San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu
www.cs.sjsu.edu/~yazdankhah

Non-Regular Languages

(Part 1)

Lecture 24

Day 26/31

CS 154

Formal Languages and Computability

Fall 2019

Agenda of Day 26

- Solution and Feedback of Quiz 9
- Summary of Lecture 23
- Lecture 24: Teaching ...
  - Non-Regular Languages (Part 1)

Solution and Feedback of Quiz 9 (Out of 19)

Section	Average	High Score	Low Score
01 (TR 3:00 PM)	14.29	18	9
02 (TR 4:30 PM)	14.02	18	7
03 (TR 6:00 PM)	15.77	18	12

Summary of Lecture 23: We learned ...

Parse-Trees

- Parse-tree is ...
  - ... an ordered-tree that can be constructed for every string by using the grammar.

Parsers Algorithms

- There are two types of algorithms for parsers:
  - Top-down and bottom-up
- Exhaustive parsing algorithm is ...
  - ... a top-down algorithm that check all possible derivations to find a derivation sequence for a given string.

Exhaustive Parsing Algorithm

- This algorithm has two serious problems:
  - It is extremely inefficient: $O(|P|^{2|w|+1})$
  - It is possible that it never terminates.
- Two good news:
  1. Theorem: there exists an efficient algorithm for every CFG with complexity $O(|w|^3)$.
  2. If we use s-grammar, the efficiency would be $O(|w|)$.

Any Question

Summary of Lecture 23: We learned ...

S-Grammar

- A simple grammar is ...
  - ... a CFG with two restrictions:
    1. All production rules are of the form
       $A \rightarrow av$
       where $A \in V$, $a \in T$, $v \in V^*$

       One terminal as prefix and any number of variables as suffix.
    2. Any pair (A, x) occurs only once in all production rules.
- Note that there is no λ .

Ambiguity in Grammars

- Ambiguity of grammars ...

  ... happens when for some strings in the language, we can construct two or more parse-tree.

Any Question

Objective of This Lecture

- We defined "regular languages" as ...

  A language is called regular if there exists a ...

  - ... DFA/NFA to accept it.
  - ... REGEX to represent it.
  - ... regular grammar to generate it.
- But the most interesting languages are non-regular.
- The main question of this lecture is:

  How TO PROVE a language is NON-REGULAR?

- Obviously, we cannot say:

  L is non-regular because I CANNOT construct a DFA/NFA/REGEX/regular grammar for it!

Objective of This Lecture

- Before, we learned a heuristic technique to figure out a language was non-regular.
  - We looked at the language's strings pattern and if it needed some kind of memory or counter, then it could not be regular.

- But this is NOT a mathematical proof!
- Also, in some cases, we might make mistake.
  - e.g.: L = {w : w has an equal number of ab and ba} is regular!

- So, in this lecture we are looking for a ...
  ... solid technique to prove a language is NON-REGULAR.
- At first, we introduce an important property of infinite regular languages.

A Property of Infinite Regular Languages

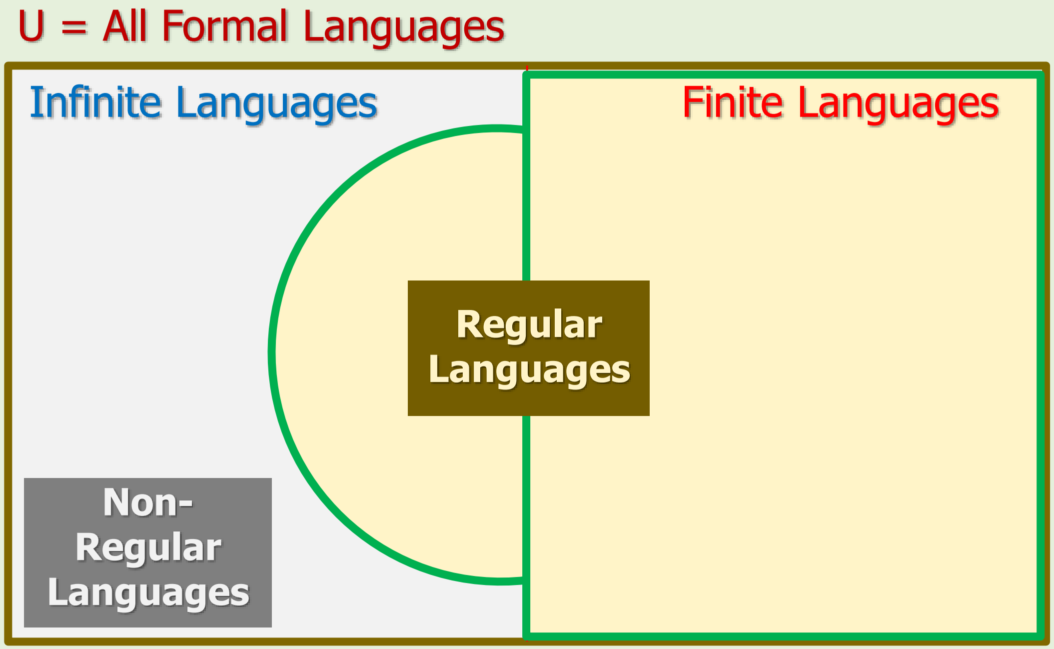
Required Background

1. The concept of regular and non-regular languages
2. Proof by contradiction
3. Cycle and simple cycle definitions in graphs
4. One-dimensional projection of a walk
5. Pigeonhole principle

   (will be covered shortly!)

Regular and Non-Regular Languages

Recap



Proof by Contradiction

Recap from Math 42

- Logically, proving a theorem means to assume the truth of some statements (e.g.: p) and entailing the truth of another statement (e.g.: q)
- Sometimes, it is hard to follow this path.
- In these cases, we might use the following logical equivalency:

Contrapositive

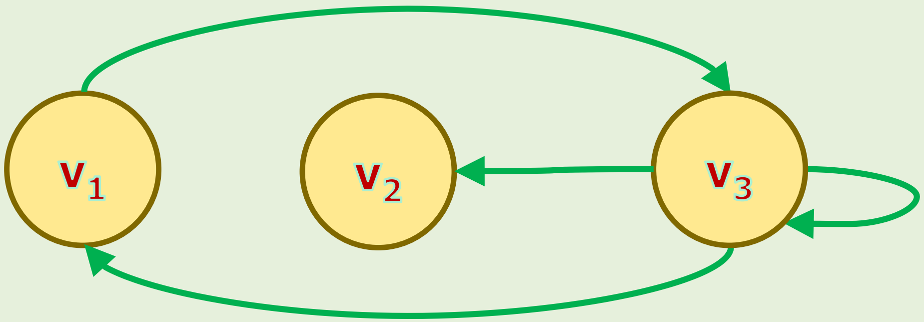
$$p \rightarrow q \equiv \sim q \rightarrow \sim p$$

- In fact, we prove that if the negation of the desired result (e.g. ~q) is true, then it leads to a contradiction.
- And to resolve this contradiction, we have no choice except blaming our assumption that was "~q is true". Therefore, $q \equiv T$.
- This technique is called "proof by contradiction".

Cycle

Recap

- A walk from a vertex (called base) to itself with no repeated edges.
- But: Walk + No repeated edges = path
- Rewording: A cycle is a path from a vertex (called base) to itself.



