CS3102 Theory of Computation

What happens if I try to use the complement construction for DFAs on NFAs?

If I invert the accepting/rejectings states of a NFA, will its language become the complement, why or why not?

Regular Expressions

- A way to describe strings using operations on characters
- Pieces:
  - Literals: Characters from Σ, ε All finite languages are regular!
  - Concatenation $(R_1 R_2)$ Preserves Regularity!
  - Union $(R_1 + R_2)$ Preserves Regularity!
  - Kleene Star (R^*) Preserves Regularity!
    - One or more (R^+), same as RR^* Preserves Regularity!
- Example: $aa(a+b)^*bb$
  - Any string that starts with 2 a's and ends with 2 b's
- Why can we only get regular languages?

Give Regular Expressions

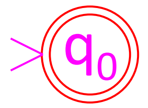
- Σ^*
- $\{\varepsilon\}$
- $EvenA$
- $EvenAOddB$
- $\{w \in \Sigma^* \mid aba \text{ appears somewhere in } w\}$
- $\{0 \text{ or more } a's, 0 \text{ or more } b's, \text{one or more } a's\}$

Regex to NFA

All literal regular expressions have an FSA:

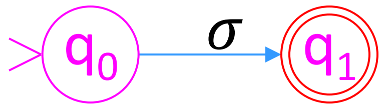
ε

Empty string



$\sigma \in \Sigma$

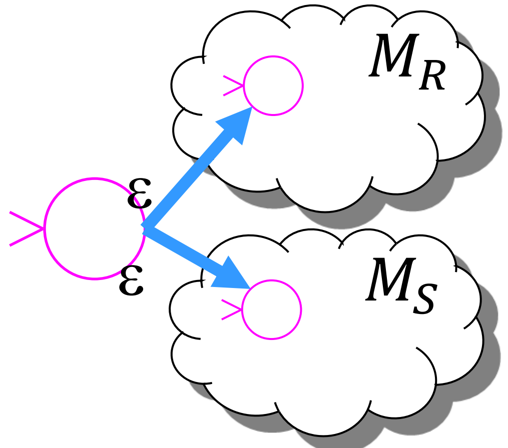
Literal characters



If regular expressions R and S have FSAs, then so do:

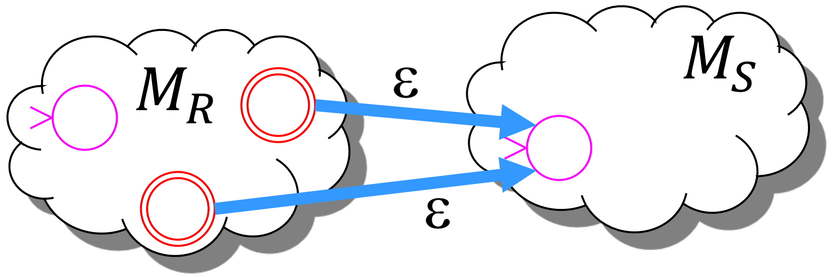
$(R + S)$

union



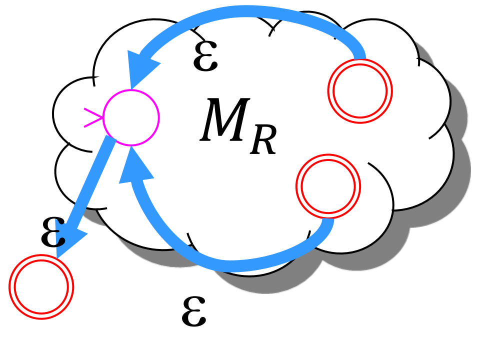
RS

concatenation



R^*

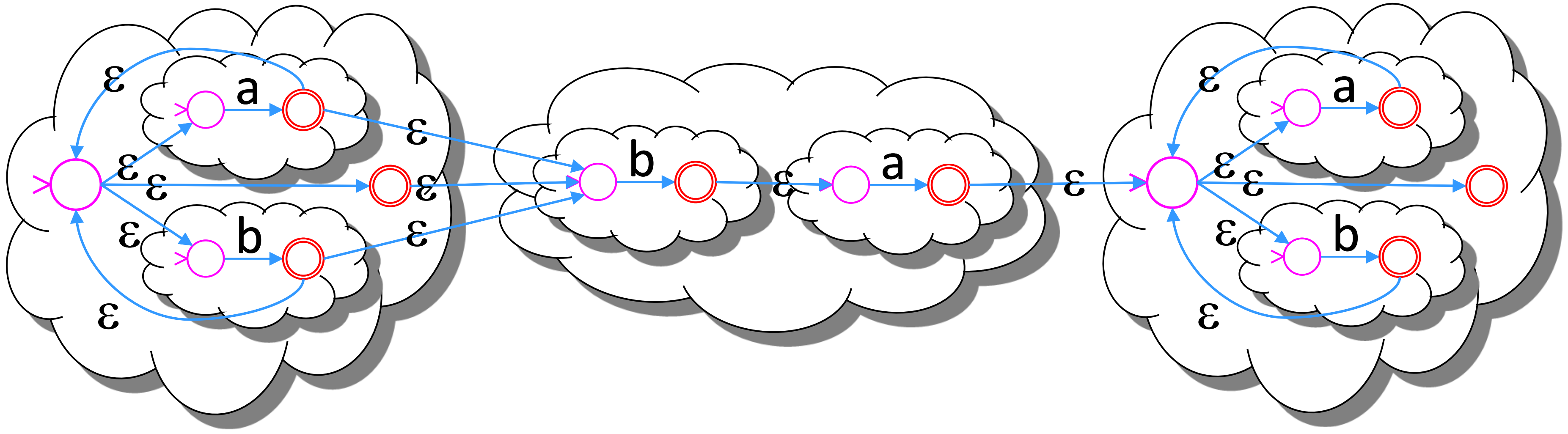
Kleene closure

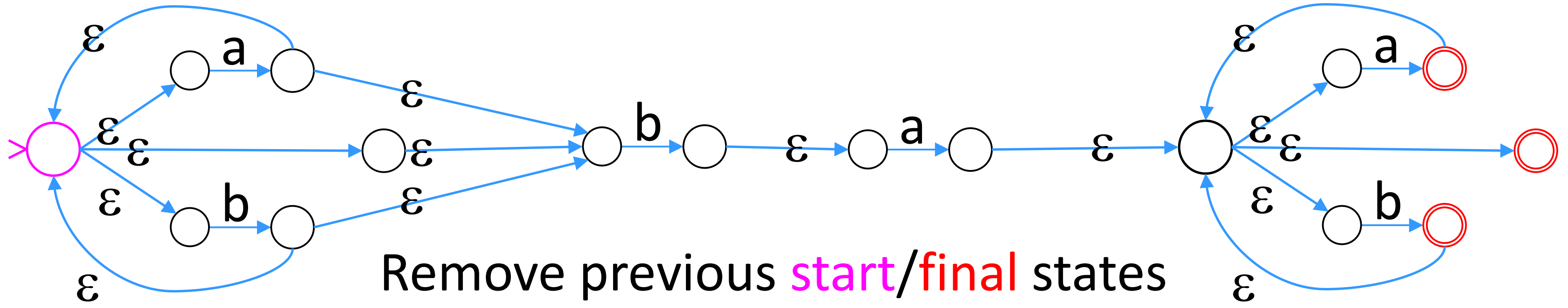


Regex to NFA

Ex: all strings over $\{a, b\}$ where there is a "b" preceding an "a"

$$(a+b)^*ba(a+b)^*$$





Remove previous start/final states

NFA to Regex

- Idea: Make transitions match on regular expressions rather than characters, eliminate states until there is just one transition left, the regex that this transition matches will be the regex for that machine
- We won’t go over details (less important than the other direction). Book shows this in Lemma 1.60.

