Properties of Regular Languages

UNIT-II

Topics

1) Closure properties of regular languages
2) How to prove whether a given language is regular or not?

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

Now, lets prove all of this!

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
  - Therefore, (L U M) is also regular

How can this be proved using FAs?

Closure Properties

Suppose $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ are finite automata accepting L_1 and L_2, respectively. Let M be the FA $(Q, \Sigma, q_0, A, \delta)$, where

$$Q = Q_1 \times Q_2$$
$$q_0 = (q_1, q_2)$$

and the transition function δ is defined by the formula

$$\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$$

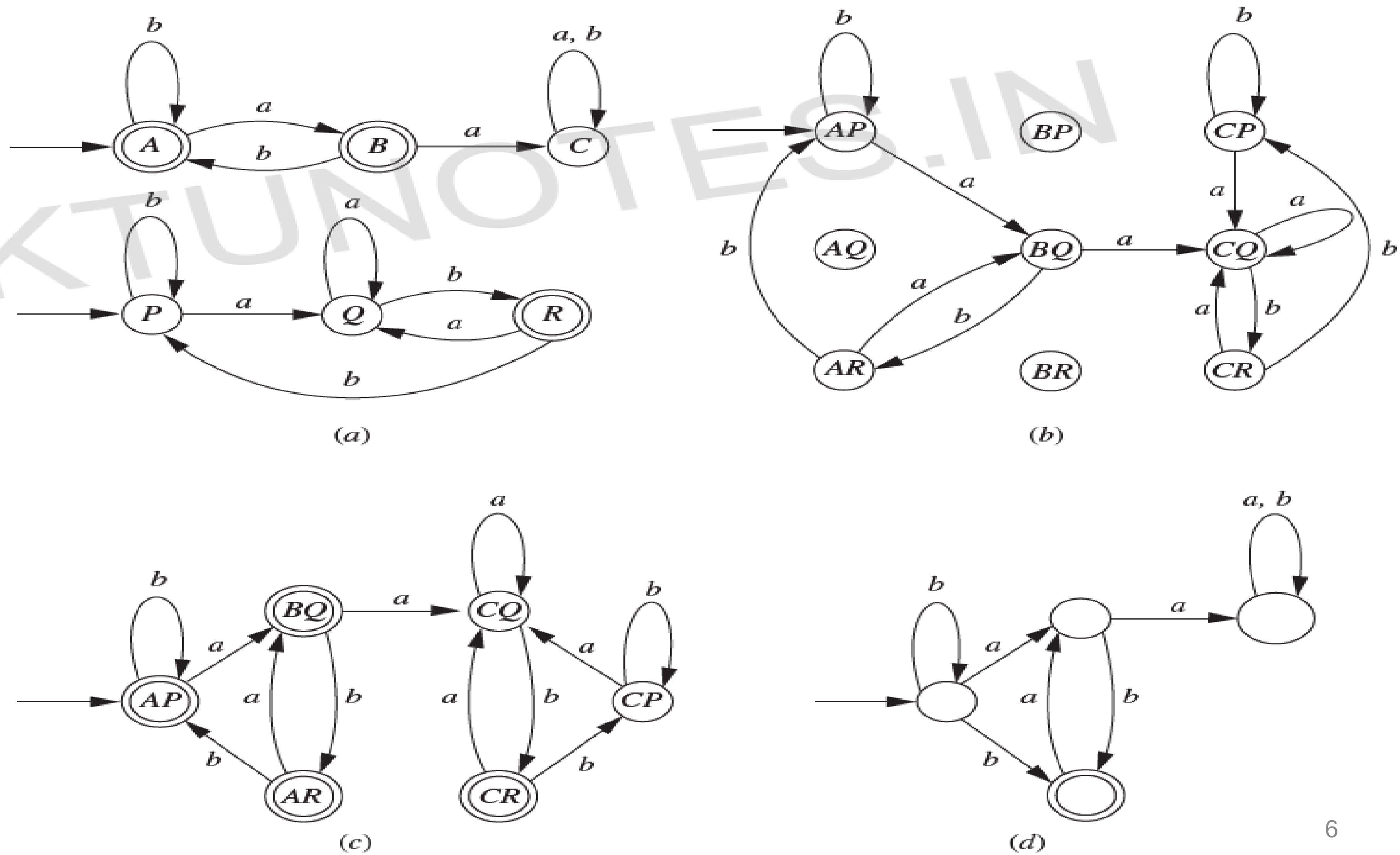
for every $p \in Q_1$, every $q \in Q_2$, and every $\sigma \in \Sigma$. Then