Examples 1

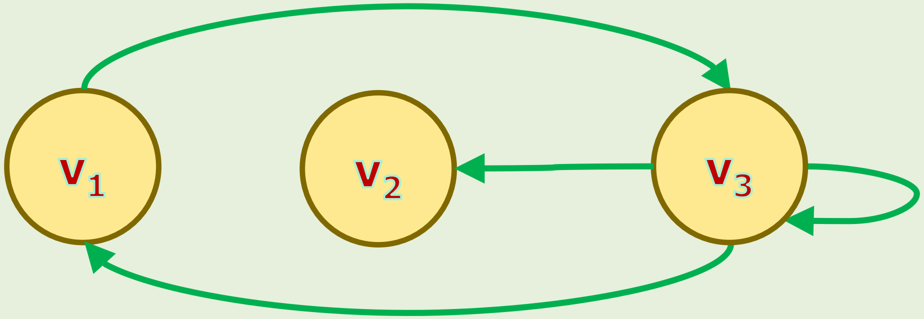
- Walk 1: $(v_3, v_1), (v_1, v_3)$
- Walk 2: $(v_1, v_3), (v_3, v_3), (v_3, v_1)$
- Walk 3: (v_3, v_3)

Simple Cycle

Recap

- A cycle that no vertex other than the base is repeated.

- In other words, in a simple cycle, all vertices (except the base) and all edges are visited uniquely.



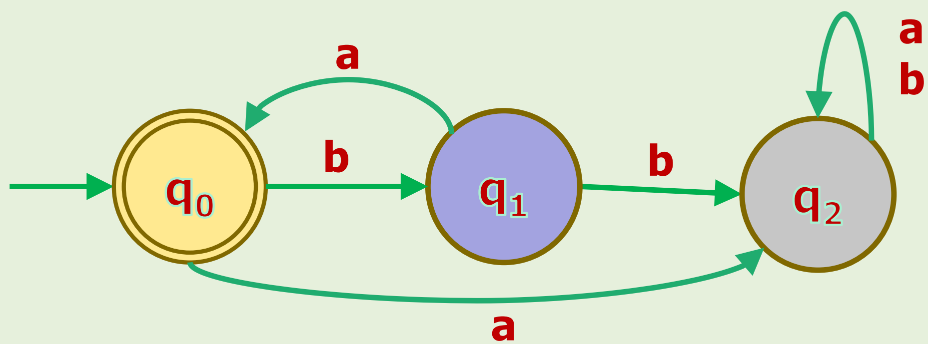
Examples 2

- Walk 1: $(v_1, v_3), (v_3, v_1)$
- Walk 2: $(v_3, v_1), (v_1, v_3)$
- Walk 3: (v_3, v_3)

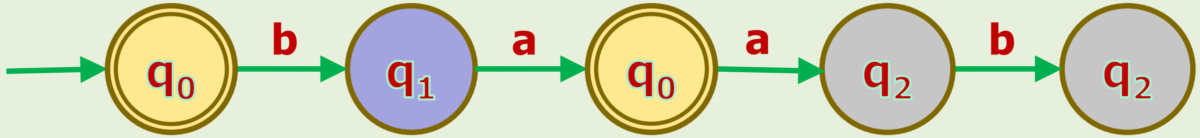
One-Dimensional Projection of a String

Example 3

- Given following DFA with 3 states over $\Sigma = \{a, b\}$:



- Show one-dimensional projection of w = baab.



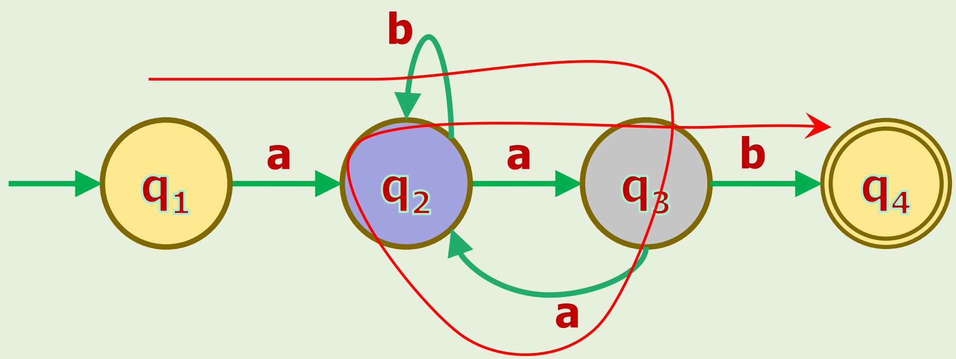
- Every string has its own one-dimensional projection.
- Note that $w \notin L$. How do we know that?

One-Dimensional Projection of a Walk

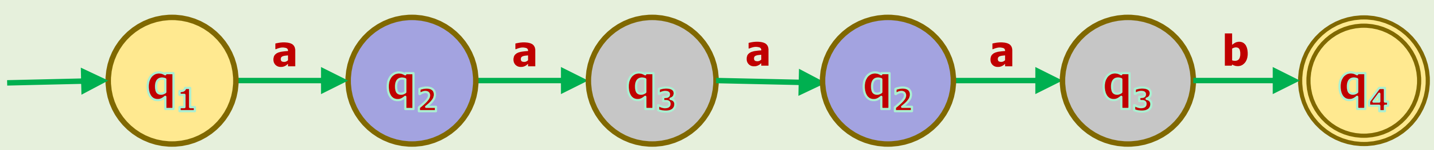
Recap

Example 4

- Given following NFA with 4 states over $\Sigma = \{a, b\}$:



- Show one-dimensional projection of w = aaaab.



- In this example, q_2 is the first repeated state.

Pigeonhole Principle

Pigeonhole Principle

Recap

Example 5

- If we have 10 pigeons and 9 pigeonholes (boxes), then one pigeonhole must contain more than one pigeon.

Pigeonhole Principle

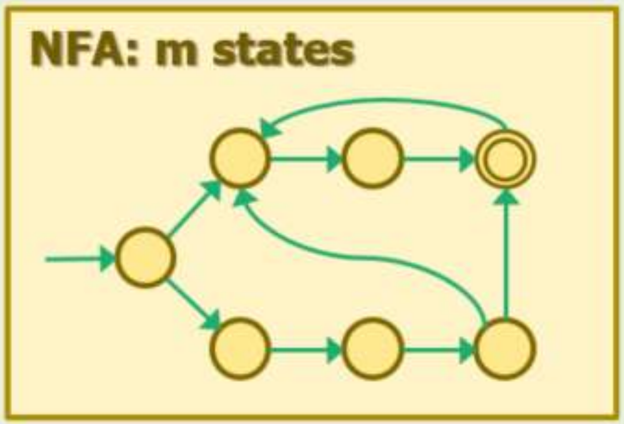
If we put n objects (pigeon) into m boxes (pigeonholes) &&

n > m

∴ At least one box must contain more than one object.

