👾

Topic 5 - Regular Languages

Week 9 - Regular Languages Pt1

regular operations — operate on regular languages

- preserve regularity of expressions

operation	denotation	use	characteristics
union	$\cup$, $L1 \cup L2 = \{x \mid x \in L1 \text{ or } x \in L2\}$	takes all elements in L2 and puts them in L1	- commutative: $A \cup B = B \cup A$ - associative: $(A \cup B) \cup C = A \cup (B \cup C)$ - idempotent: $A \cup A = A$ - union with $\emptyset$: $A \cup \emptyset = A$ - $(A \cup B) \circ C = (A \circ B) \cup (B \circ C)$ - $A \circ (B \cup C) = (A \circ B) \cup (A \circ C)$
concatenation	$\circ$, $L1 \circ L2 = \{xy \mid x \in L1.\text{ and } y \in l2\}$	a combination of the elements in L1 and L2, forming hybrid elements	- associative: $(A \circ B) \circ C = A \circ (B \circ C)$ - concatenation with ϵ: $A \circ \epsilon = \epsilon \circ A = A$ - concatenation with $\emptyset$: $A \circ \emptyset = \emptyset$ - $(A \cup B) \circ C = (A \circ C) \cup (B \circ C)$ - $A \circ (B \cup C) = (A \circ B) \cup (A \circ C)$
Kleene’s star	$*$, $L1^* = \{x1,x2,x3,\ldots xm \mid m \geq 0, \text{each } xi \in$	contains all concatenations of all	- $\emptyset^* = \{\epsilon\}$ - $\epsilon^* = \epsilon$

operation	denotation	use	characteristics
	L1}	elements in L1 only, including ϵ	- (A*)A* = A* - A* ∘ A* = A* - (A ∪ B)* = (A*B*)*

regular expression(RE) — used to express regular languages in shorter way, using regular operations

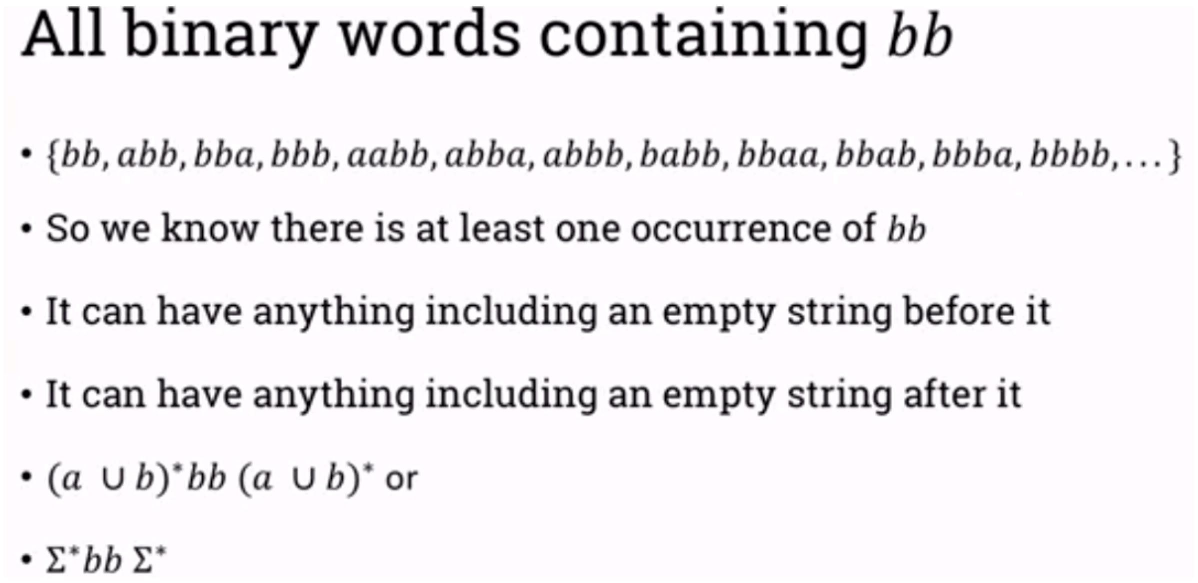
- value of a regular expression is a language; basically how you would read it
- formal definition of regular expression: say that R is a regular expression if R is

