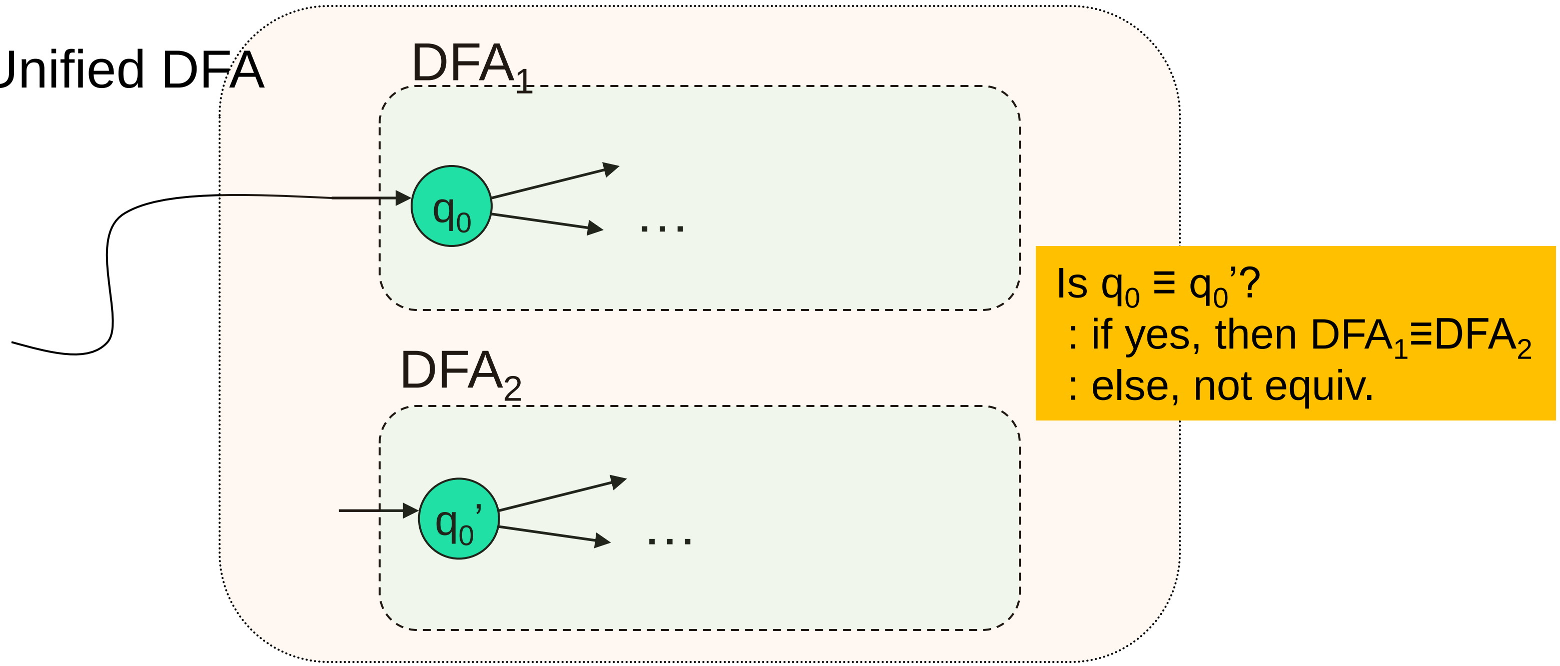
# How to minimize a DFA?