- Reference: https://en.wikipedia.org/wiki/Pigeonhole_principle

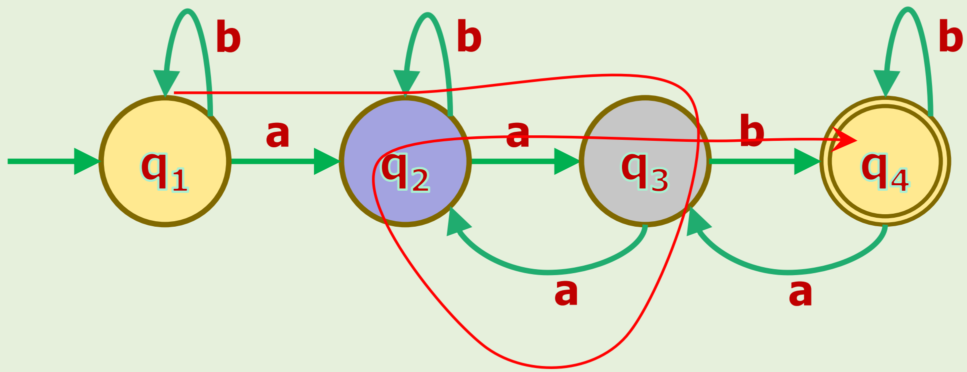
What is Pigeonhole Principle and DFAs Relationship!



Pigeonhole Principle and DFAs Relationship

Example 6

- Given following DFA with 4 states.



- Consider the walk of w = aaaab. (|w| = 5)

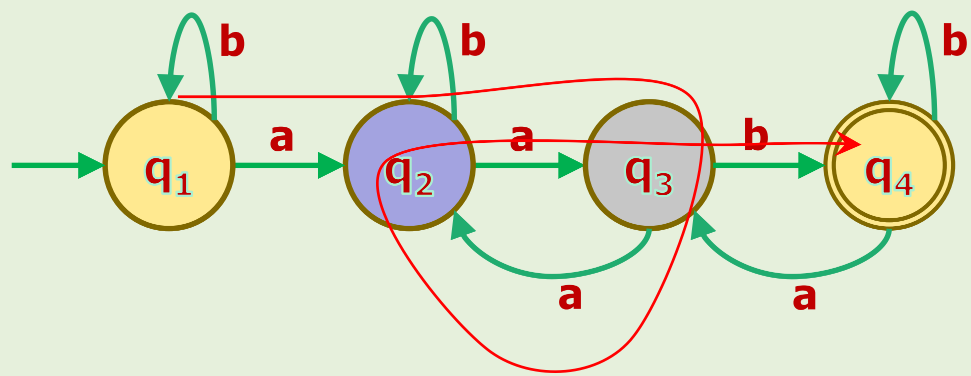
- Can we conclude that:

At least one state must be visited more than once.

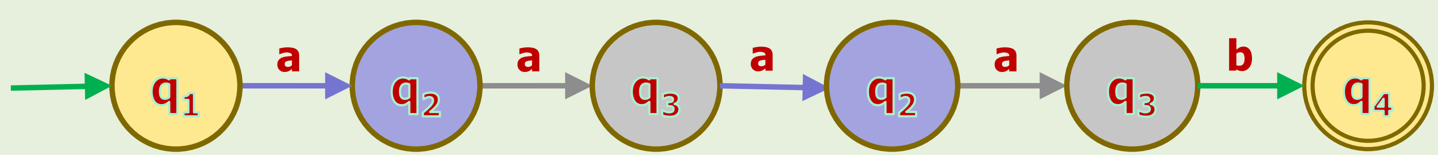
- Yes, because the size of the string is bigger than the number of states.

Pigeonhole Principle and DFA's

Example 6 (cont'd) w = aaaab



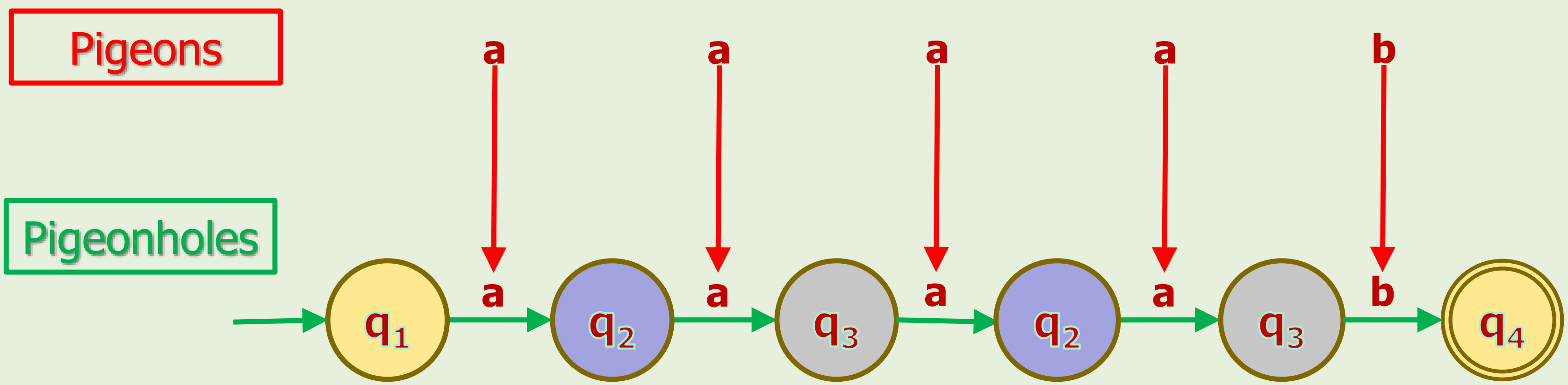
- Now, let's show the walk by one-dimensional projection method to investigate our guess.



- q_2 and q_3 are visited twice.

What is Pigeon and what is Pigeonhole?

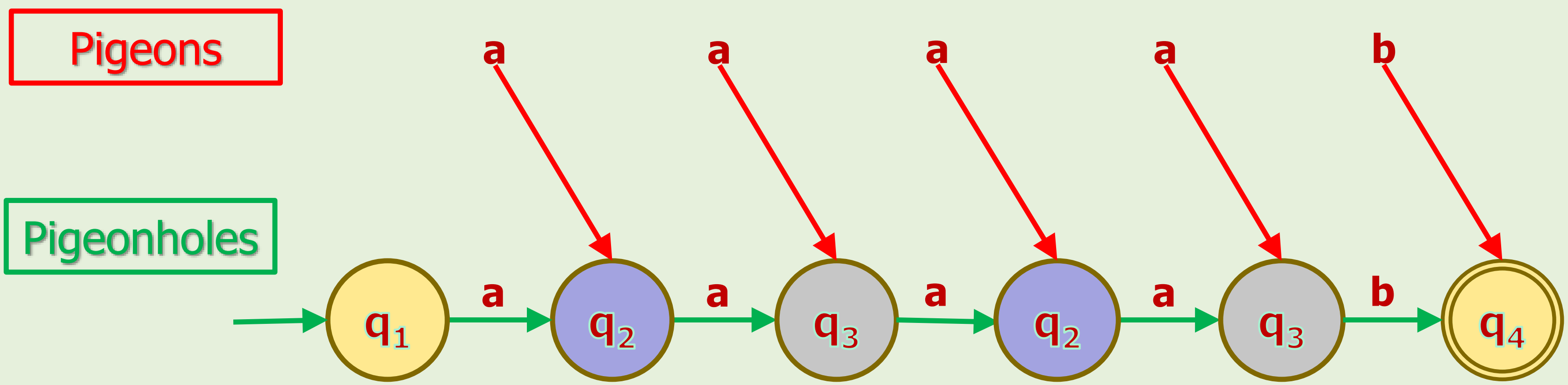
- Pigeons are the symbols of the string w = aaaab.
- Pigeonholes are the transitions.