  1. a for some a in the alphabet Σ,
  2. ϵ,
  3. ∅,
  4. (R1 ∪ R2), where R1 and R2 are regular expressions,
  5. (R1 ∘ R2), where R1 and R2 are regular expressions, or
  6. (R1*), where R1 is a regular expression

  In items 1 and 2, the regular expressions a and ϵ represent the languages $\{a\}$ and $\{\epsilon\}$, respectively. in item 3, the regular expression ∅ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R1 and R2, or the start of the language R1, respectively.

  - don't mistake ϵ and ∅.
    - ϵ = the language containing a single string; the empty **string**
    - ∅ = the language that doesn't contain any string; the empty **language**
  - the formal definition is an inductive definition (defining regular expressions in terms of smaller regular expressions)
- atomic regular expressions:
  - empty language, ∅, is a regular expression, which is the empty regular language
  - and letter a in Σ is a regular expression and its languages is $\{a\}$

    - empty string, ϵ, is a regular expression representing the regular language $\{\epsilon\}$
- example:



All binary words containing bb

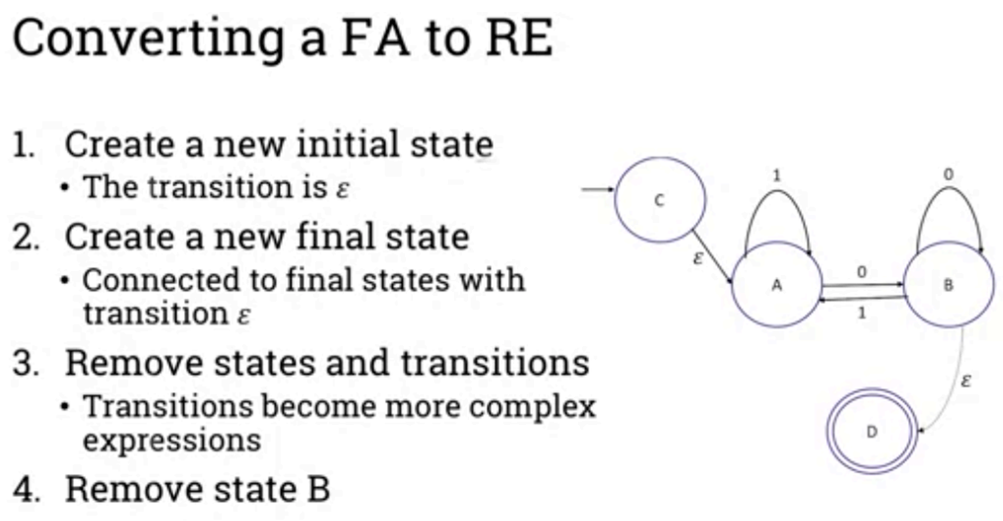
- $\{bb, abb, bba, bbb, aabb, abba, abbb, babb, bbaa, bbab, bbba, bbbb, \dots\}$
- So we know there is at least one occurrence of bb
- It can have anything including an empty string before it
- It can have anything including an empty string after it
- $(a \cup b)^* bb \, (a \cup b)^*$ or
- $\Sigma^* bb \, \Sigma^*$

regular language — a language described by a regular expression

- Kleene's theorem: a language is regular if and only if it can be described by a regular expression
    - is a two-way theorem
    1. if a language is described by a regular expression, then it is regular
    2. if a language is regular, it can be described by a regular expression
- correlation with FA(finite automata): proof
    - language of FA is regular, and for every regular language, there is a FA admitting this language
    - a language is regular if and only if it can be expressed by a regular expression
    - now there is a link between FA and RE
    - two main theorems:

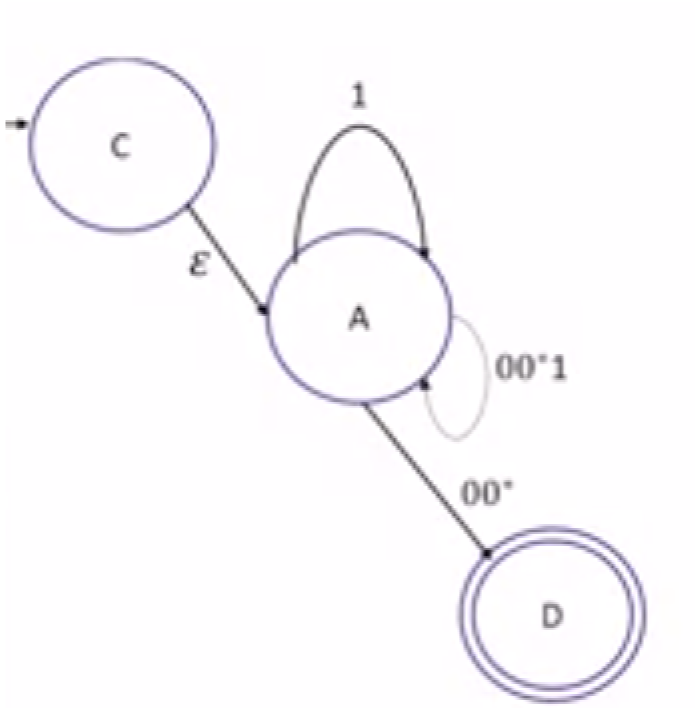
1. If L = L(A) for some FA, A, then there is a RE, R, such that L(R) = L

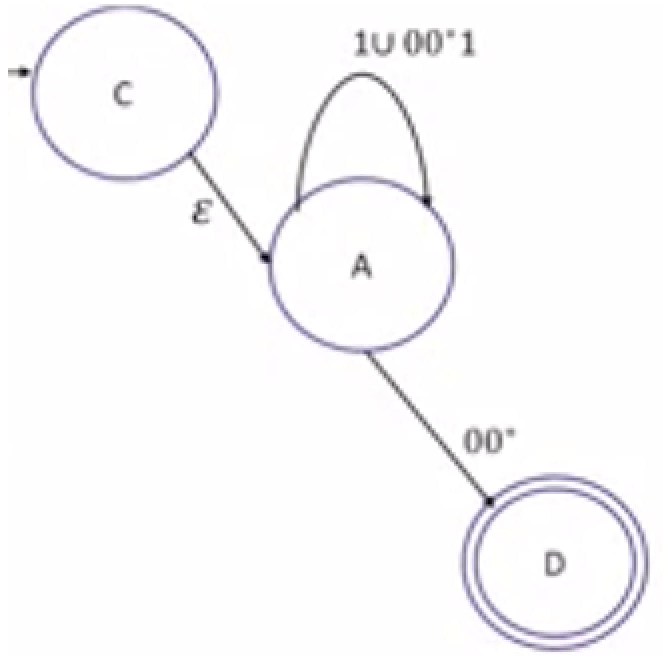
- converting FA to RE:

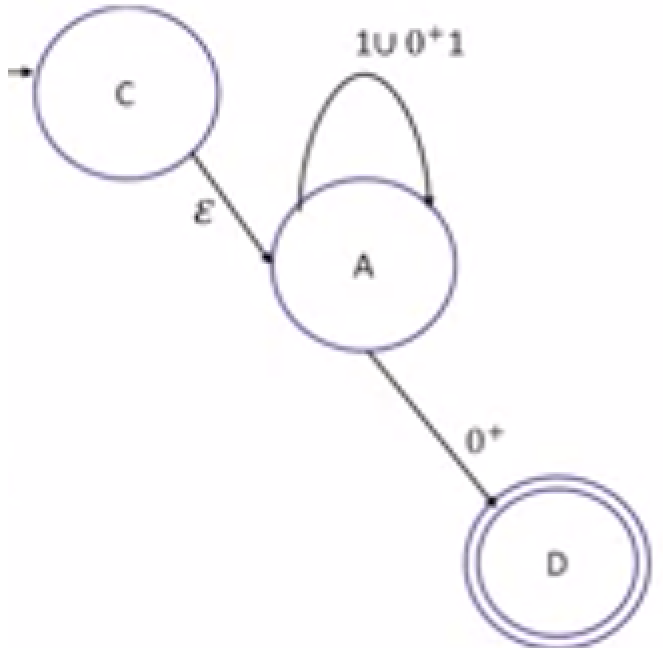


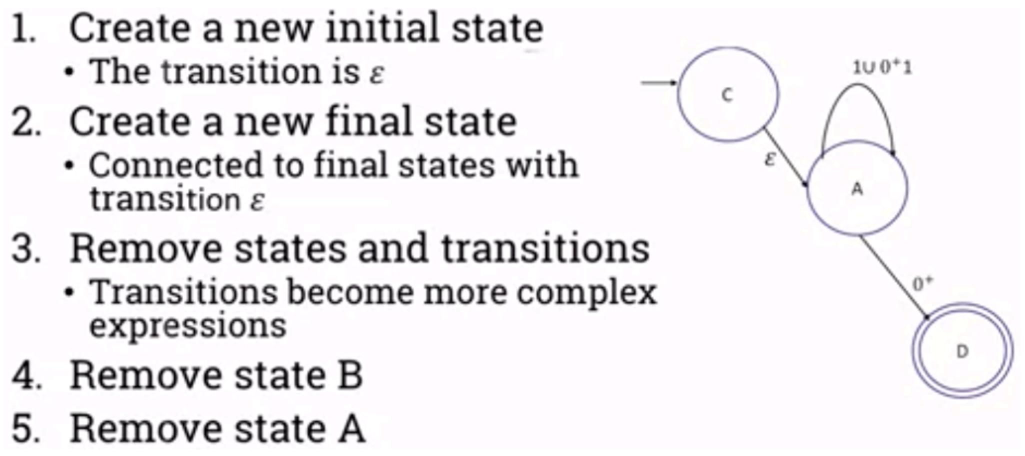
Converting a FA to RE

1. Create a new initial state
   - The transition is ε
2. Create a new final state
   - Connected to final states with transition ε
3. Remove states and transitions
   - Transitions become more complex expressions
4. Remove state B