1. If $A = \{(p, q) \mid p \in A_1 \text{ or } q \in A_2\}$, M accepts the language $L_1 \cup L_2$.
2. If $A = \{(p, q) \mid p \in A_1 \text{ and } q \in A_2\}$, M accepts the language $L_1 \cap L_2$.
3. If $A = \{(p, q) \mid p \in A_1 \text{ and } q \notin A_2\}$, M accepts the language $L_1 - L_2$.

RV College of Engineering

Union and Intersection

Go, change the world

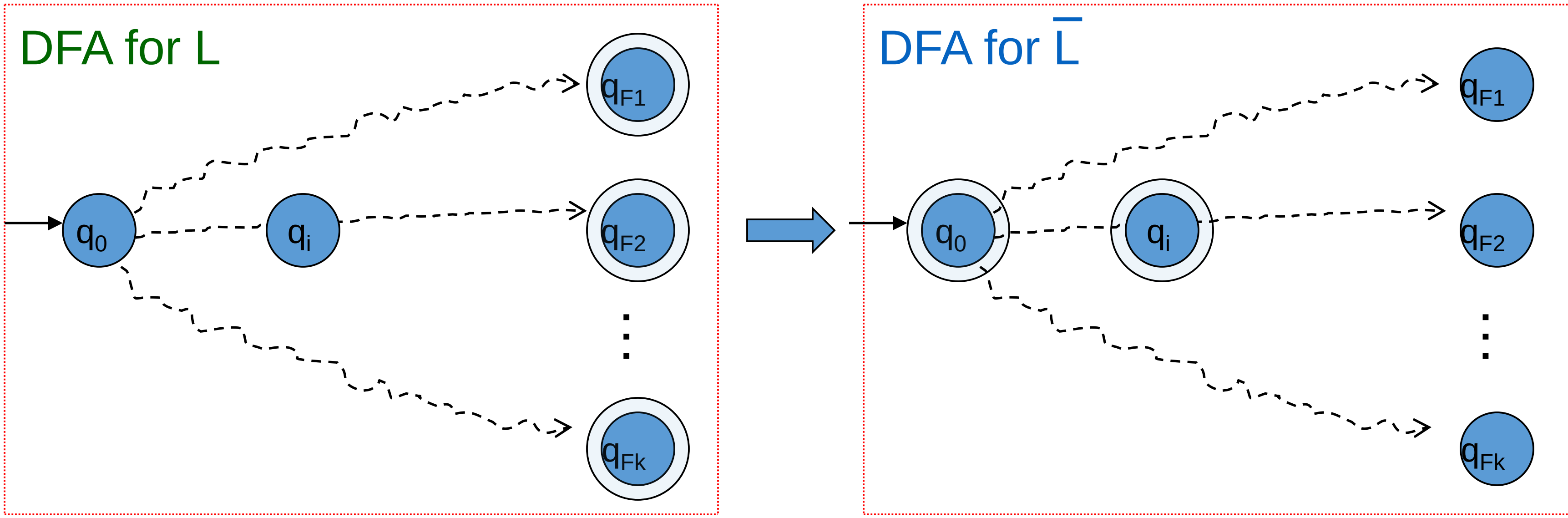


RLs are closed under complementation

- If L is an RL over ∑, 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



Assumes q0 is a non-final state. If not, do the opposite.

