San José State University
Department of Computer Science

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

Regular Languages

Lecture 12

Day 12/31

CS 154

Formal Languages and Computability

Spring 2019

Agenda of Day 12

- Collecting Quiz +
- Summary of Lecture 11
- A Few Slides From the Past
- Lecture 12: Teaching ...
  - Regular Languages

A Few Slides From the Past

Summary of Lecture 11: We learned ...

NFAs' Formal Definition

- An NFA M is defined by a quintuple:

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Except δ, the rest items are similar to DFAs'.

$$\delta : Q \times \Sigma \rightarrow 2^Q$$

δ is total function.

Machines and Languages Association

- Every machine has an associated language.
- BUT we do NOT know yet whether or not for every language, we can construct a machine!

DFAs vs NFAs

- What is power?
- Automata class A is more powerful than class B iff ...
  - ... the set of languages recognized by class B is a proper subset of the set of the languages recognized by class A.

Theorem

- The set of languages recognized by NFAs are equal to the set of languages recognized by DFAs.
- NFAs and DFAs have the same power.

Any question?

Introduction

Example 1

- Design a DFA/NFA to recognize our famous language:
- $L = \{a^n b^n : n \geq 0\}$ over $\Sigma = \{a, b\}$
- Struggling?!

You!

Me!

- After some struggling, we realized that we could not construct such machines.
- But why?

Why We Could NOT Construct Such Machines

Reason

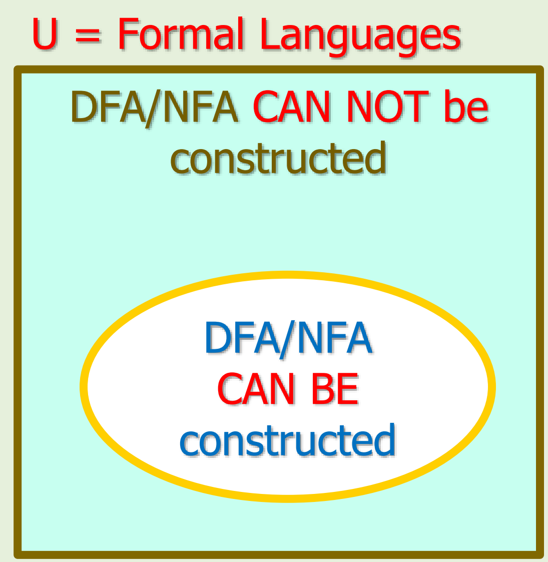
- Because number of a's and b's must be equal.
- How can we count the number of a's?
- To count it, we'd need a counter and a storage!

- But DFAs/NFAs don't have storage.
- And we cannot implement a counter by them!

Categorizing Formal Languages

- We just realized that there are different kinds of formal languages.
  - Some languages are more complex than the others.
- To study formal languages, we need to categorize them.

- Up to this point, we've realized that ...
  - We can construct a DFA/NFA for some languages while we CANNOT for the others.



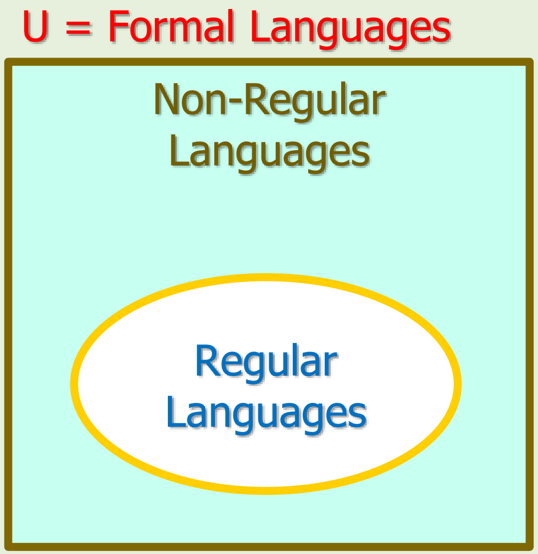
- Let's give a name to these two categories!

Regular Languages

Definition

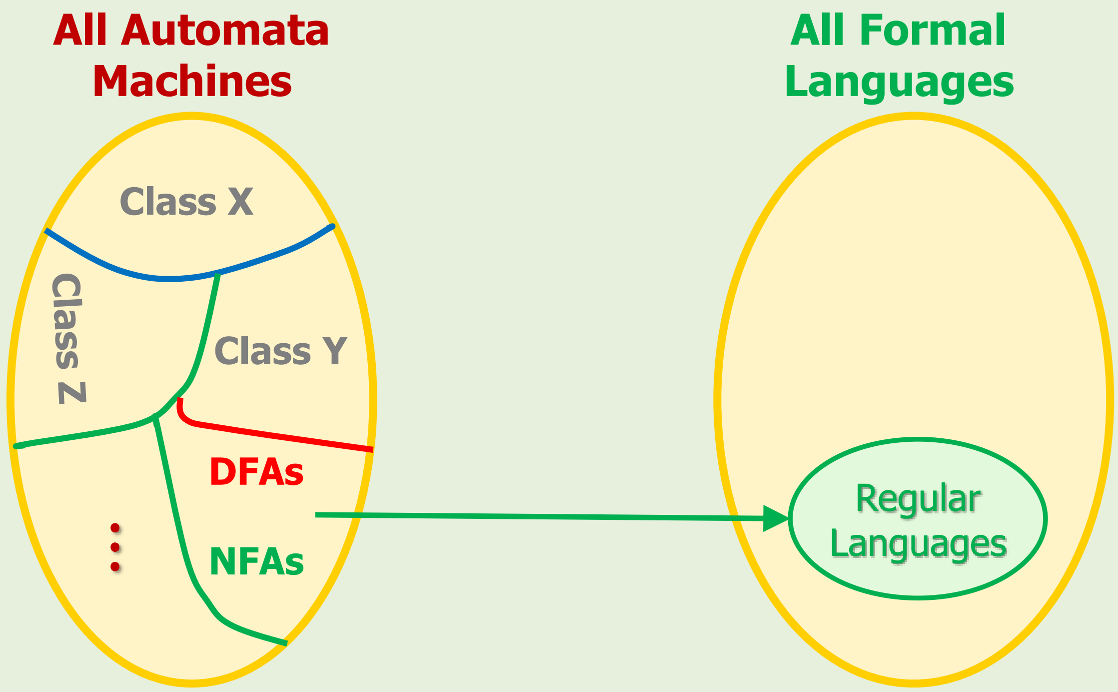
- A language is called regular if
  there exists a DFA/NFA to recognize it.