- The edges don't look like HOLES!
- But states do!

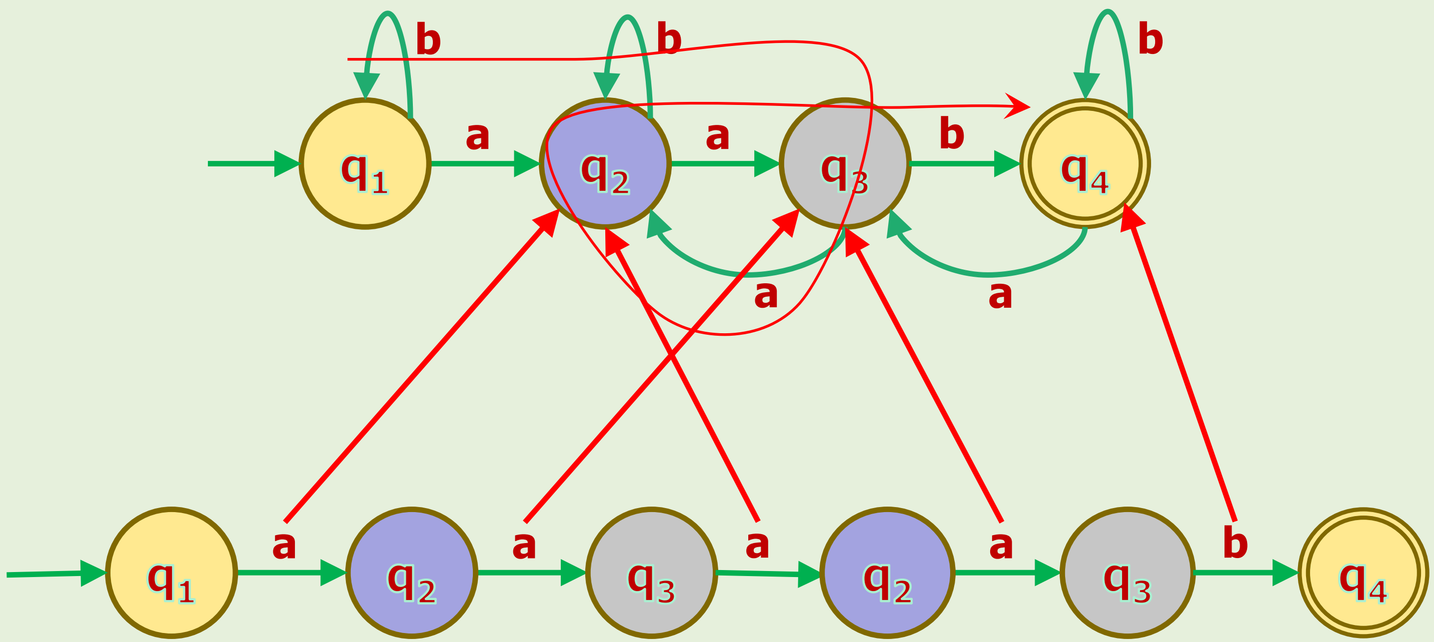
What is Pigeon and what is Pigeonhole?

- The states give us a better feeling of pigeonholes!
- Therefore, we might consider states as pigeonholes.



- Note that in one-dimensional projection, q_0 (q_1 in this example), does not have any role.
- The first repeated-state in this example is q_2.
- Now, let's see this relationship in the original transition graph.

What is Pigeon and What is Pigeonhole?



- So, we can consider the edges or the states right after them as the pigeonholes.
- In this lecture, we'll switch between these two based on the context.

Pigeonhole Principle and DFA's

Conclusion

If a DFA has m states, and

we process a string w whose size is $|w| \geq m$,

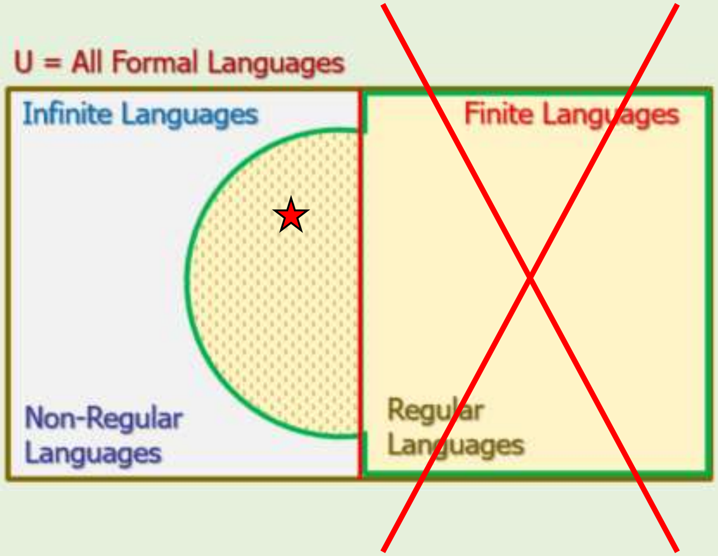
then by the pigeonhole principle,

at least one state should be visited more than once.

A Property of Infinite Regular Languages

A Property of Infinite Regular Languages

- Consider L as an
  INFINITE REGULAR language.
- Since L is regular, so,
  there exists a DFA that accepts it.
- Let's assume this DFA has m states
  (that should be a finite number).



- Take the general string $w = a_1\ a_2\ \ldots\ a_k \in L$ whose size is $|w| \geq m$.

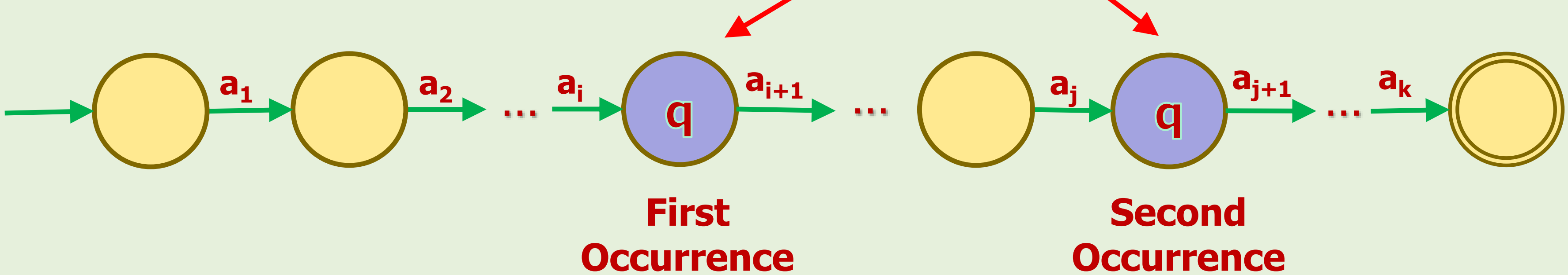
- Since $|w| \geq m$, therefore, based on pigeonhole principle,
  in the walk of w,

  at least one state is visited more than once.

