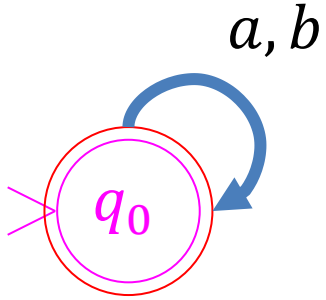
CS3102 Theory of Computation

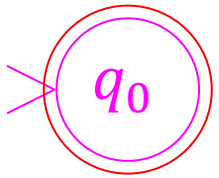
Draw NFAs for the following languages over $\Sigma = \{a, b\}$ (using at most the given number of states):

- Σ^*
  - (1 state)
- $\{\varepsilon\}$
  - (1 state)
- $\emptyset$
  - (1 state)
- $\{0 \text{ or more } a's, 0 \text{ or more } b's, \text{one or more } a's\}$
  - (3 states)
- $\{w \in \Sigma^* \mid aba \text{ appears somewhere in } w\}$
  - (4 states)

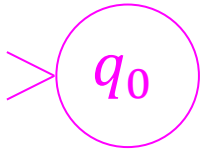
NFAs



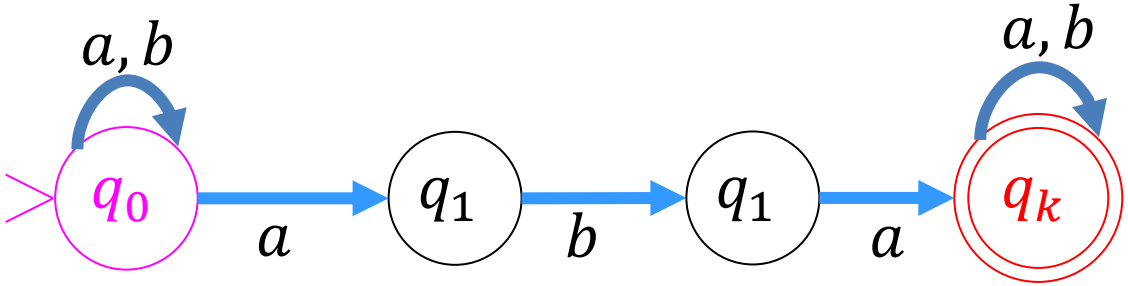
Σ^*



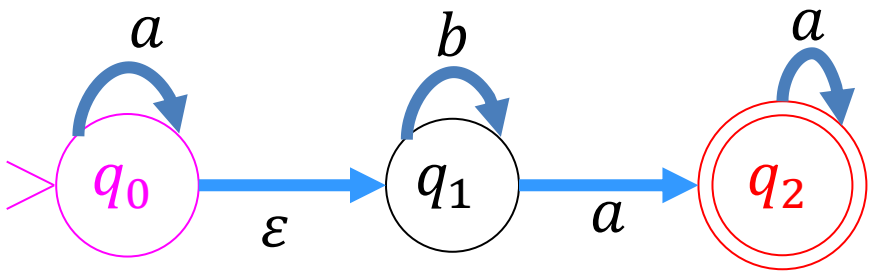
$\{\varepsilon\}$



$\emptyset$



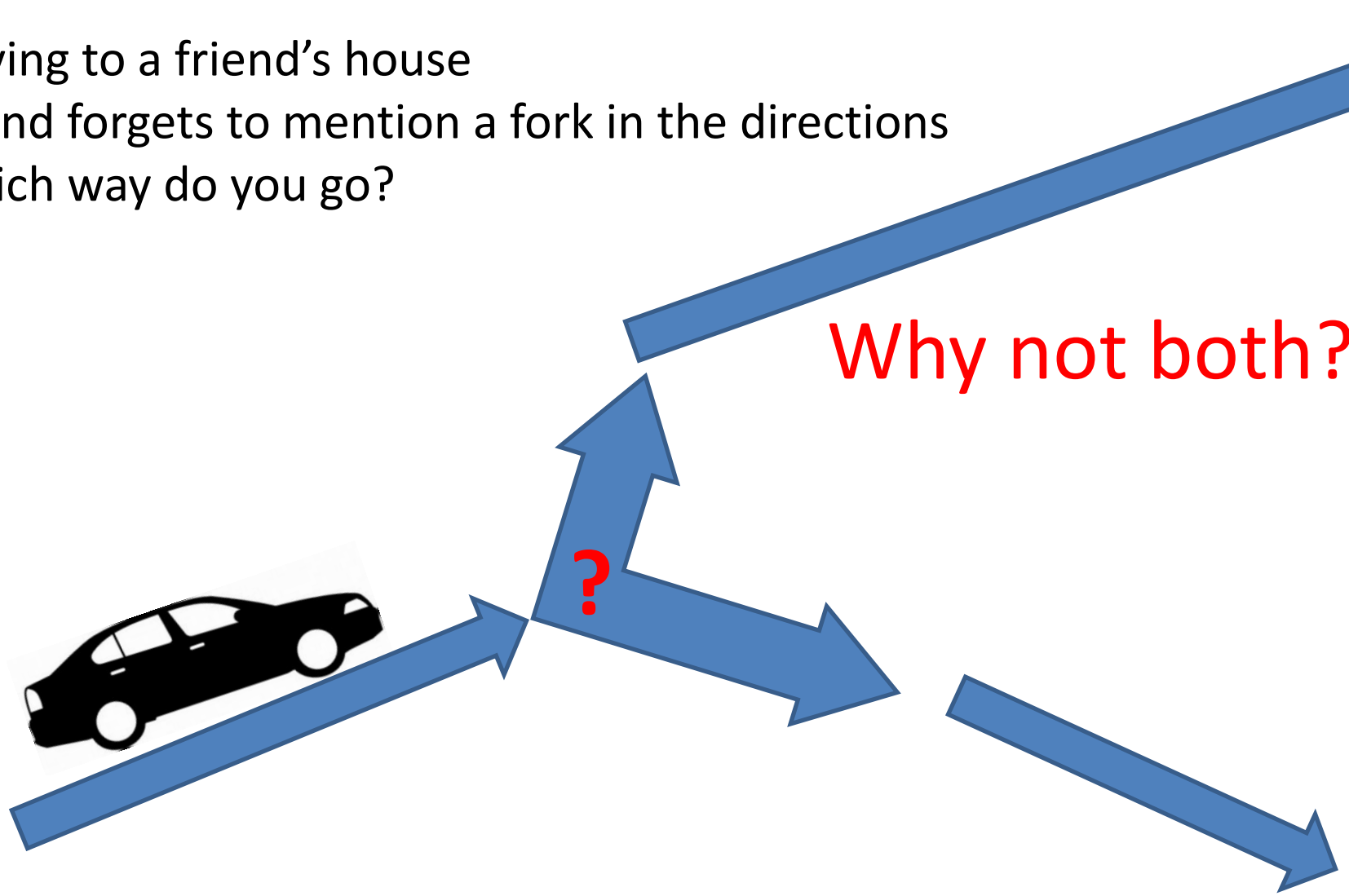
$\{w \in \Sigma^* \mid aba \text{ appears somewhere in } w\}$



$\{0 \text{ or more } a's, 0 \text{ or more } b's, \text{one or more } a's\}$

Nondeterminism

Driving to a friend's house
Friend forgets to mention a fork in the directions
Which way do you go?



Why not both?