- Therefore, the rest of languages are called non-regular.



Machines and Languages Association

- We already saw the association between machines and languages.
- Now we have a name for the languages that DFAs/NFAs recognize.



Categorizing Formal Languages

- Recall that before, we categorized formal languages as "finite" and "infinite".
- And this is our second categorization:
  1. "Regular Languages", and
  2. "Non-Regular Languages"
- Note that the correct English word is "irregular" but in computer science we use "non-regular".
- How can we prove that a language is regular?
- We need to construct a DFA/NFA for it.

Regular Languages

Example 2

- Which of the following languages is regular over $\Sigma = \{a, b\}$?
- $L = \{ \}$
- $L = \{\lambda\}$
- $L = \{abbaa\}$
- $L = \{\lambda, a, abb\}$
- $L = \{a, b\}^*$
- $L = \{a^n b : n \geq 0\}$

- We've already constructed DFAs for (almost) all of the above languages.
- So, all of them are regular.

Homework

1. Prove that the language $L = \{awa : w \in \{a, b\}^*\}$ is regular.
2. Write a set-builder for L^2.
3. Prove that L^2 is regular.

Heuristically Recognizing Regular Languages

- Sometimes we can heuristically find out whether a language is regular or not. How?
- Let's explain it through some examples.

Example 3

- Which of the following languages are regular?

1. $L = \{ab\ w : w \in \{a, b\}^*\}$
2. $L = \{w\ w^R : w \in \{a, b\}^*\}$
3. $L = \{w\ w : w \in \{a, b\}^*\}$
4. $L = \{w\ abb\ w : w \in \{a, b\}^*\}$
5. $L = \{1^{2k} : k \geq 0\}$ over $\Sigma = \{1\}$
6. $L = \{1^n + 1^m = 1^{n+m} : n, m \geq 1\}$ over $\Sigma = \{1, +, =\}$ (Unary addition)

Finite Languages

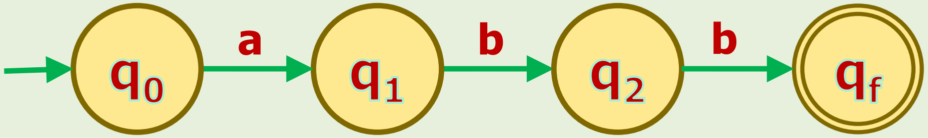
Finite Languages

Theorem

- All finite languages are regular.

Proof

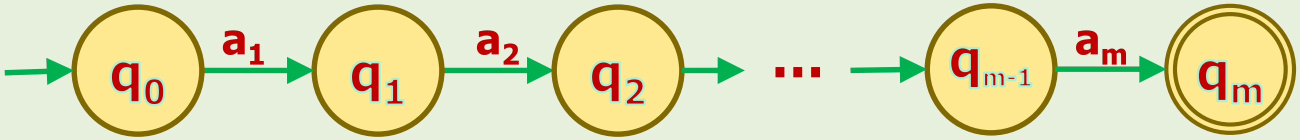
- To prove this theorem, we need to construct an NFA for a general finite language $L = \{s_1, s_2, \ldots, s_n\}$
  - Where $s_j \in \Sigma^*$ for $j = 1, 2, \ldots, n$ and $n \in \mathbb{N}$
- We know that strings are finite sequence of symbols.
- So, we can construct an NFA for every string.
- For example if s_2 = abb, then the following NFA can recognize it.



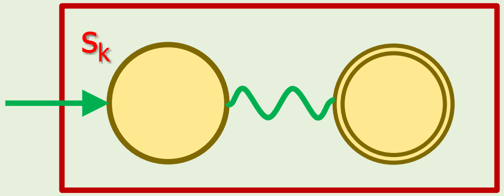
Finite Languages Are Regular

Proof (cont'd)

- Let $s_k = a_1\ a_2\ \ldots\ a_m$ be a general string where $a_i \in \Sigma$ for $i = 1, 2, \ldots, m$ and $m \in \mathbb{N}$
- We can construct the following NFA to recognize s_k.