A Property of Infinite Regular Languages

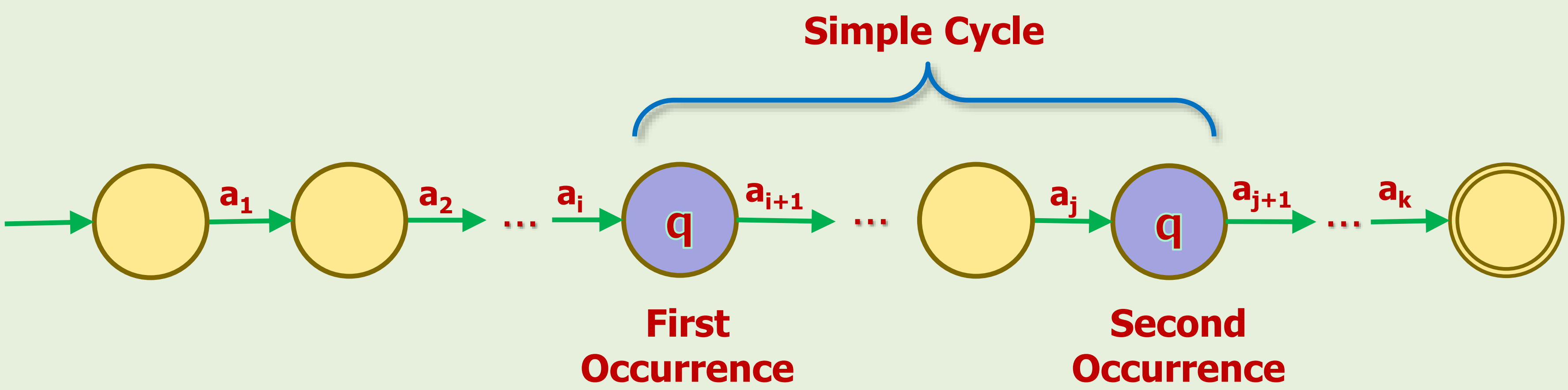
- The following graph is the one-dimensional projection of w.
- Why is the last state accepting-state?
  - Because $w \in L$, therefore the last state must be accepting-state.
- Can there be any other accepting-state?
  - Of course! Any state can be accepting-state.
- Let's assume the first repeated-state is "q".

$w = a_1\ a_2 \dots a_k$



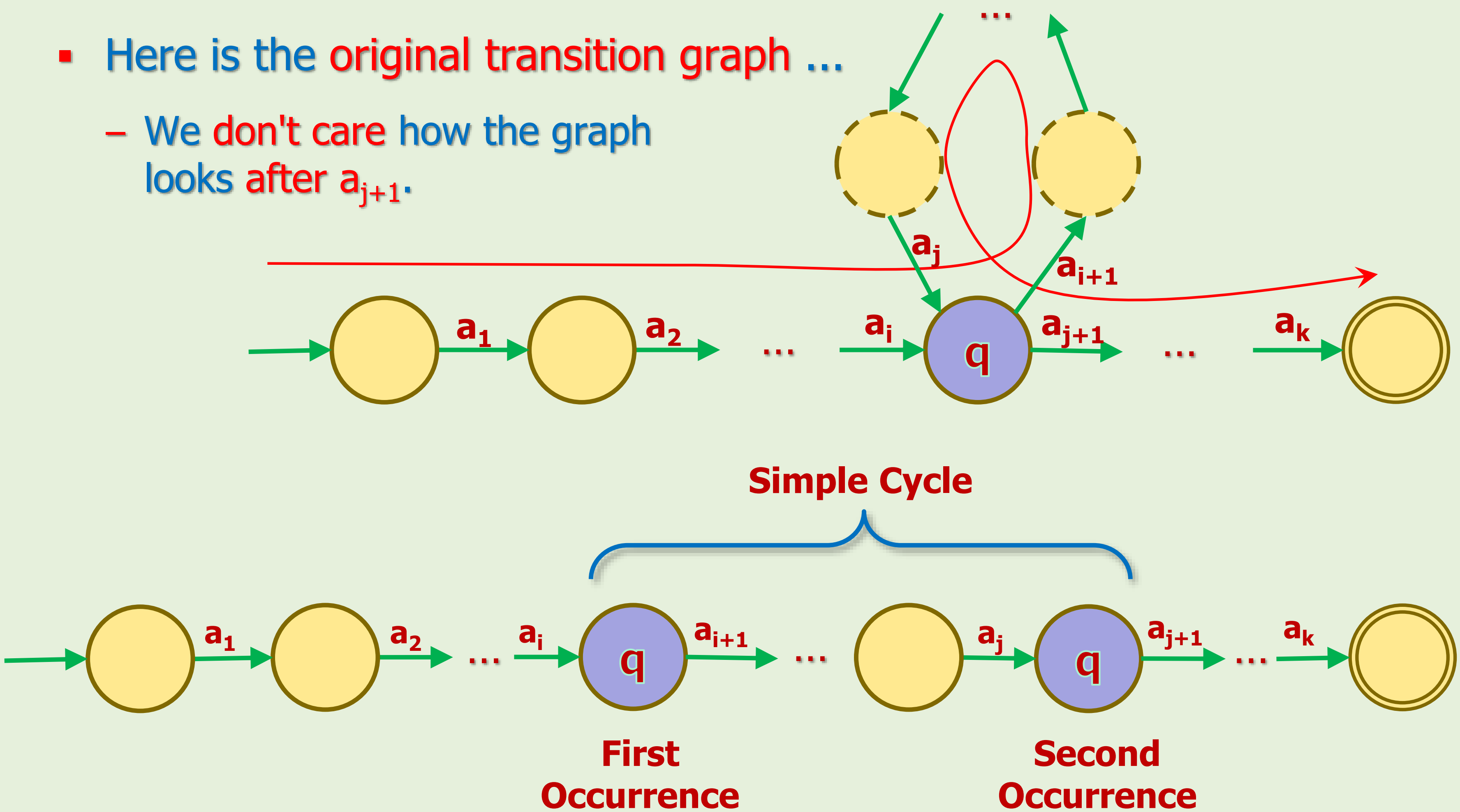
A Property of Infinite Regular Languages

- Note that between two q's, there is no nested repeated-states.
  - We can always pick the first repeated state in which there is no nested repeated-states.
- Therefore, if we show the original transition graph, this portion must be a "simple cycle".



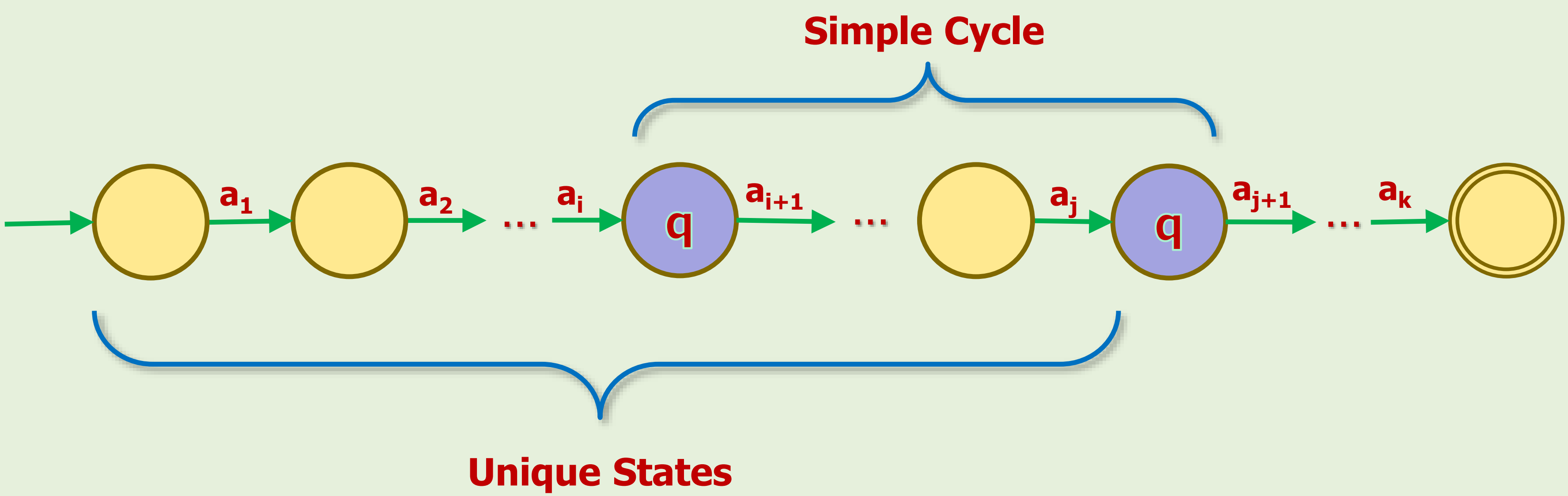
A Property of Infinite Regular Languages