- Goal: Minimize the number of states in a DFA
- Algorithm:
  1. Eliminate states unreachable from the start state
  2. Identify and remove equivalent states
  3. Output the resultant DFA

Depth-first traversal from the start state

Table filling algorithm

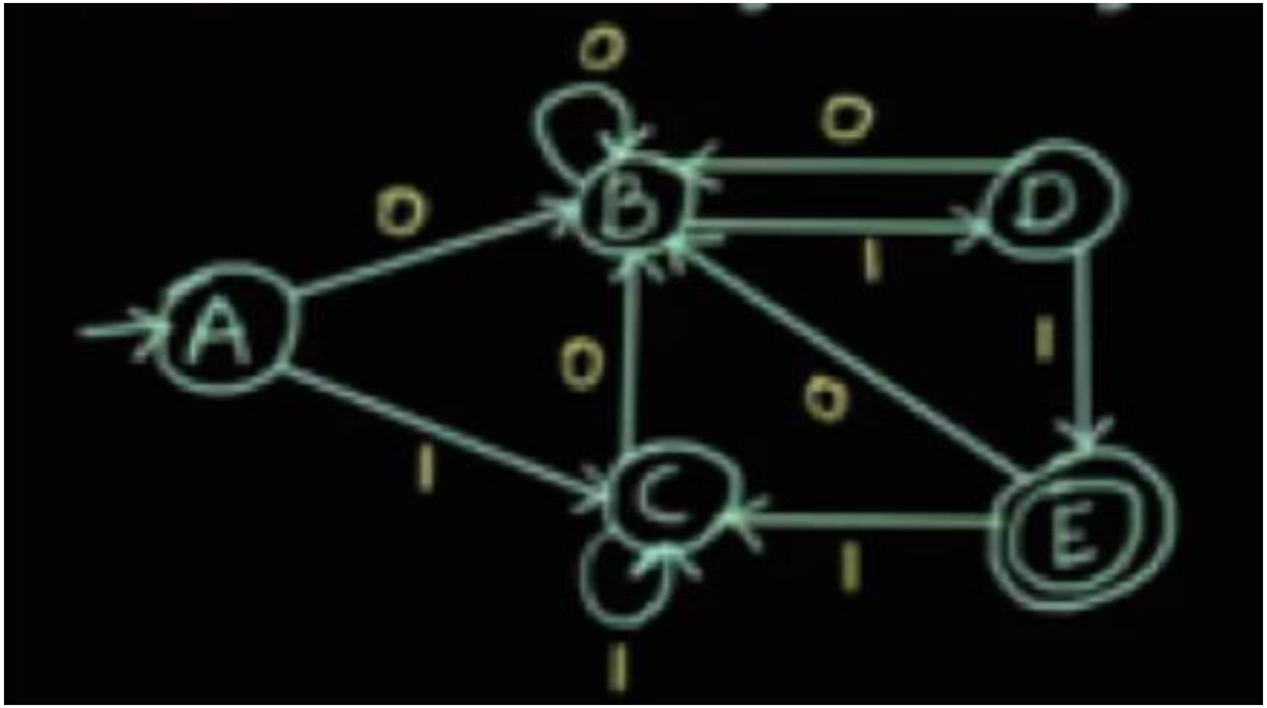
# Are Two DFAs Equivalent?