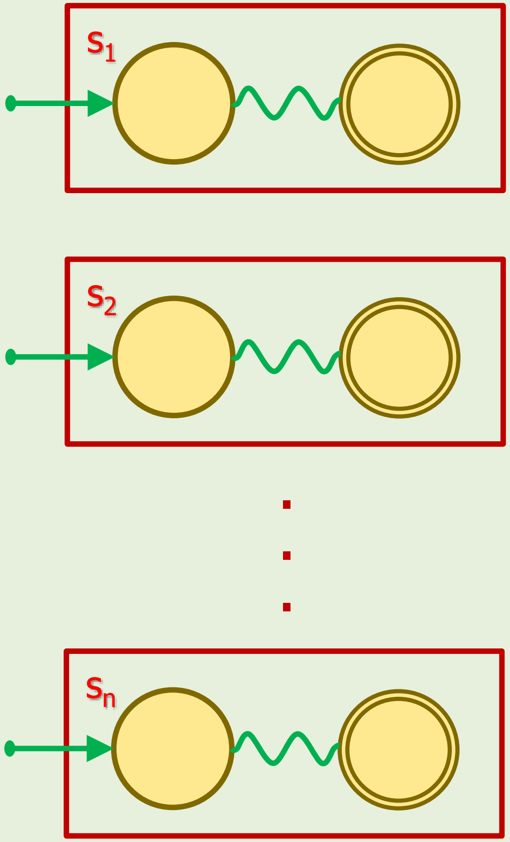
- For simplicity, let's show this NFA as:



Finite Languages Are Regular

Proof (cont'd)

- In the similar way, we can construct an NFA for every s_j in the language.

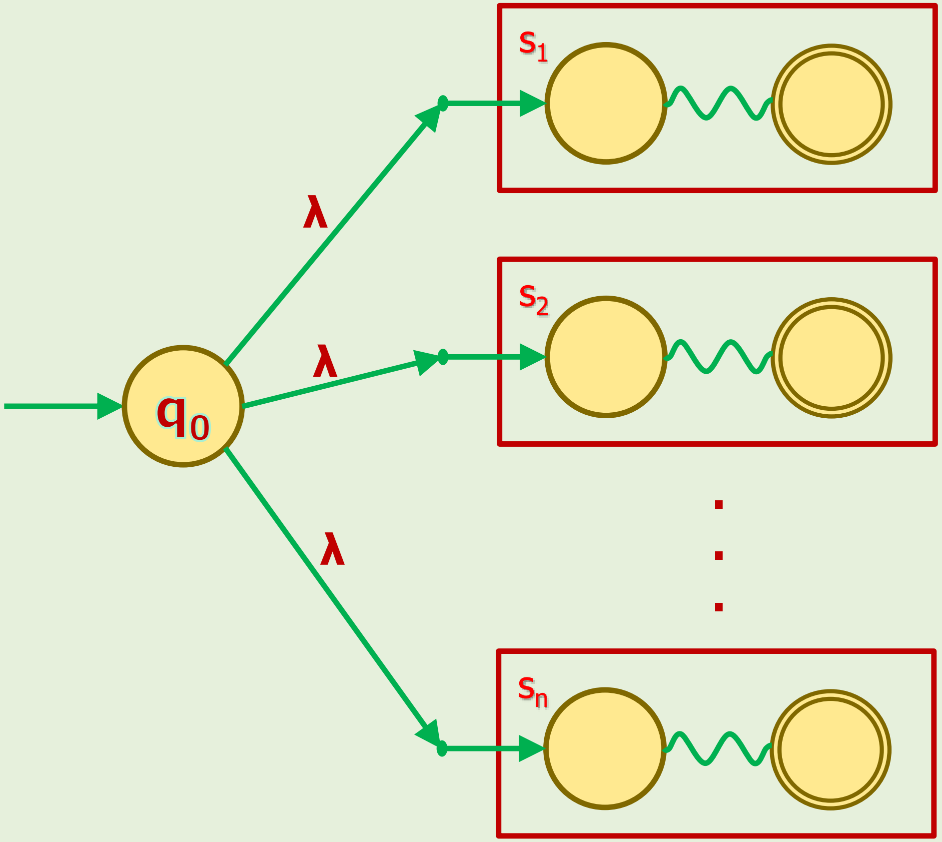


- Now, we need to combine these simple NFAs and construct an NFA that recognizes L.

Finite Languages Are Regular

Proof (cont'd)

- We combine them by using λ-transitions.
- This new NFA recognizes L.
  - $L = \{s_1, s_2, ..., s_n\}$
- Explain why?



- Since L is a general finite language, so, we proved all finite languages are regular.

Non-Regular Languages Are Infinite

- The contrapositive of every theorem is also true.

Recap: Contrapositive

$p \rightarrow q \equiv {\sim}q \rightarrow {\sim}p$

- The theorem we just proved:

If L is finite, then L is regular.