- Here is the original transition graph …
  - We don't care how the graph looks after a_{j+1}.



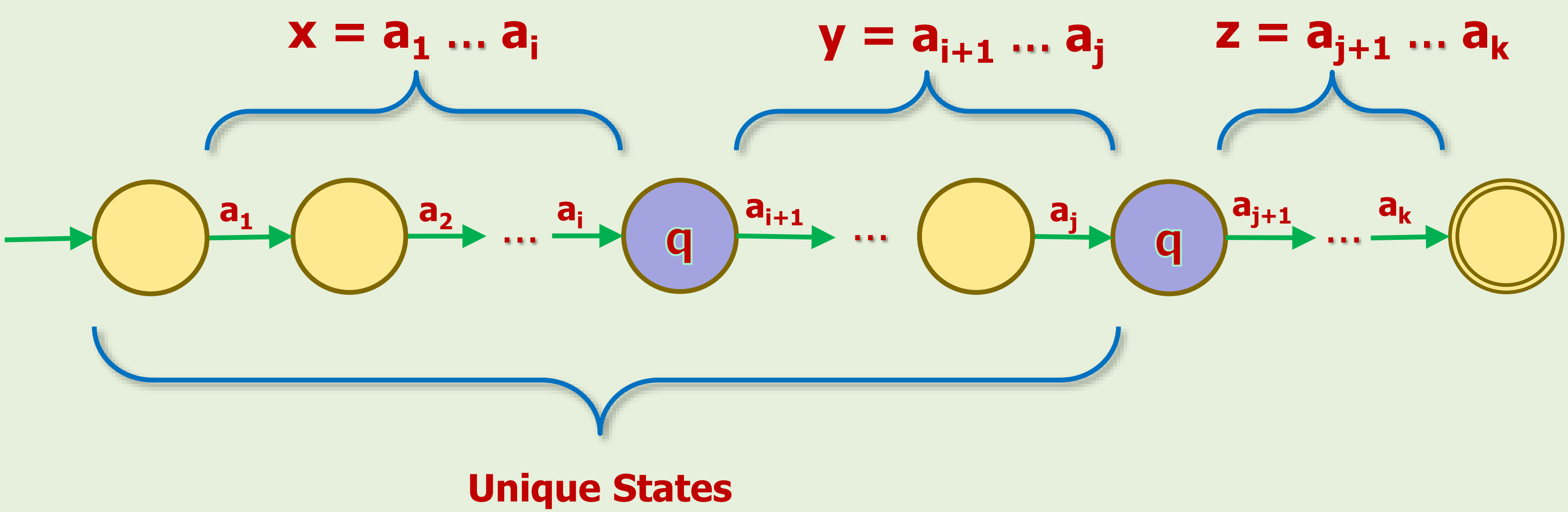
A Property of Infinite Regular Languages

- Let's review the facts we have so far:
  1. From a_1 to a_i, we have unique states (visited once). because we assumed q is the first repeated-state.
  2. From a_{i+1} to a_j, we have unique states because it is a simple cycle. (Only q (the base) is repeated!)
- Therefore, we have unique states from a_1 to a_j.



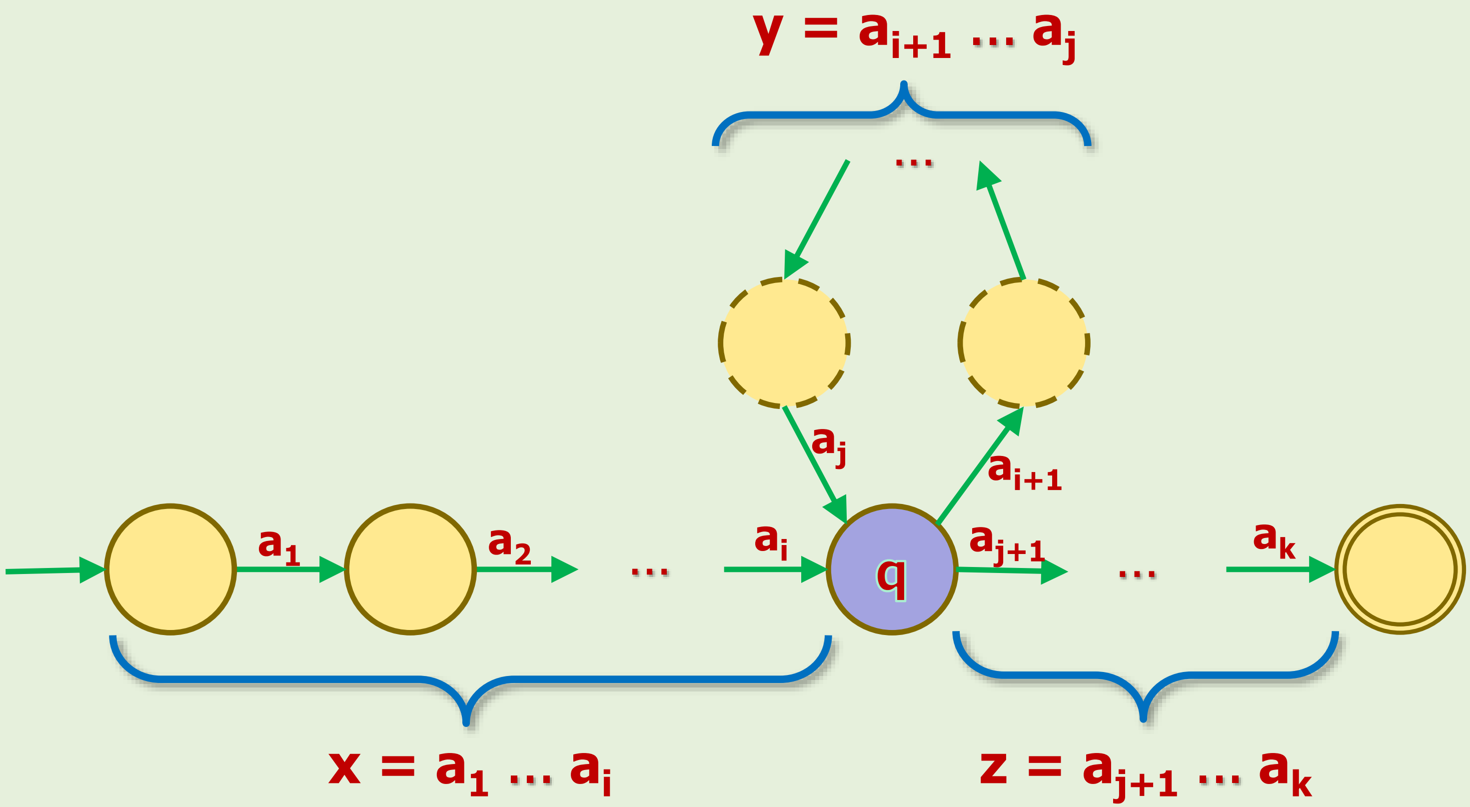
A Property of Infinite Regular Languages

- Now, let's name different portions of the string:
- We can split string w as xyz.
  (x, y, and z are variables for substrings.)
- Note that y corresponds to substring between two q's.