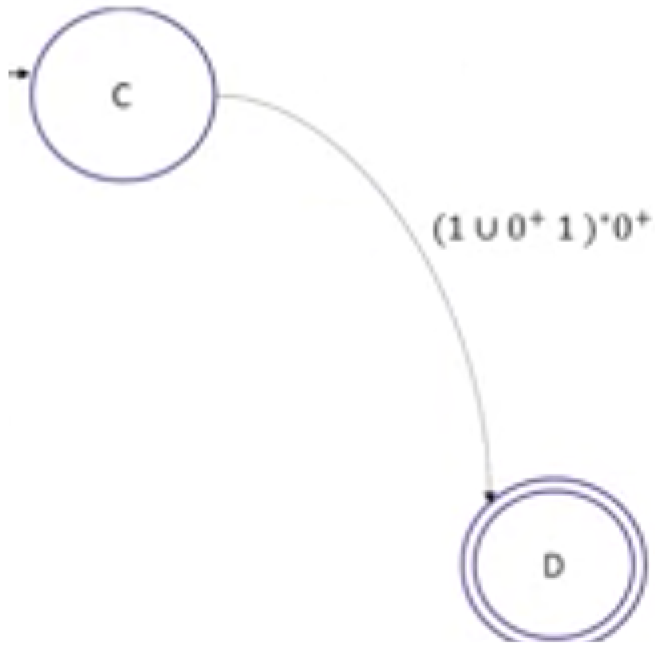



1∪00*1
C
ε
A
00*
D


1∪0+1
C
ε
A
0+
D



1. Create a new initial state
   - The transition is ε
2. Create a new final state
   - Connected to final states with transition ε
3. Remove states and transitions
   - Transitions become more complex expressions
4. Remove state B
5. Remove state A



2. If L = L(R) for some FA, A, then there is a RE, R, such that L(A) = L
   a. proof idea: proof by induction

Week 10 - Non-regular languages & Pumping Lemma

closure properties:

- theorem: If L_1 and L_2 are regular languages on alphabet Σ, then the following languages are also regular:
  - U - L_1 (Σ* - L_1)

    - L_1 ∪ L_2
    - L_1 ∩ L_2
    - L_1 x L_2
    - L_1*

non-regular language — meaning no FA exists that admits it

- cannot test all FA
- no regular expressions exist in the same language
- examples:

    - $L = \{a^n b^n | n \in \mathbb{N}\}$
    - $L = \{xx | x \in \{\mathrm{a}, \mathrm{b}\}^* \}$
    - $L = \{a^{n!} | n \in \mathbb{N}\}$
    - $L = \{xx^R | x \in \{\mathrm{a}, \mathrm{b}\}^* \}$
    - $L = \{a^{n^2} | n \in \mathbb{N}\}$
    - $L = \{a^n | n \in \mathbb{N}, n \text{ is a prime number}\}$

pumping lemma — technique in proving nonregularity

- theorem about regular languages
- states that all regular languages have special property, and if it doesn’t have this property it is NOT regular

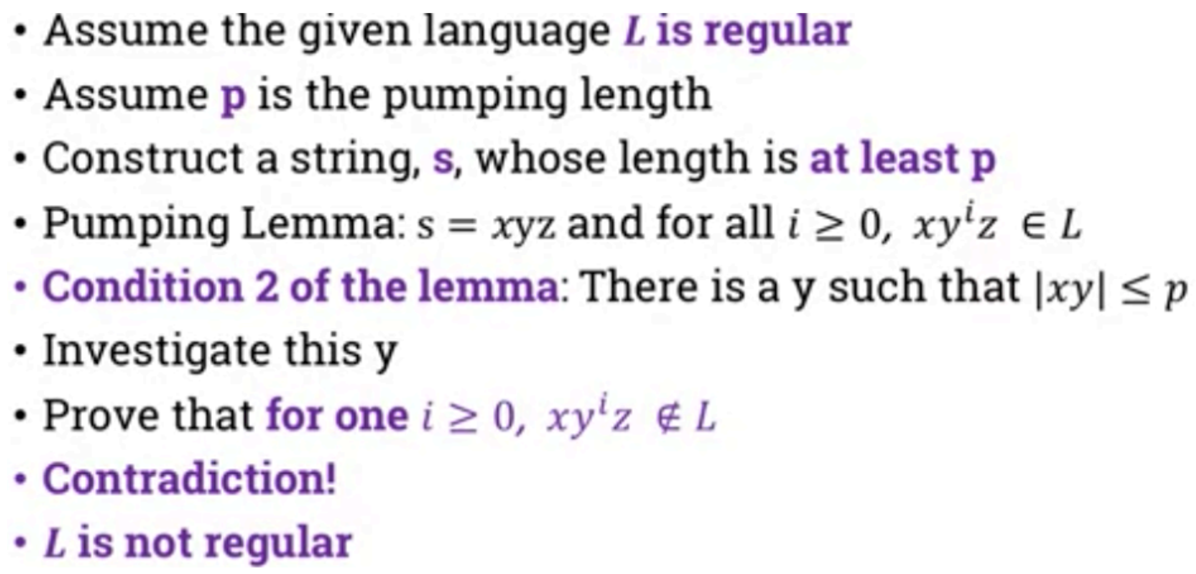
- states that all strings in the language can be "pumped" if they are at least as long as the pumping length
    - means that each such strung contains a section that can be repeated any number of times with the resulting string remaining in the language; basically one can look for a repeating pattern within a certain length/chunk(pumping length) of a string
- pumping lemma theorem:

  if A is a regular language, then there is a number of *p* (pumping length) where if *s* is any string in A of length at least *p,* then *s* may be divided into three pieces, *s = xyz,* satisfying the following conditions:

    1. for each i ≥ 0, $xy^iz \in$ A, (i copies of y are concatenated together)
    2. |y| > 0, and
    3. |xy| ≤ p

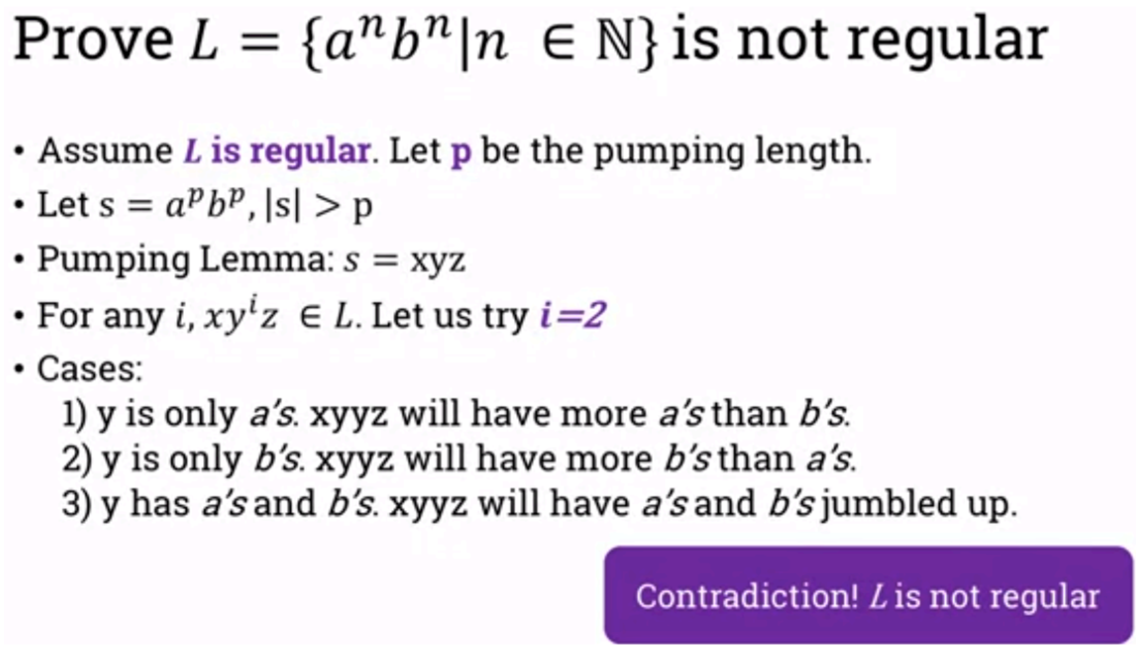
  when *s* is divided into xyz, either x or z may be ϵ, but condition 2 says that y ≠ ϵ. Observe that without condition 2 the theorem would be trivially true. Condition 3 states that the pieces x and y together have length at most *p.*

    - if |s| ≥ p, the s must have a repeated state (Pigeonhole Principle)
- how pumping lemma works:



- Assume the given language ***L* is regular**
- Assume **p** is the pumping length
- Construct a string, **s**, whose length is **at least p**
- Pumping Lemma: $s = xyz$ and for all $i \geq 0,\ xy^iz \in L$
- **Condition 2 of the lemma**: There is a y such that $|xy| \leq p$
- Investigate this y
- Prove that **for one** $i \geq 0,\ xy^iz \notin L$
- **Contradiction!**
- ***L* is not regular**

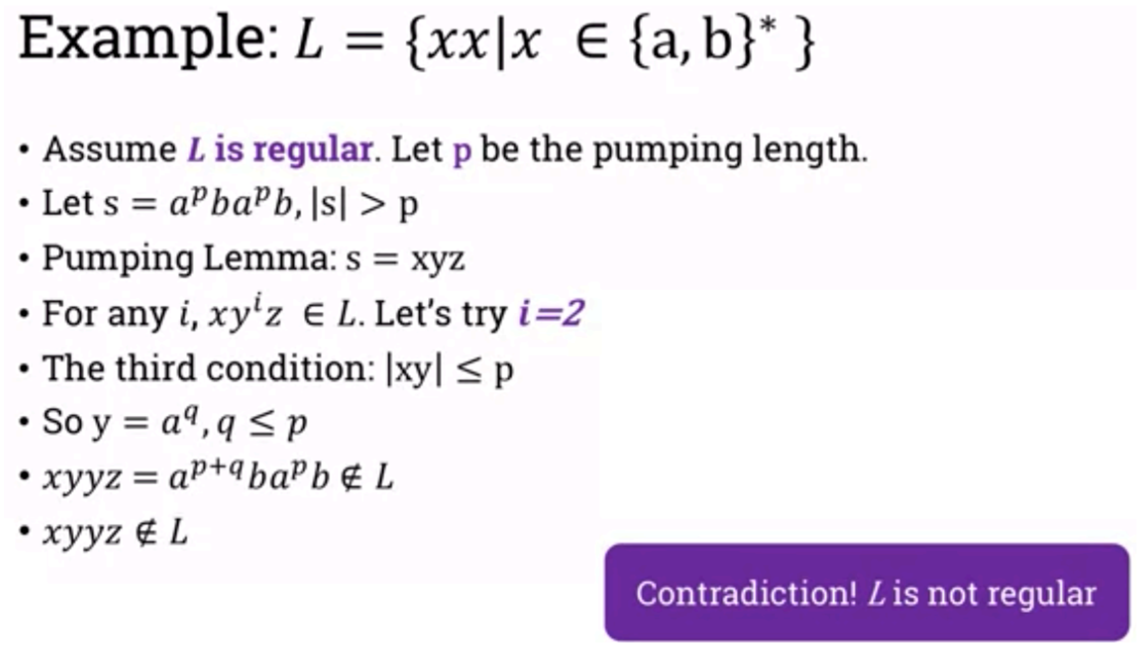
- examples:

1. 

Prove $L = \{a^n b^n | n \in \mathbb{N}\}$ is not regular

- Assume ***L* is regular**. Let **p** be the pumping length.
- Let s = $a^p b^p$, |s| > p
- Pumping Lemma: s = xyz
- For any i, $xy^i z \in L$. Let us try ***i=2***
- Cases:
  1) y is only *a's*. xyyz will have more *a's* than *b's*.
  2) y is only *b's*. xyyz will have more *b's* than *a's*.
  3) y has *a's* and *b's*. xyyz will have *a's* and *b's* jumbled up.

Contradiction! *L* is not regular

2. 

Example: $L = \{xx | x \in \{a, b\}^*\}$

- Assume ***L* is regular**. Let **p** be the pumping length.
- Let s = $a^p b a^p b$, |s| > p
- Pumping Lemma: s = xyz
- For any i, $xy^i z \in L$. Let's try ***i=2***
- The third condition: |xy| ≤ p
- So y = $a^q, q \le p$
- $xyyz = a^{p+q} b a^p b \notin L$
- $xyyz \notin L$

Contradiction! *L* is not regular