RLs are closed under intersection

- A quick, indirect way to prove:
  - By DeMorgan's law:
  - $L \cap M = (L \cup M)$
  - 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

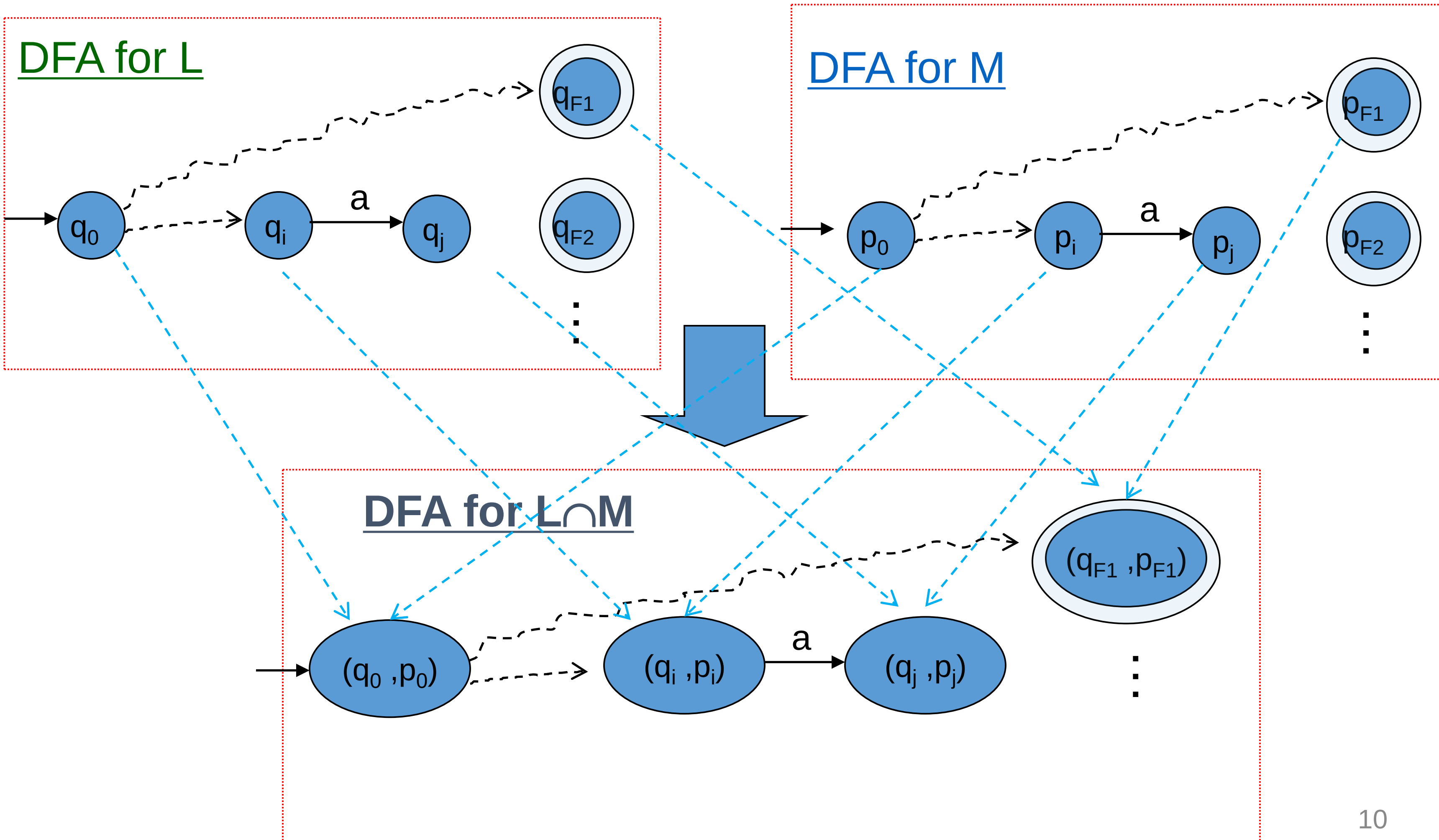
Go, change the world

- 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 $A_{L \cap M} = \{Q_L \times Q_M, \Sigma, (q_L, q_M), F_L \times F_M, \delta\}$ 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 ∩ 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