A Property of Infinite Regular Languages

- Now let's see how x, y, and z looks in the original transition graph.



Important Questions

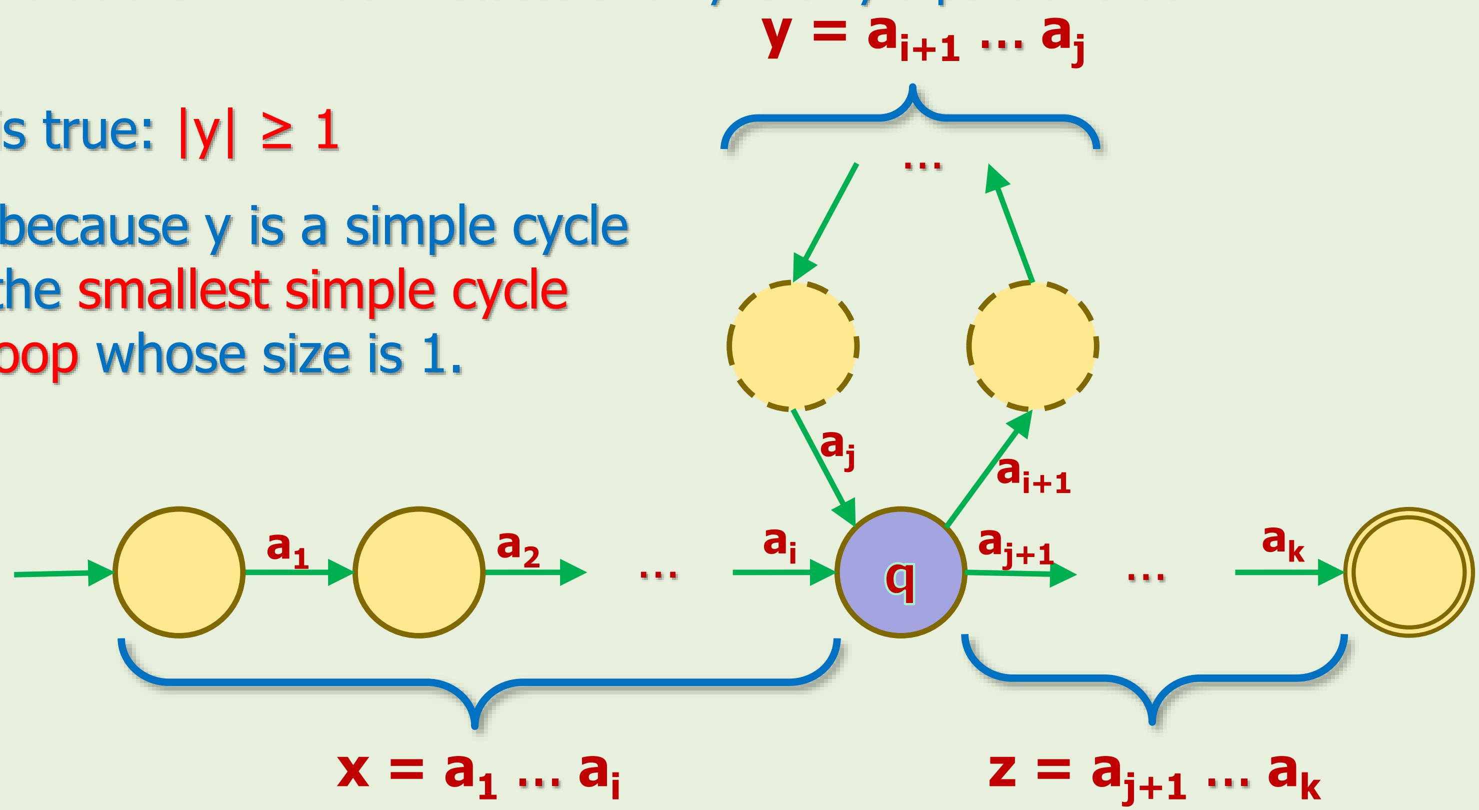
1. Is this true: $|xy| \le m$

   Yes, because we learned a_1 to a_j (= xy) are unique states and there is no repeated-states between them.

   Recall that the DFA has m states and xy is only a part of that.

2. Is this true: $|y| \ge 1$

   Yes, because y is a simple cycle and the smallest simple cycle is a loop whose size is 1.



More Questions!

We've already known $w = xyz \in L$.

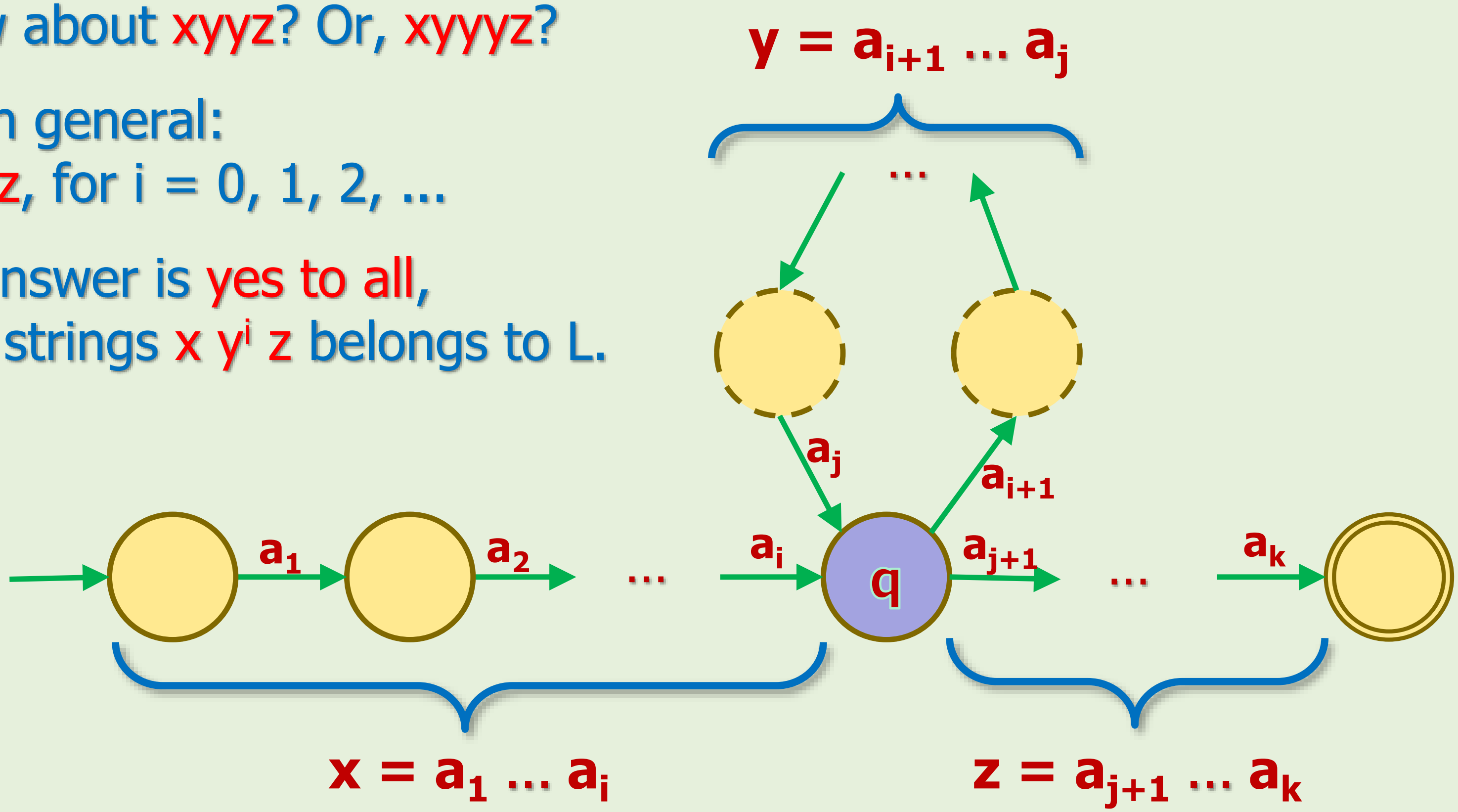
3. Is string $xz = a_1\ a_2 \ldots a_i\ a_{j+1} \ldots a_k$ accepted by this DFA?