Non-regular Languages

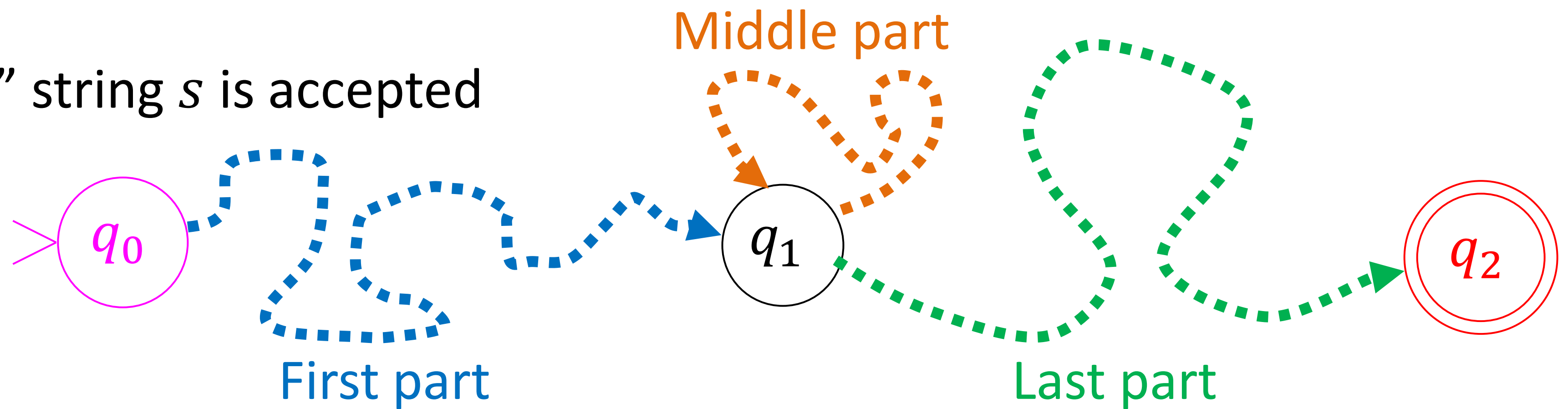
- A language is regular if:
  - There is a NFA/DFA which accepts exactly the strings in the language
  - There is a regular expression which describes exactly the strings in the language
- A lanugage is non-regular if:
  - There is no NFA/DFA which accepts exactly the strings in the language (there will always be false positives/negatives)
  - You cannot write a regular expression which describes exactly the strings in the lanuage (it always misses some or describes too many)
- Examples:
  - $\{a^n b^n \text{ for } n \in \mathbb{N}\}$
  - $\{s \mid s \text{ is a palindrome}\}$

Non-existence Proof!

Proving Non-Regularity

- For a language to be regular, there must be a DFA for it
- That same DFA must work for *every* string in the language (no matter how long)
- If the language is infinite, there must be some string in the language larger than the machine's size
- This long string it must visit at least one state twice on its way to the final state (pigeonhole principle)