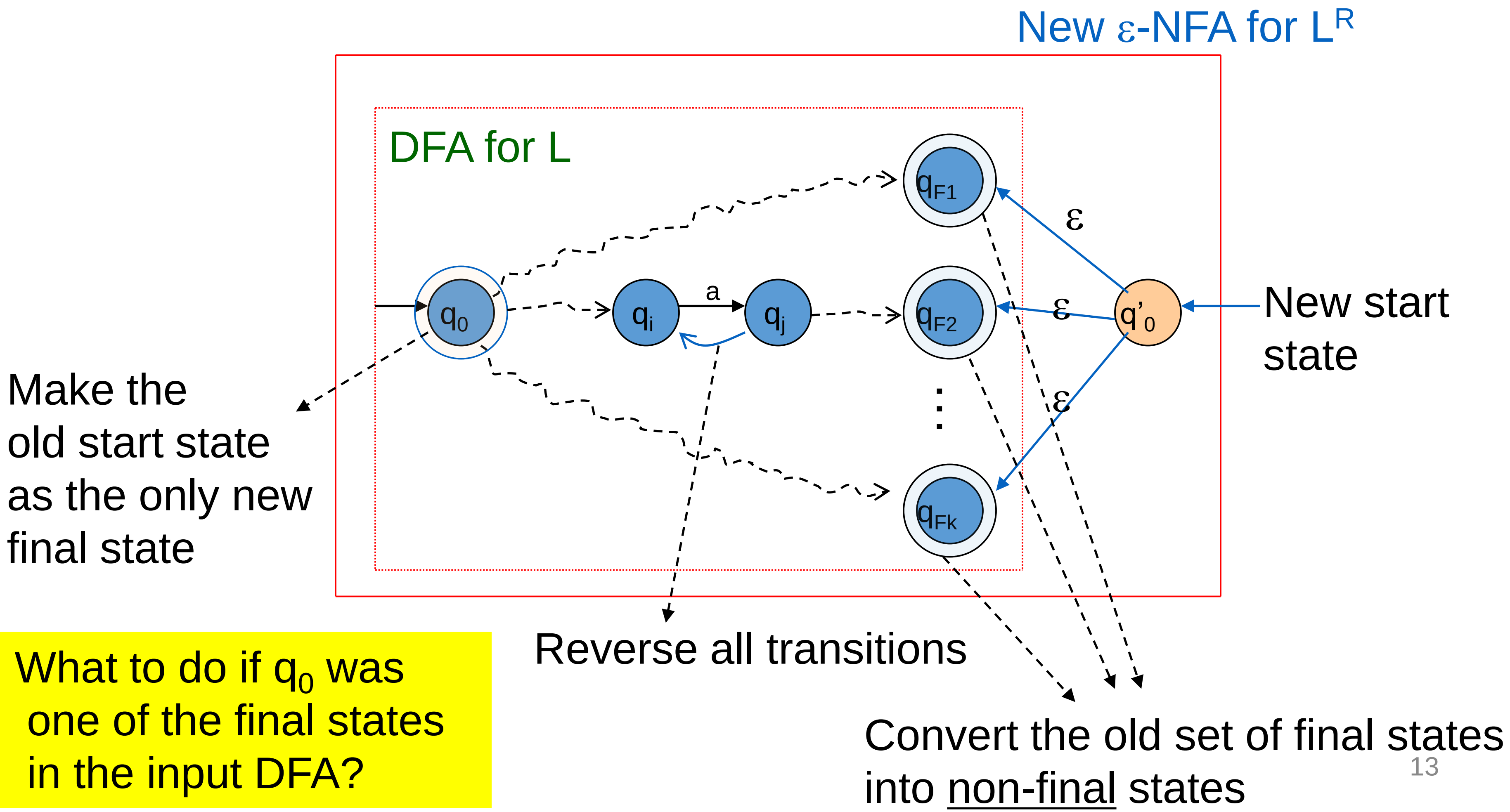
- E.g., w=00111, w^R=11100

<u>Reversal of a language:</u>

- L^R = The language generated by reversing <u>all</u> strings in L

<u>Theorem:</u> If L is regular then L^R is also regular

ε -NFA Construction for L^R



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?
- Basis: If E= ε, Ø, or a, then E^R=E
- Induction: 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. $F = (F_1)^*$
     - $(F^R)^* = (F_1^R)^*$