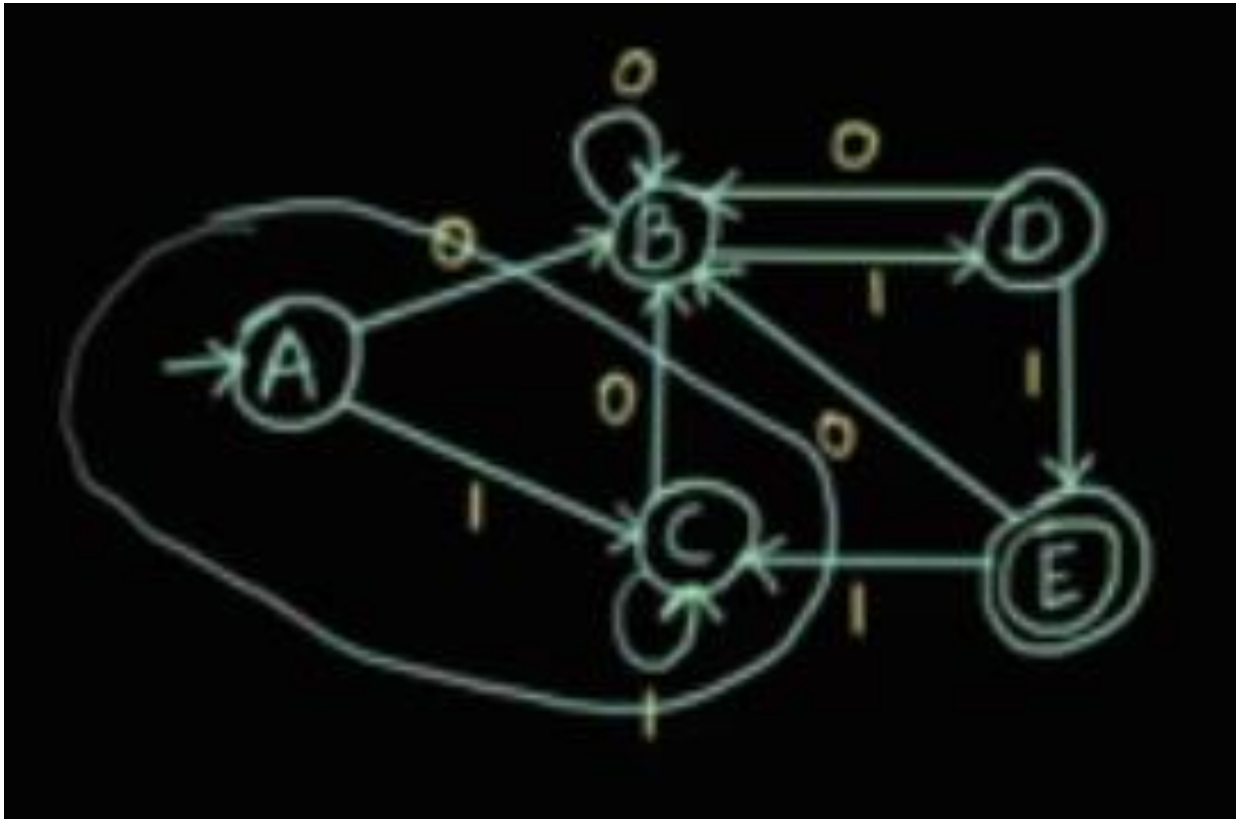



1. Make a new dummy DFA by just putting together both DFAs
2. Run table-filling algorithm on the unified DFA
3. *IF* the start states of both DFAs are found to be equivalent,