- L is finite. ≡ LF
- L is regular. ≡ LR

$$\mathbf{LF \rightarrow LR \equiv {\sim}LR \rightarrow {\sim}LF}$$

- Translation:

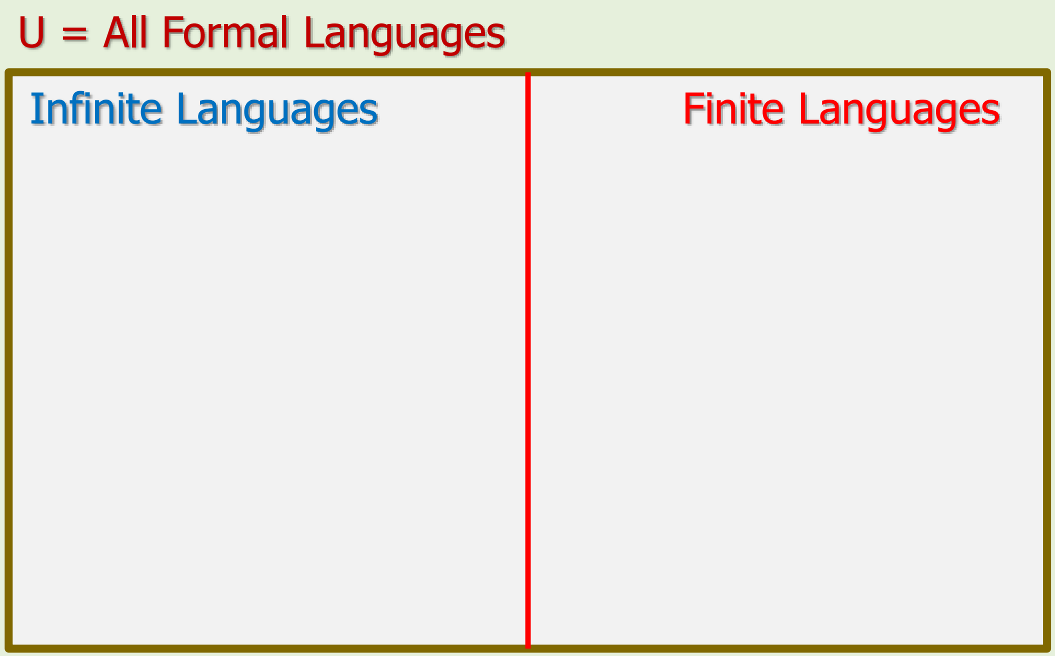
If L is non-regular (= not regular), then L is infinite (= not finite).

- The compact version:

All non-regular languages are infinite.

Formal Languages Categorization

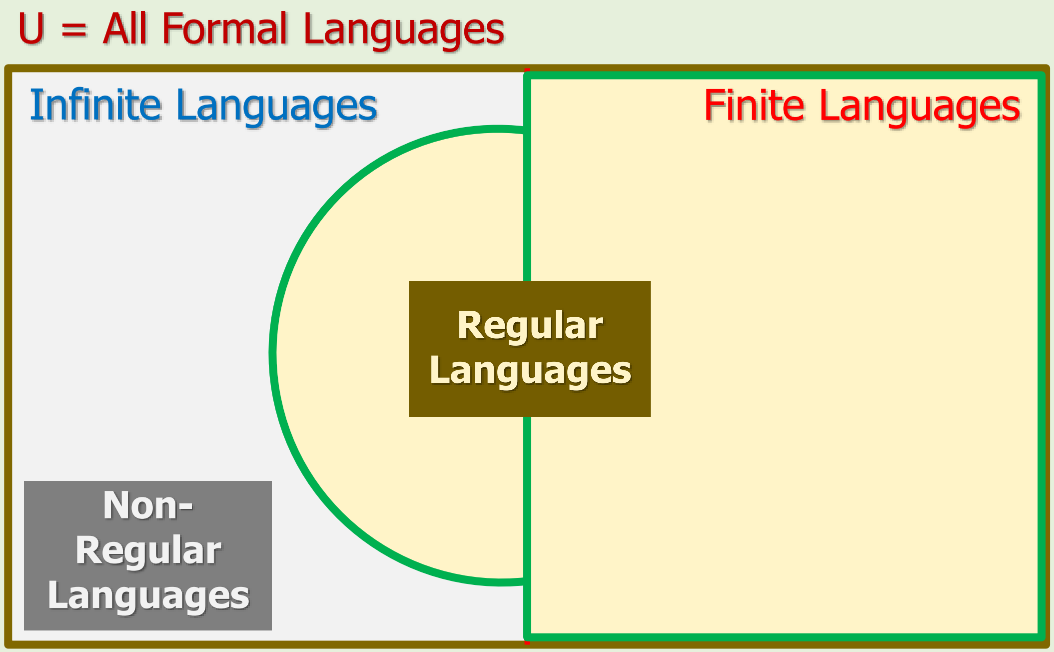
1st Categorization: Finite and Infinite



- Where would you locate "regular" and "non-regular" languages?

Formal Languages Categorization

2nd Categorization: Regular and Non-Regular



Closure Property of Regular Languages

Theorem

- If L, L_1 and L_2 are all regular languages, then:

Union	$L_1 \cup L_2$
Concatenation	$L_1 L_2$
Star-Closure	L^*
Reversal	L^R
Complement	$\overline{L}$
Intersection	$L_1 \cap L_2$
Minus	$L_1 - L_2$

are regular languages too.

- It means: The family of regular languages is closed under the above operations.