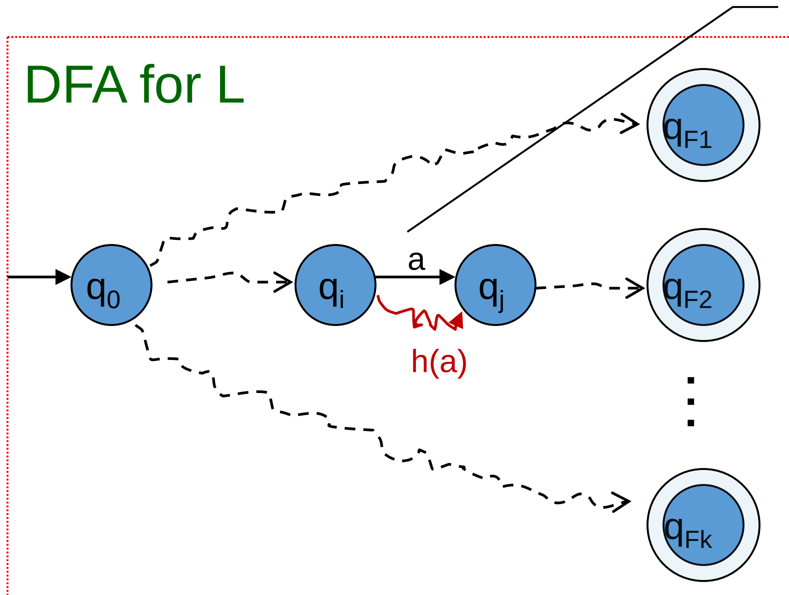
Homomorphisms

- 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)?

Go, change the world

FA Construction for h(L)



Replace every <u>edge</u> "a" by a <u>path</u> 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 $\sum^*$ whose homomorphic translation results in the strings of M

Inverse homomorphism

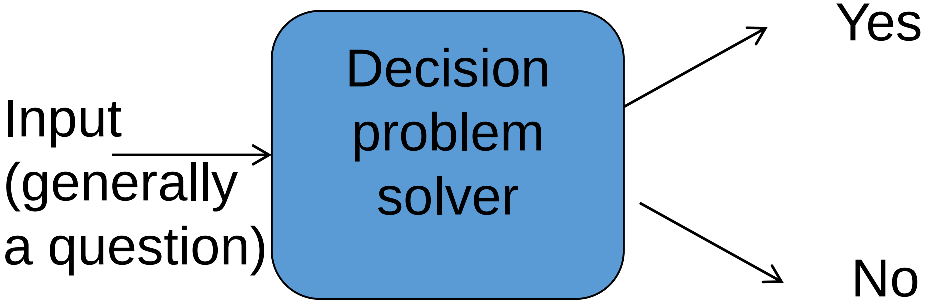
- Let h: $\sum$--->T*
- Let M be a language over alphabet T
- $h^{-1}(M) = \{w \mid w \in \sum^* \text{ s.t., } h(w) \in M\}$

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 $\sum$, A' will simulate *h(a)* in A.
    - $\delta(p,a) = \hat{\delta}(p,h(a))$

Decision properties of regular languages

Any "decision problem" looks like this:



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.

Emptiness test

- Decision Problem: Is L=Ø ?
- 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

How to implement the reachability test?

Finiteness

- Decision Problem: Is L finite or infinite?
- Approach:

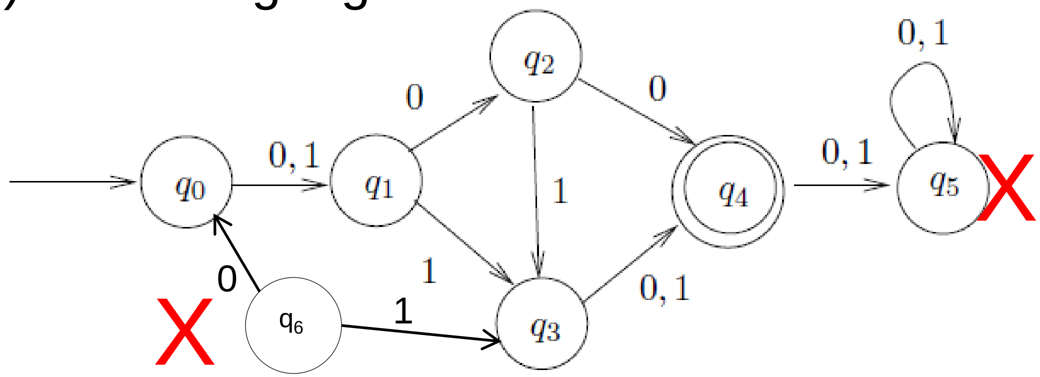
  On a DFA for L:

  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
- Another approach
  - Build a regular expression and look for Kleene closure

How to implement steps 2 and 3?