   *THEN:* $DFA_1 \equiv DFA_2$

   *ELSE:* different

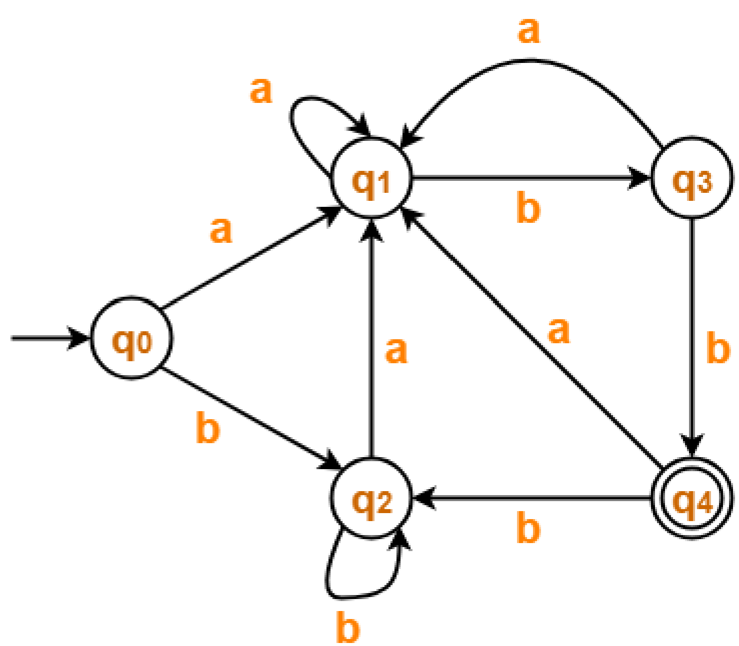

A
B
C
D
E
0
1

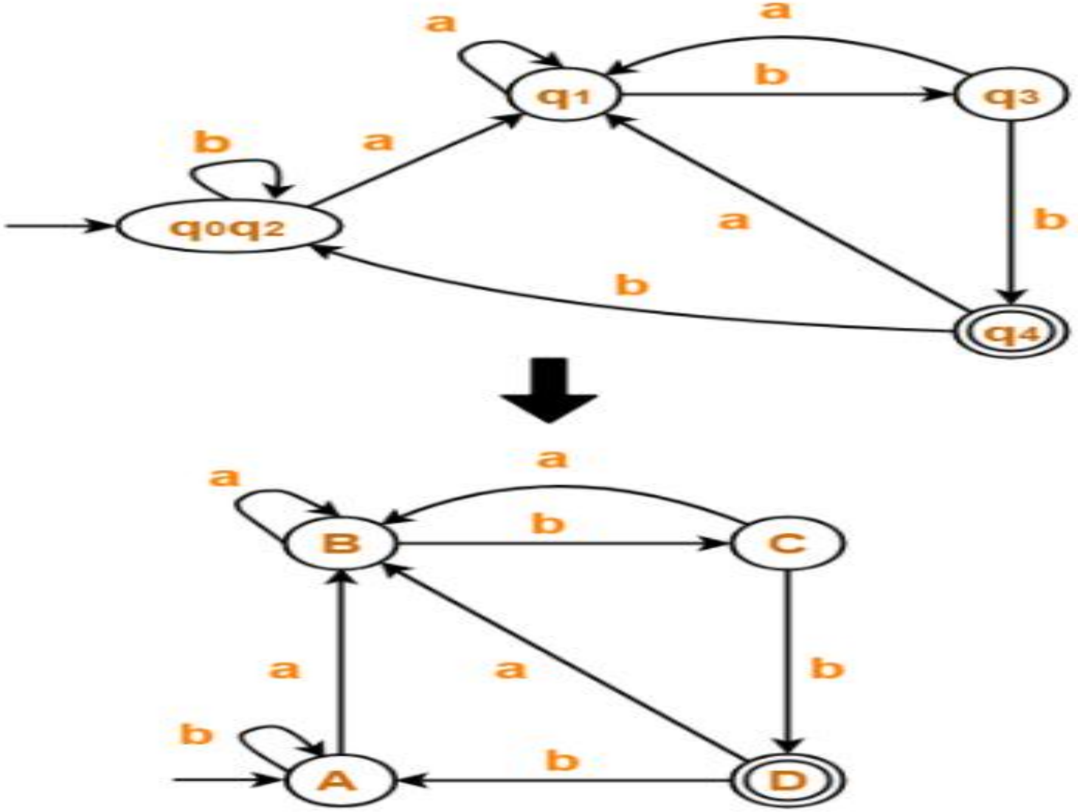

A
B
C
D
E
0
0
0
0
0
1
1
1
1
1

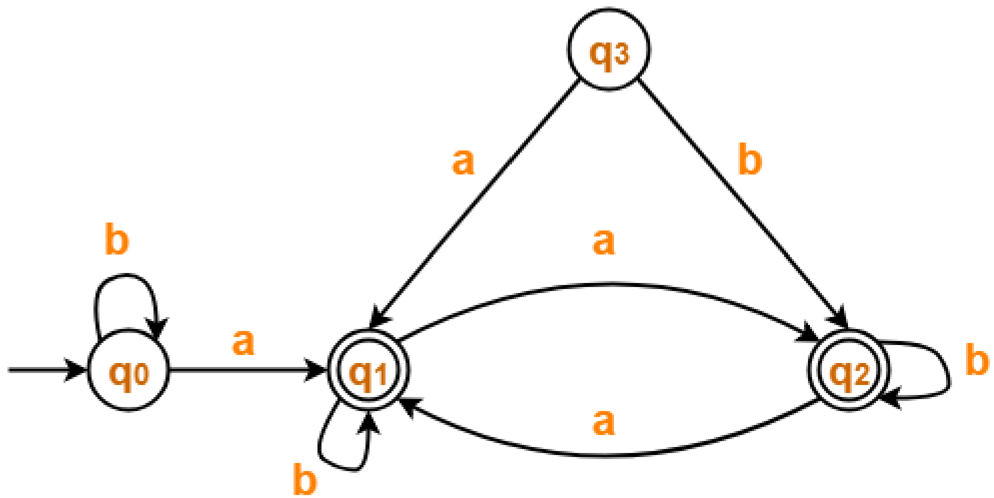
# Myhill-Nerode Theorem
# (table filling method)