Nondeterminism in Automata

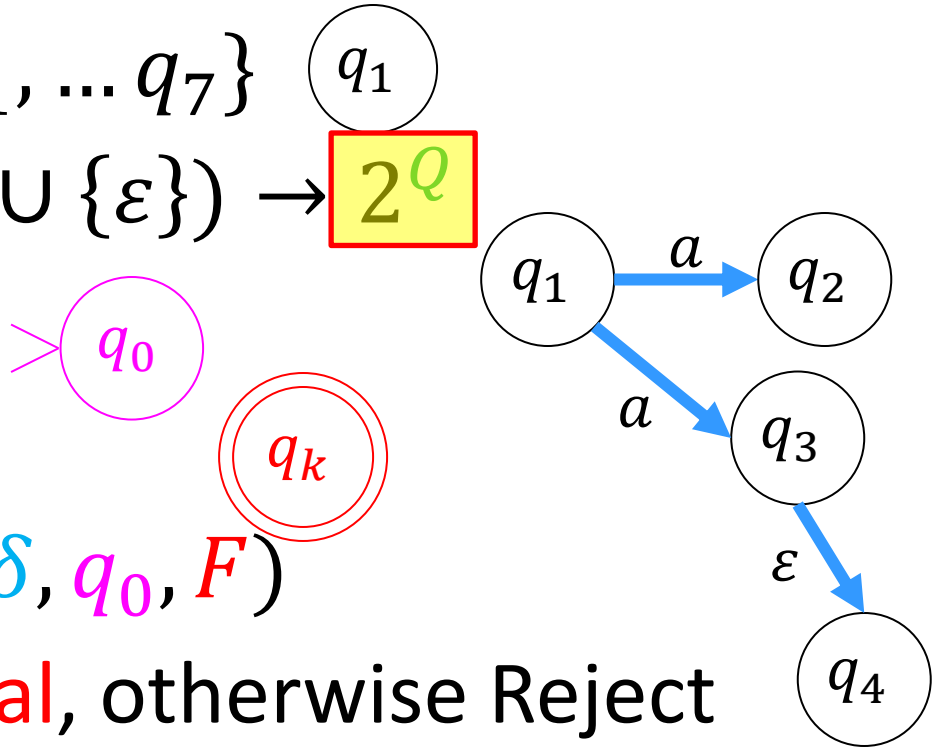
- Your machine can be in multiple states at once
- Accepts if any if the states it ends in are accepting states
- Relax restrictions:
  - Exactly one transition per symbol (can make multiple without consuming a symbol)
  - There must be exactly one outgoing transition for each symbol for every state (will allow 0 to many of them)
- Keep restriction:
  - One start state

NFAs = DFAs

- We can convert any NFA into a DFA
- Powerset Construction
  - Idea: Make a new DFA whose states each represent a subset of states from the original machine.

Nondeterministic Finite State Automata

- Basic idea: a NFA is a "machine" that changes states while processing symbols, one at a time.
- Finite set of states: $Q = \{q_0, q_1, \ldots q_7\}$
- Transition function: $\delta: 2^Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$
- Initial state: $q_0 \in Q$
- Final states: $F \subseteq Q$
- Finite state automaton is $M = (Q, \Sigma, \delta, q_0, F)$
- Accept if any states we end in are Final, otherwise Reject only when none of the states are final
- If no transition defined, that "branch" rejects