   Yes, it skips y part. So $xz \in L$

4. How about $xyyz$? Or, $xyyyz$?

5. Or in general:
   $x\,y^i\,z$, for $i = 0, 1, 2, \ldots$

- The answer is yes to all,
  so all strings $x\,y^i\,z$ belongs to L.



A Property of Infinite Regular Languages

Conclusion

- If L is an infinite regular language,

  and if we pick a string $w = xyz \in L$,

  and split it in a certain way (means it satisfies some conditions),
- ... then, we can pump any number of y's

  and the resulting strings would belong to the language L.
- Means: $w_i = xy^iz \in L$ for $i = 0, 1, 2, \ldots$

- And this was the **mysterious concept** of "Pumping Lemma"!

Pumping Lemma

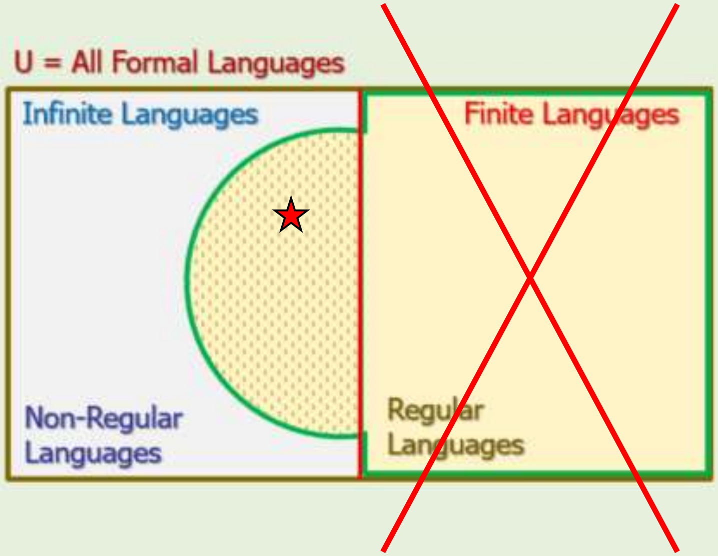
What is a Lemma?

Etymology

- "Lemma" is a smaller theorem to help proving a bigger one.

- Very occasionally lemmas can take on a life of their own.
- In computer science, "pumping lemma" is one of them.

Pumping Lemma



If L is an INFINITE REGULAR language,

Then there exists an $m \geq 1$ such that

If $w \in L$ and $|w| \geq m$

Then //pumping lemma guarantees that ...

We must be able to divide w into three parts xyz in such a way that all of the following conditions are satisfied:

$|xy| \leq m$, and

$|y| \geq 1$, and

$w_i = x\,y^i\,z \in L$ for $i = 0, 1, 2, 3, \ldots$.

Formal Statement of Pumping Lemma

If L is an infinite regular language,

Then

(1) There exists an $m \geq 1$ such that

If (2) $w \in L$ and (3) $|w| \geq m$

Then //P. L. guarantees that ...

(4) We must be able to divide w into xyz in such a way that all of the following conditions are satisfied:

(5) $|xy| \leq m$, and

(6) $|y| \geq 1$, and

(7) $w_i = x\, y^i\, z \in L$

(8) w_i for $i = 0, 1, 2, \ldots$

Steps of Pumping Lemma

Step	Description	Comment
1	Take an m	Always take it as m
2	Take w	A string from the language dependent to m
3	Check $\|w\| \geq m$	
4	Find x, y, z	$w = x\ y\ z$
5	Check $\|xy\| \leq m$	
6	Check $\|y\| \geq 1$	
7	Construct $w_i = xy^iz \in L$	
8	Check various i's of w_i	For i = 0, 1, 2, 3, ...

Pumping Lemma

Example 7

- Verify the pumping lemma property on the following infinite regular language.

$$L = \{a^n b : n \geq 0\}$$

Solution

(1) Let's take the m = 2. Why not 3?

OK, **let's take it as m**.

– If we need, we'd make some boundary on m later.

(2) Let's take $w = a^m b$

– Note that m is a constant.

– It means, $a^m b$ is a string, NOT a pattern.

(3) Check w's size:
$|w| = |a^m b| = m+1 \geq m$ ✓

- Pumping lemma guarantees that:

(4) There exists x, y, z such that:

$$w = a^m b = xyz = \lambda \quad a \quad a^{m-1}b$$

(5) $|xy| = |a| = 1 \leq m$ ✓

(6) $|y| = 1 \geq 1$ ✓

(7) $w_i = xy^iz = \lambda\, a^i\, a^{m-1}b \in L$

(8) Check various i's of w_i:

- i=0 $w_0 = xz = a^{m-1}b \in L$ ✓
- i=1 $w_1 = xy^1z = a^m b \in L$ ✓
- i=2 $w_2 = xy^2z = a^{m+1}b \in L$ ✓
- i=3 $w_3 = xy^3z = a^{m+2}b \in L$ ✓
- ...

Pumping Lemma

Example 8

- Verify the pumping lemma property on the following infinite regular language.

$$L = \{bba^n : n \geq 0\}$$

Solution

References

1. Linz, Peter, "An Introduction to Formal Languages and Automata, 5th ed.," Jones & Bartlett Learning, LLC, Canada, 2012
2. Kenneth H. Rosen, "Discrete Mathematics and Its Applications, 7th ed.," McGraw Hill, New York, United States, 2012
3. Costas Busch's website: http://csc.lsu.edu/~busch/
4. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790