Closure Property of Regular Languages: Notes

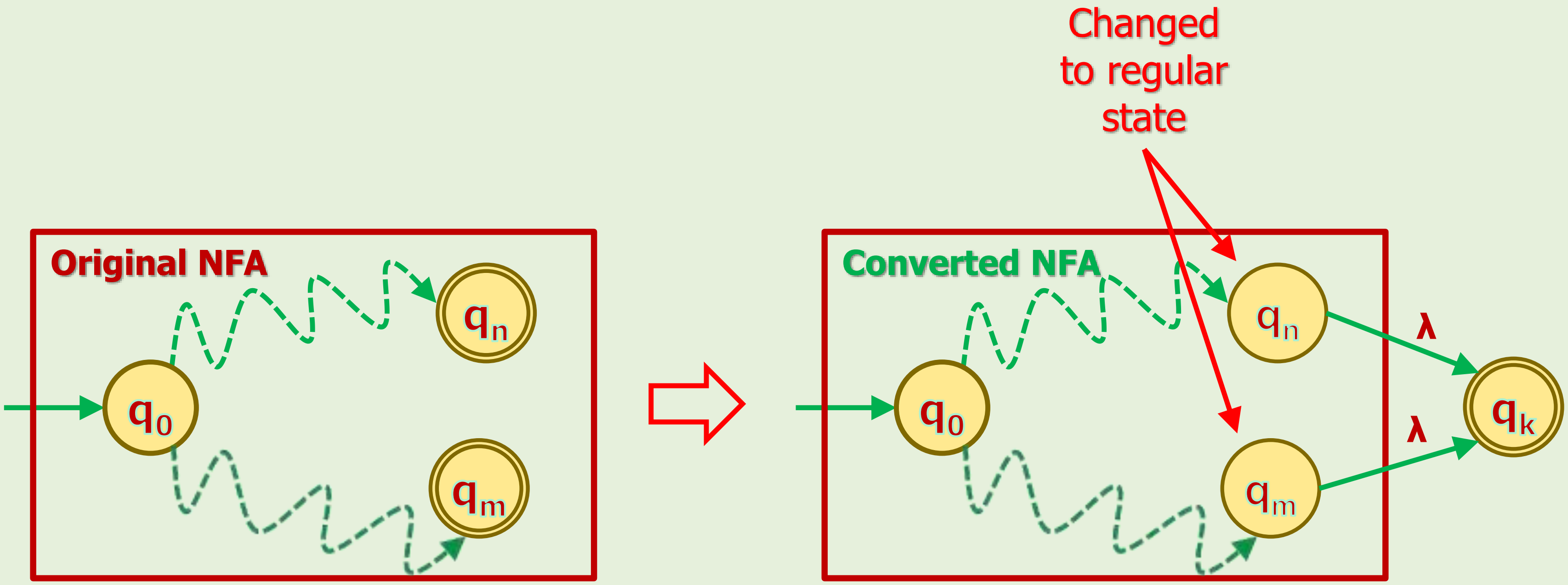
1. To prove this theorem, we'd use NFA's because they are simpler and more flexible.
2. Since, there is always an equivalent DFA for any NFA, we won't lose the generality of the theorem.

- Before proving the theorem, we need to be familiar with an useful transformation.

A Useful Transformation

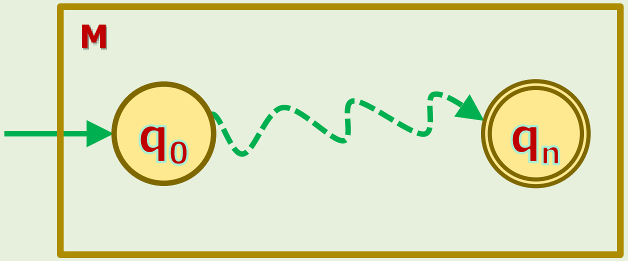
Converting an NFA to Single-Accept-State

- Consider the following NFA (left side) that has two or more accept state.
- We can easily convert it to a single-accept-state by using the following technique.



Representing an NFA in General Form

- So, the resulting NFA has a single accept-state.



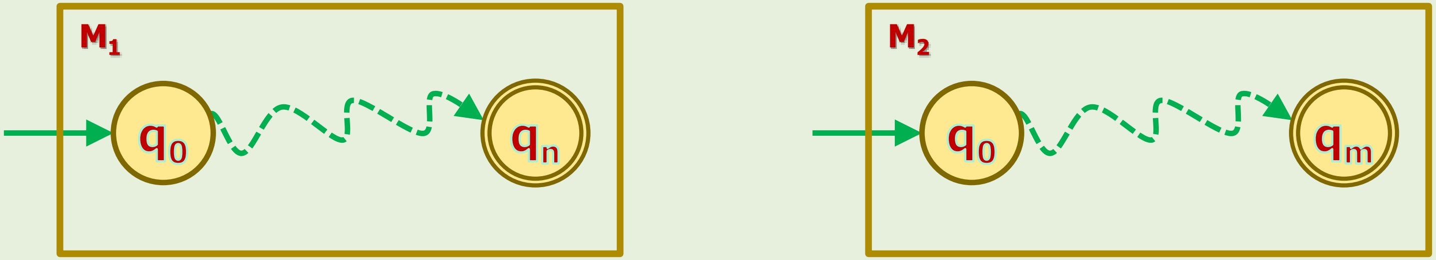
- Using this technique, we can convert any NFA with two or more accept-states to a single-accept-state.

Union

- If L_1 and L_2 are regular languages, then prove that $L_1 \cup L_2$ is regular language.

Proof

- L_1 and L_2 are regular languages, so there are an NFA for each of them (M_1 and M_2).