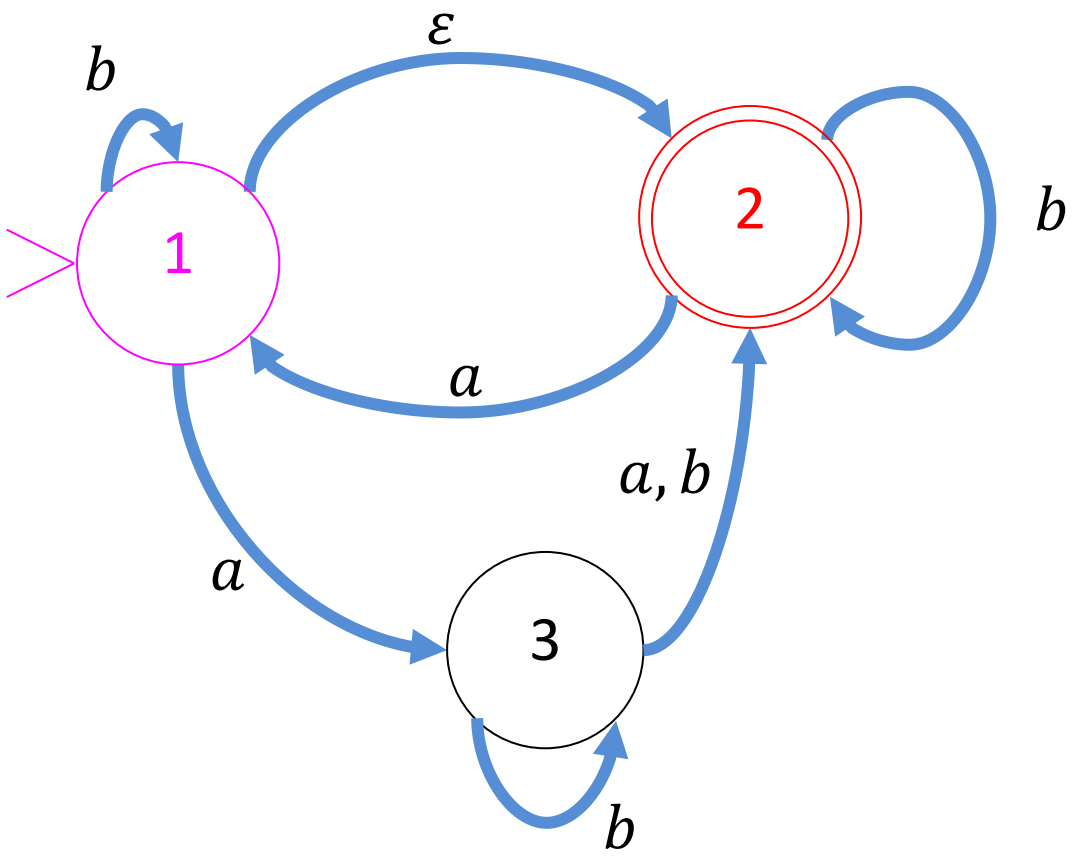
Powerset Construction

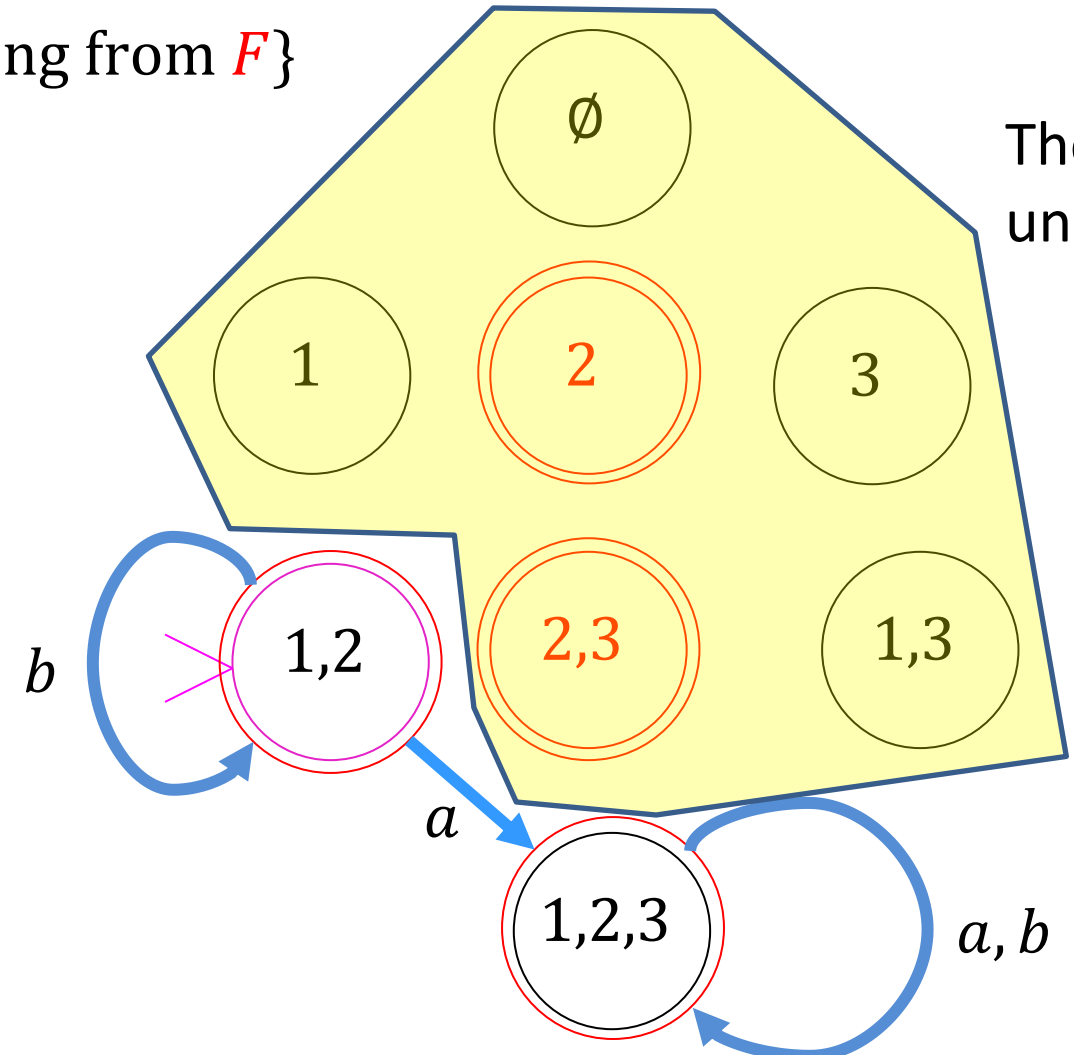
$Q_{power} = 2^Q$

$\delta_{power}(q_1, \sigma) = \bigcup_{q \in q_1} \delta(q, \sigma)$ (all states that any active state transitions to)

$q_{o,power}$ = everything reachable from q_0

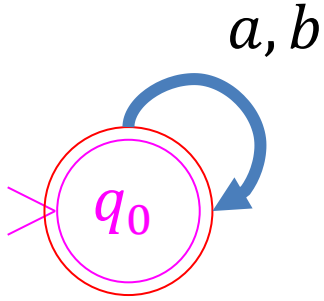