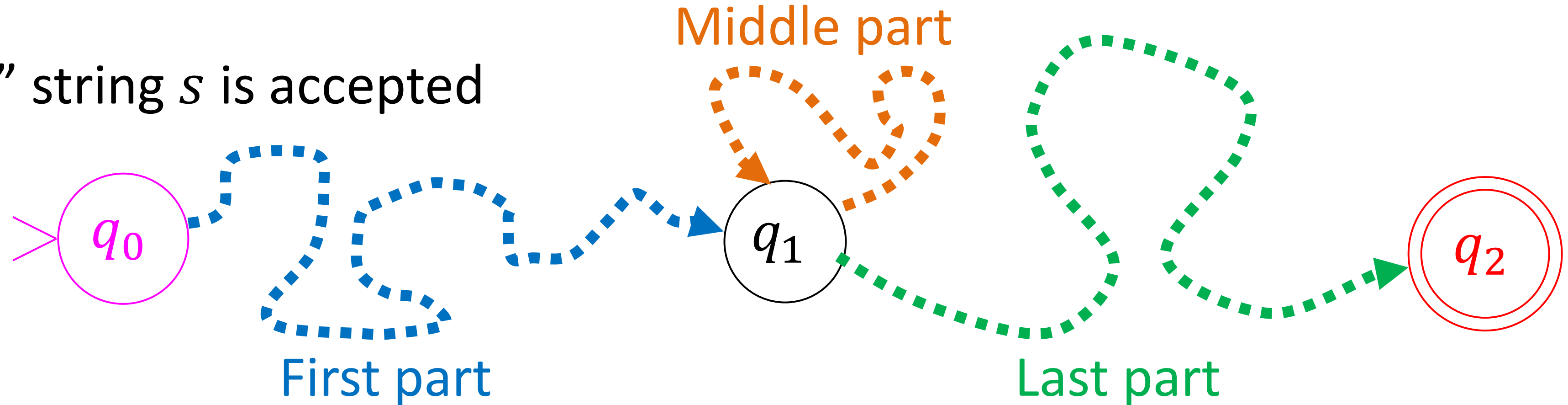
When "long" string s is accepted



“Pumping Lemma” Idea

- Any infinite regular language has a “long” string
- Any “long” string can be broken into 3 parts
  - A first part that takes you from start to **some state**
  - A middle part that takes you back to that **same state**
  - A last part that takes you to a final state
- Copying the middle part times (or skipping it) still makes a path from start to a final state
- If we can’t break up a “long” string into these parts which allows us to “pump” the middle, the language isn’t regular

When “long” string s is accepted



Pumping Lemma

- If L is a regular language, then there is a number $p \in \mathbb{N}$ such that, for *any* string $s \in L$ where $|s| > p$, we can find strings x, y, z where $s = xyz$ satisfying all of:
  1. For each $i \geq 0$, $xy^iz \in L$
  2. $|y| > 0$
  3. $|xy| \leq p$
- Use contrapositive to show a language is not regular:
  - if you can find a long string where you can't do this, the language is not regular

$L = a^n b^n$ is not regular

- Let the "pumping length" be p
- Consider the string $a^p b^p$, note that $|a^p b^p| > p$, so if L is regular this string can be pumped
- If we had $a^p b^p = xyz$ there are 3 options for what y could be:
    1. $y \in a^+$
        - In this case, xy^iz has too many a's
    2. $y \in b^+$
        - In this case, xy^iz has too many b's
    3. $y \in a^+b^+$
        - In this case, xy^iz has a's and b's out of order
- Since $a^p b^p$ cannot be "pumped", L is not regular

$L = \{w \in \Sigma^* | w = w^R\}$ is not regular

- Let the "pumping length" be p
- Consider the string $a^p b^p a^p$, note that $|a^p b^p a^p| > p$, so if L is regular this string can be pumped
- If we had $a^p b^p a^p = xyz$ there are 3 options for what y could be:
  1. $y \in a^+$
     - In this case, xy^iz has too many a's before/after the b's
  2. $y \in a^+b^+ + \mathrm{b}^+\mathrm{a}^+$
     - In this case, xy^iz is not palindrome
  3. $y \in b^+$
     - In this case, $|xy| \geq p$ has too many b's
- Since $a^p b^p$ cannot be "pumped", L is not regular

$L = a^n b^m$ where $n \neq m$ is not regular

- Idea: Use closure properties!
- Assume toward reaching a contradiction that L is regular
- In this case, $\overline{L}$ is regular too (since complement preserves regularity)
- What strings are in $\overline{L}$?
  - Those that have a b before an a, i.e. $(a+b)^* ba (a+b)^*$
  - Those from $a^* b^*$ where the number of a's matches the number of b's
- Since $\overline{L}$ is regular, so is $\overline{L} \cap a^* b^*$ (since both are regular and intersection preserves regularity)
- $\overline{L} \cap a^* b^* = a^n b^n$ which we know isn't regular!

Pumpable $\neq$ Regular

- The pumping lemma can only be used to show NON-regularity
  - If this languages is not pumpable, then it is not regular
- The pumping lamma cannot be used to prove a language is regular
  - Some non-regular languages are pumpable

A pumpable non-regular language

- $L = a^i b^j c^k$ where $i = 1 \Rightarrow j = k$
- $L \cap ab^*c^* = ab^n c^n$ which is not regular (it can't be pumped), so L isn't regular either
- I can pump everything of form ab^*c^* by letting $y = a$
- I can pump everything else by letting y have either just a's or just b's