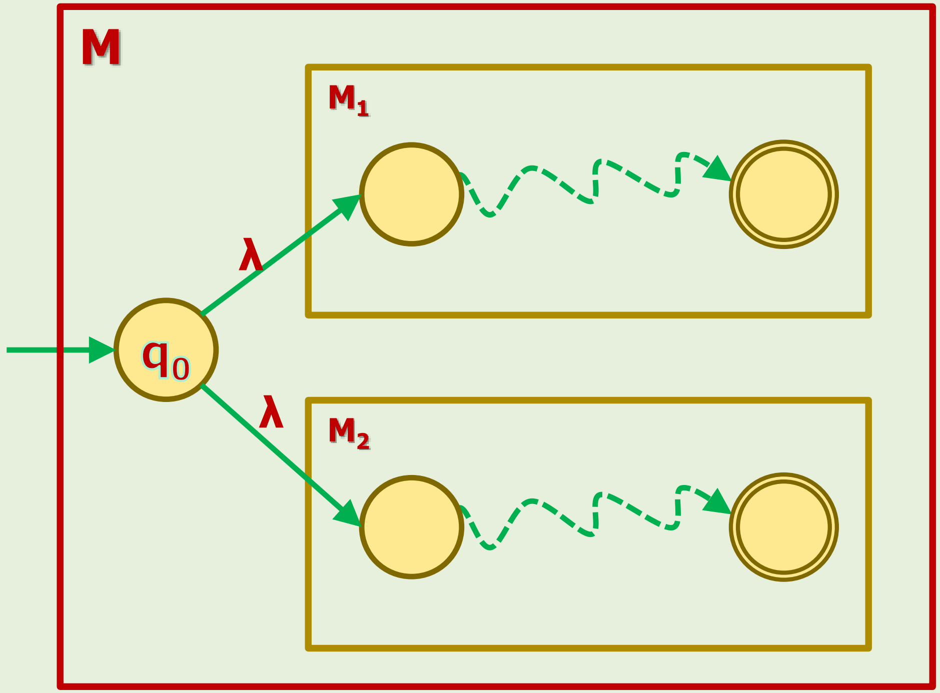


- To prove $L_1 \cup L_2$ is regular, . . .
- . . . we need to construct an NFA for $L_1 \cup L_2$.

Union

Proof (cont'd)

- We construct the following NFA for $L_1 \cup L_2$.

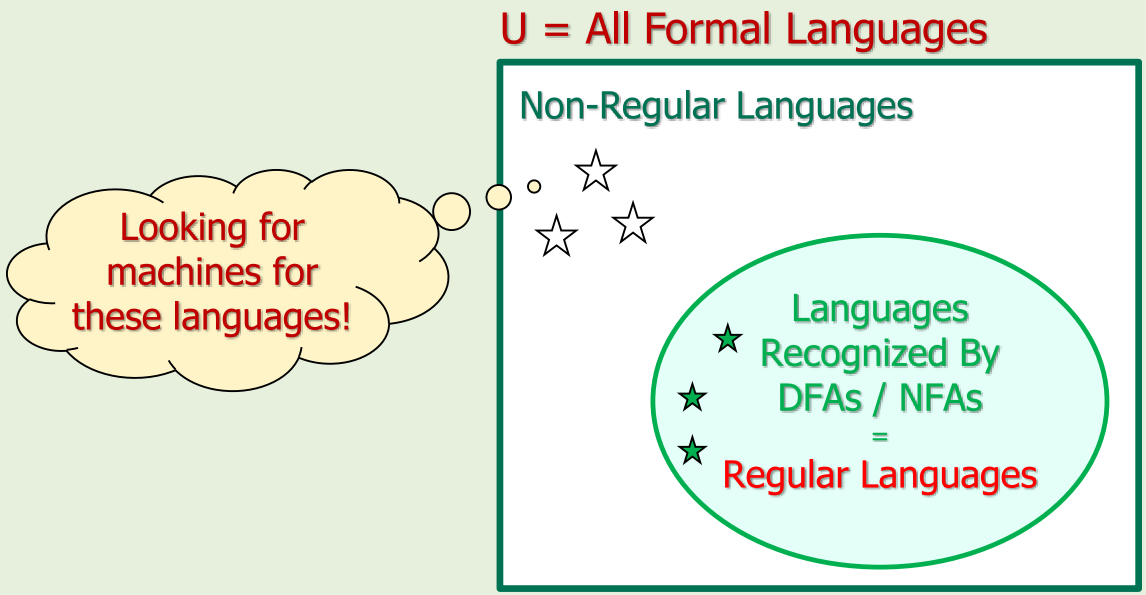


$$w \in L_1 \cup L_2 \Rightarrow w \in L_1 \text{ OR } w \in L_2$$

What is the Next Step?

Conclusion

- NFAs and DFAs recognize "regular languages".
- The next step is to define a new class of machines that recognizes "non-regular languages".