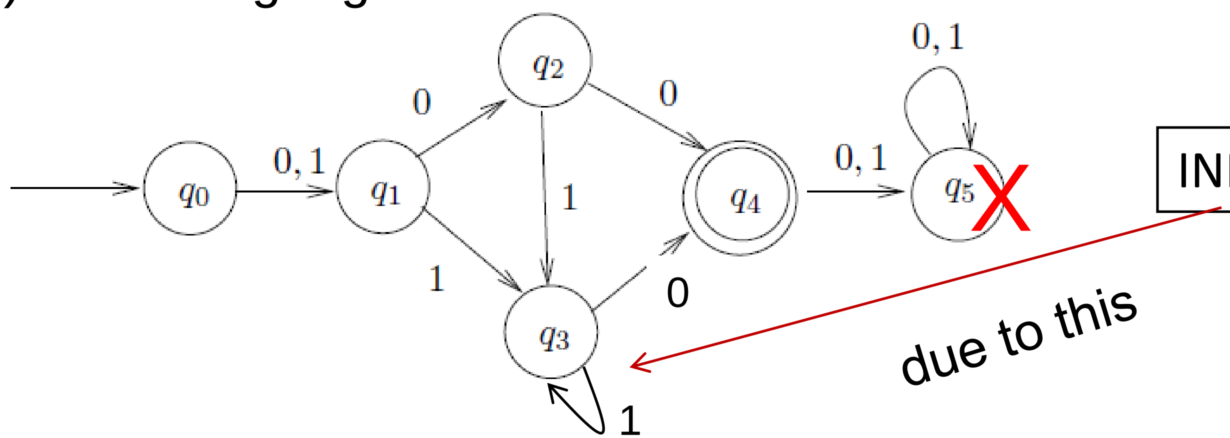
Finiteness test - examples

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



FINITE

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



INFINITE

Some languages are *not* regular

When is a language is regular?
if we are able to construct one of the following: DFA *or* NFA *or* ε -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?

> *"The hardest thing of all is to find a black cat in a dark room, especially if there is no cat!" -Confucius*

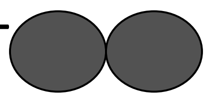
Example of a non-regular language

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

- *Hypothesis:* *L is not regular*
- Intuitive rationale: How do you keep track of a running count in an FA?
- A more formal rationale:
  - By contradition, if L is regular then there should exist a DFA for
  - 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

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 $i < j$
- ==> We can fool the DFA by inputing $0^{(k-(j-i))}1^k$ and still get it to accept (note: k-(j-i) is at most k-1).
- ==> DFA accepts strings w/ unequal number of 0s and 1s, implying that the DFA is wrong!

The Pumping Lemma for Regular Languages

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

Pumping Lemma for Regular Languages

Let L be a regular language

Then *there exists* some constant ***N*** such that *for every* string $w \in L$ s.t. $|w| \geq N$, *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.

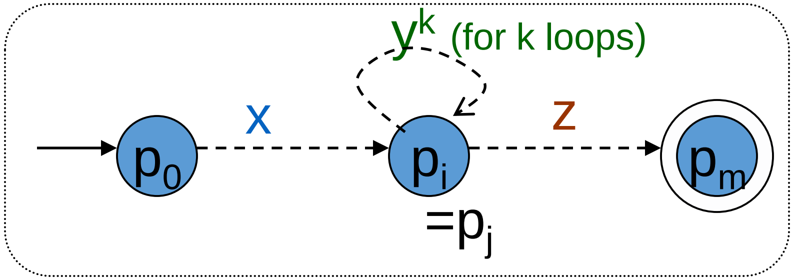
Definition: *N* is called the "Pumping Lemma Constant"