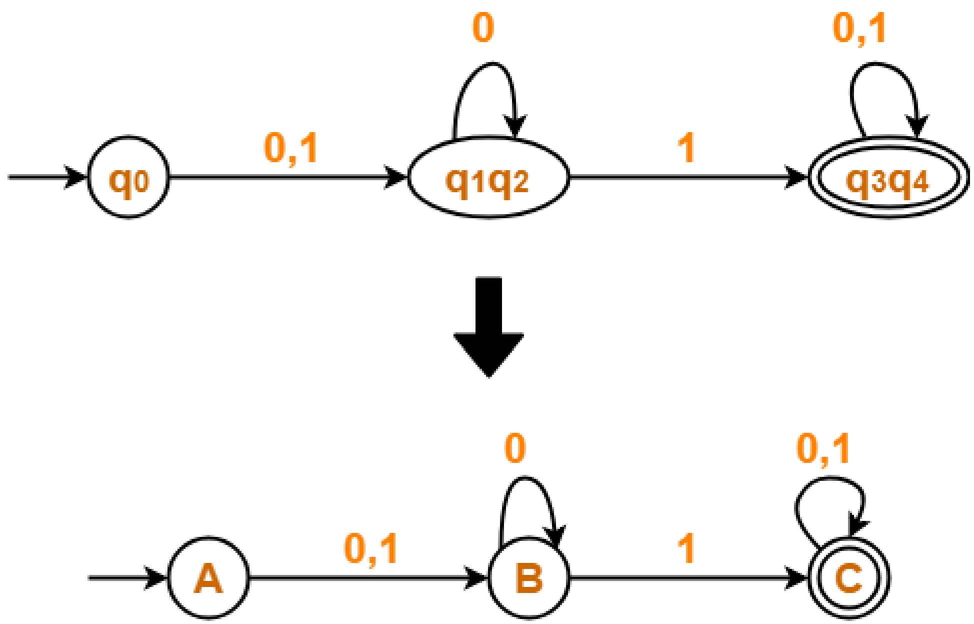
<u>Steps</u>:

1) Draw a table for all pairs of states (P,Q)
2) Mark all pairs where $P \in F$ and $Q \notin F$
3) If there are any Unmarked pairs (P,Q) such that [ $\delta$(P,x), $\delta$(Q,x)] is marked, then mark [P,Q] where 'x' is an input symbol
   REPEAT THIS UNTIL NO MORE MARKINGS CAN BE MADE
4) Combine all the Unmarked Pairs and make them a single state in the minimized DFA


a
a
q1
b
q3
a
a
q0
a
b
b
q2
q4
b
b


a
a
b
b
a
q0q2
q1
q3
a
b
b
q4
a
a
b
B
C
a
a
b
b
b
A
D


q3
a
b
b
a
q0
a
q1
q2
b
b
a



Minimal DFA

# Summary

- How to prove languages are not regular?
  - Pumping lemma & its applications

- Closure properties of regular languages

- Simplification of DFAs
  - How to remove unreachable states?
  - How to identify and collapse equivalent states?
  - How to minimize a DFA?
  - How to tell whether two DFAs are equivalent?