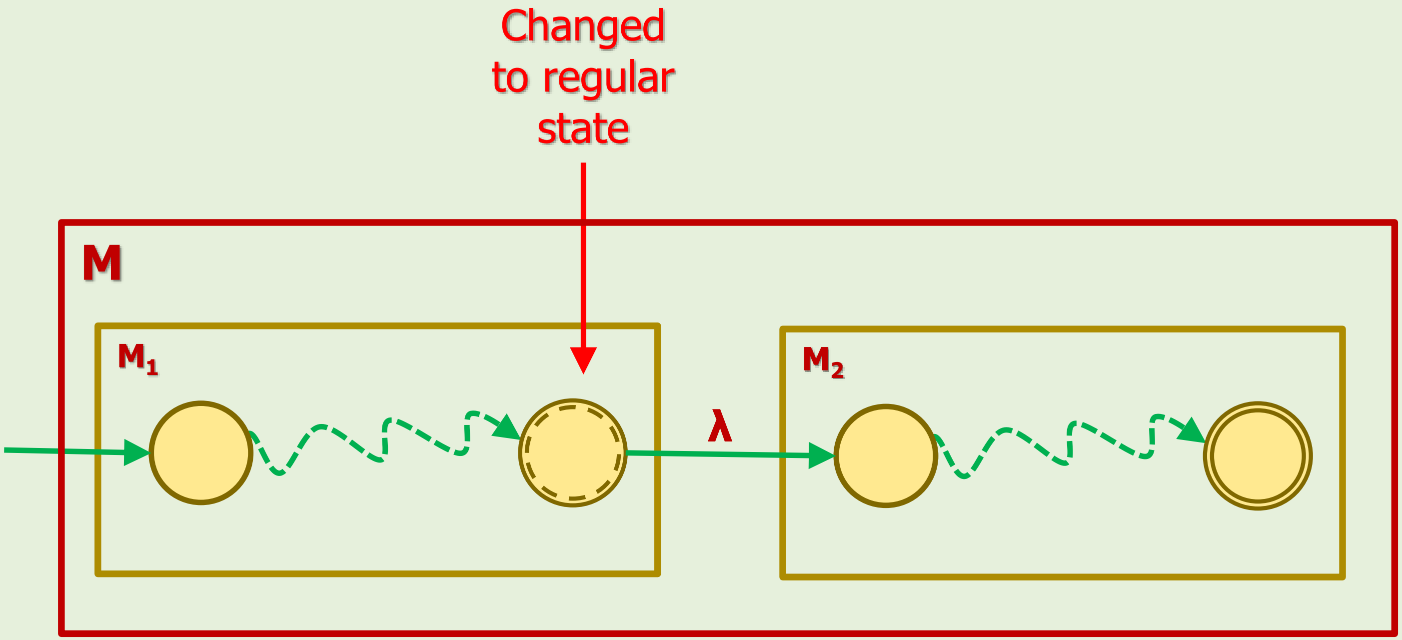
Reading Assignment

Concatenation

- If L_1 and L_2 are regular languages, then prove that $L_1 L_2$ is regular language.

Proof

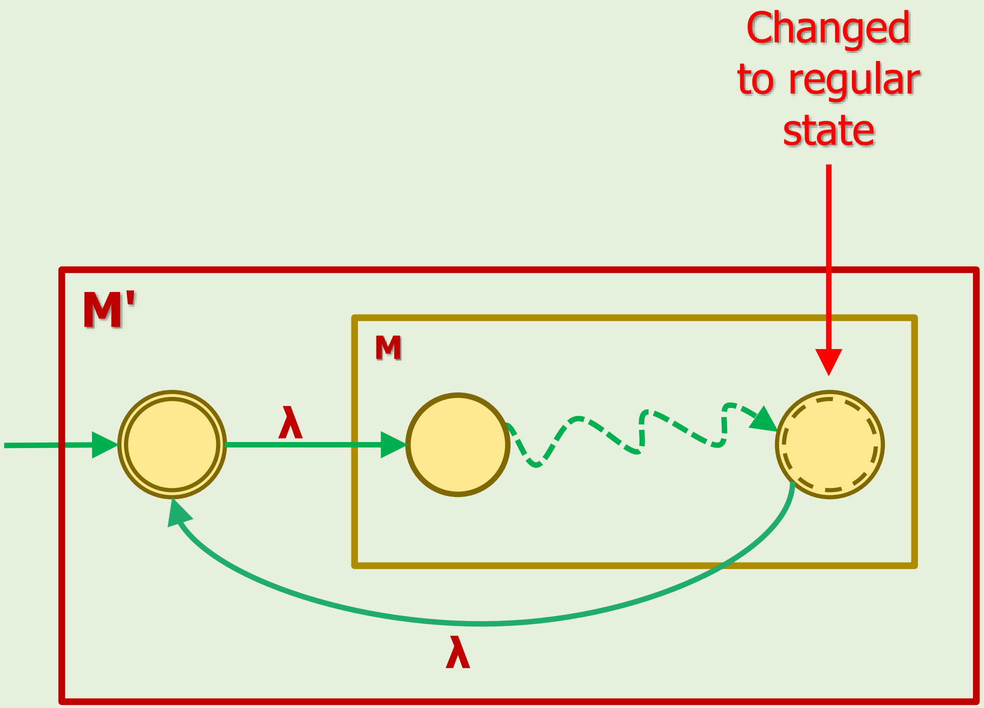
- To prove $L_1 L_2$ is regular, we need to construct an NFA for it.



Star Closure

- If L is regular language, then prove that L^* is regular language.