Pumping Lemma: Proof

- L is regular => it should have a DFA.
  - Set N := number of states in the DFA
- Any string $w \in L$, s.t. $|w| \geq N$, should have the form:
  $w = a_1 a_2 \ldots a_m$, where $m \geq N$
- Let the states traversed after reading the first N symbols be: $\{p_0, p_1, \ldots p_N\}$
  - ==> There are N+1 p-states, while there are only N DFA states
  - ==> at least one state has to repeat
    i.e, $p_i = p_J$ where $0 \leq i < j \leq N$ (by PHP)

Pumping Lemma: Proof…

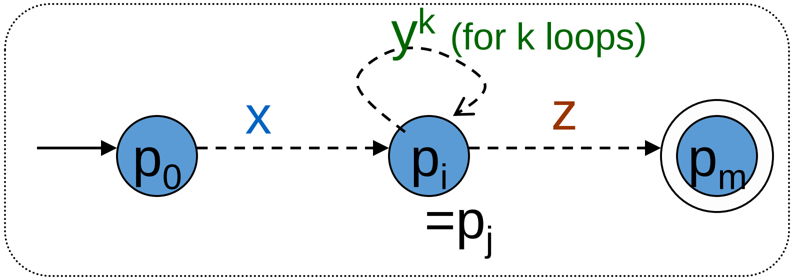
- => We should be able to break $w=xyz$ as follows:
  - $x=a_1a_2..a_i$; $y=a_{i+1}a_{i+2}..a_J$; $z=a_{J+1}a_{J+2}..a_m$
  - x's path will be $p_0..p_i$
  - y's path will be $p_i\, p_{i+1}..p_J$ (but $p_i=p_J$ implying a loop)
  - z's path will be $p_Jp_{J+1}..p_m$
- Now consider another string $w_k=xy^kz$, where $k\geq 0$
- Case k=0
  - DFA will reach the accept state p_m
- Case k>0
  - DFA will loop for y^k, and finally reach the accept state p_m for z
- In either case, $w_k \in L$



This proves part (3) of the lemma

Pumping Lemma: Proof…

- For part (1):
  - Since i<j, $y \neq \varepsilon$



- For part (2):
  - By PHP, the repetition of states has to occur within the first N symbols in w
  - ==> $|xy| \leq N$

□

The Purpose of the Pumping Lemma for RL

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

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 P/L constant
3. (we) Using N, choose a string $w \in L$ s.t.,
    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 P/L 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)

- <u>What the Adversary does?</u>
  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.
  - P/L constant should exist → adv.
    - Let *N* = that P/L constant
  - Consider input w = 0^N1^N → you
    (your choice for the template string)
  - By pumping lemma, we should be able to break w=xyz, such that: →you
    1) y≠ ε
    2) |xy|≤N
    3) For all k≥0, the string xy^kz is also in L

Template string $w = 0^N1^N = \underbrace{00 \ldots 0}_{N}\underbrace{11 \ldots 1}_{N}$

Go, change the world

Proof…

- Because $|xy| \leq N$, 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 P/L (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 become exceed the #1s

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 | i is a perfect square} is not regular

- Proof:
  - By contradiction, let L be regular.
  - P/L should apply
  - Let *N* = P/L 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≥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.)

Example 4

Show that $F = \{ww|w \in \{0,1\}^*\}$ is nonregular using pumping lemma

Proof: Assume that F is regular and p is its pumping length.

Consider $s = 0^p 1 0^p 1 \in F$. Since $|s| > p$, $s = xyz$ and satisfi esthe conditions of the pumping lemma.

Example 5

Show that $D = \{1^{n^2} | n \geq 0\}$ is nonregular.

Proof by contradiction: Assume that D is regular and let p be its pumping length. Consider $s = 1^{p^2} \in D$, $|s| \geq p$. Pumping lemma guarantees that s can be split, $s = xyz$, where for all $i \geq 0$, $xy^iz \in D$

Example 6

- We illustrate this using pumping lemma to prove that $E = \{0^i 1^j | i > j\}$ is not regular
- Proof: by contradiction using pumping lemma. Assume that E is regular and its pumping length is p.
- Let $s = 0^{p+1} 1^p$; From decomposition $s = xyz$, from condition 3, $|xy| \leq p$ it results that y consists only of 0s.