Proof

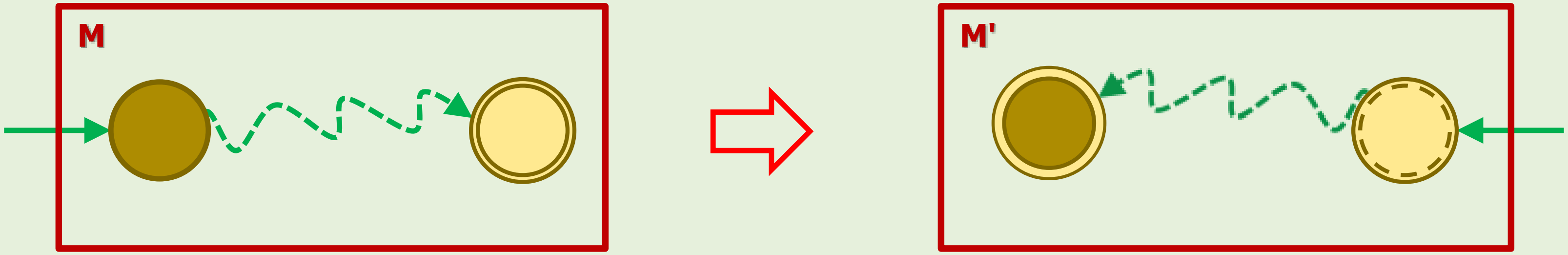


Reversal

- If L is regular language, then prove that L^R is regular language.

Proof

- Make the following transformations in the original machine M:
  1. Reverse all transitions
  2. Change the initial state to accept state.
  3. Change the accept state to initial state.
- The resulting machine will accept L^R.

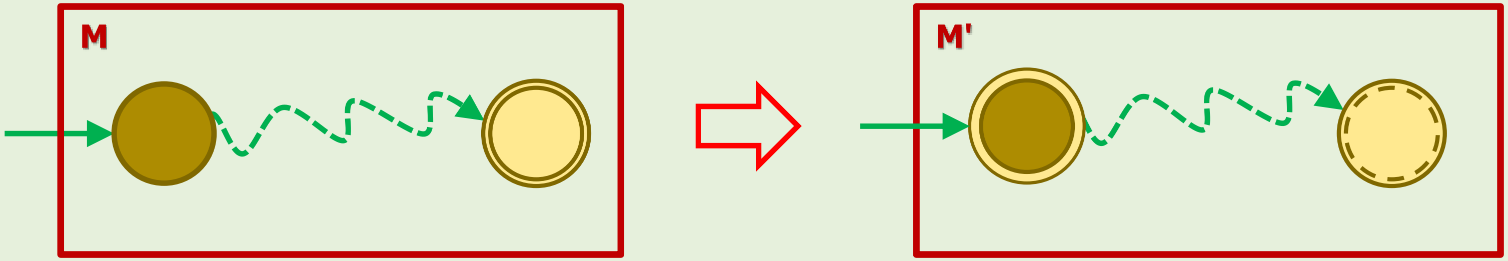


Complement

- If L is regular language, then prove that $\overline{L}$ is regular language.

Proof

- Make the following transformations in the original **DFA** M:
  1. Change the accept states to regular states.
  2. Change the regular states to accept states.
- The resulting machine will accept $\overline{L}$.

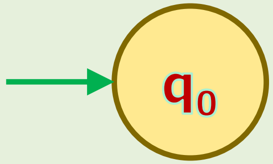


Complement

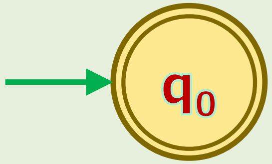
- Note that the transformations for "complement" does not work for NFA's.
- The following example shows why?

Example 4

- Construct an NFA for $L = \{\ \}$.



- Now, by using the rules in the previous slide, transform the NFA to accept $\overline{L} = \Sigma^* = \{a, b\}^*$



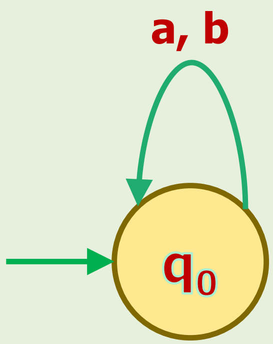
- But this new machine accepts just $\{\lambda\}$, not Σ^*.

Complement

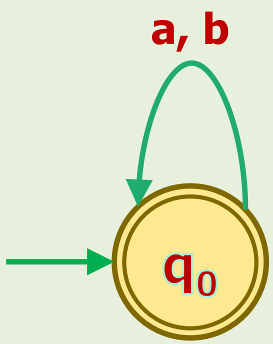
- The same example works fine when we use DFA's.

Example 5

- Construct a DFA for L = { }.



- Now transform the DFA to accept $\overline{L} = \Sigma^* = \{a, b\}^*$



Intersection

- If L_1 and L_2 are regular languages, then prove that $L_1 \cap L_2$ is regular language.

Proof

- We can use set theory identities to prove this theorem:

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ DeMorgan's law

- L_1 and L_2 are regular languages.
- $\overline{L_1}$ and $\overline{L_2}$ are regular languages.
- $\overline{L_1} \cup \overline{L_2}$ is regular language.
- $\overline{\overline{L_1} \cup \overline{L_2}}$ is regular language.
- Therefore, $L_1 \cap L_2$ is regular.

Minus

- If L_1 and L_2 are regular languages, then prove that $L_1 - L_2$ is regular language.

Proof

- We can use set theory identities to prove this theorem:
- $L_1 - L_2 = L_1 \cap \overline{L_2}$
- L_1 and L_2 are regular languages.
- $\overline{L_2}$ is regular language.
- $L_1 \cap \overline{L_2}$ is regular language.
- Therefore, $L_1 - L_2$ is regular

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. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790