F_{power} = {any state containing something from F}



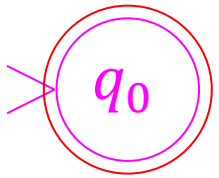


These states are unreachable.

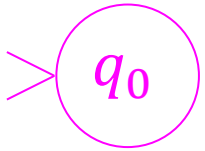
Do Powerset Construction on these



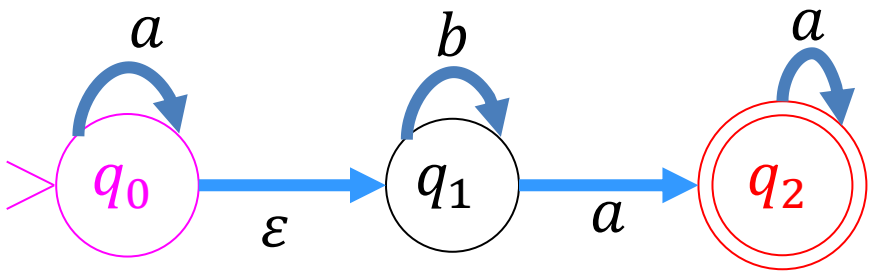
Σ^*



$\{\varepsilon\}$

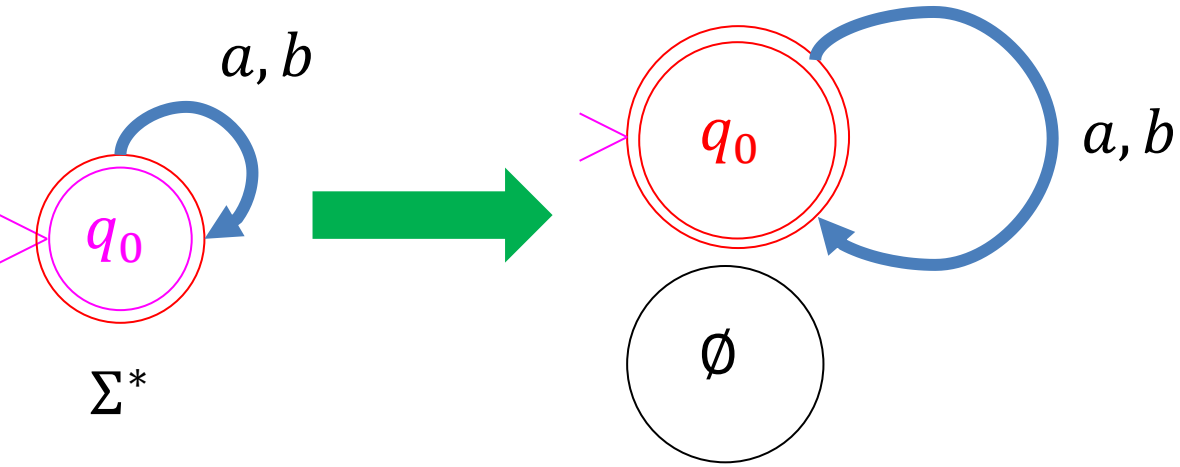


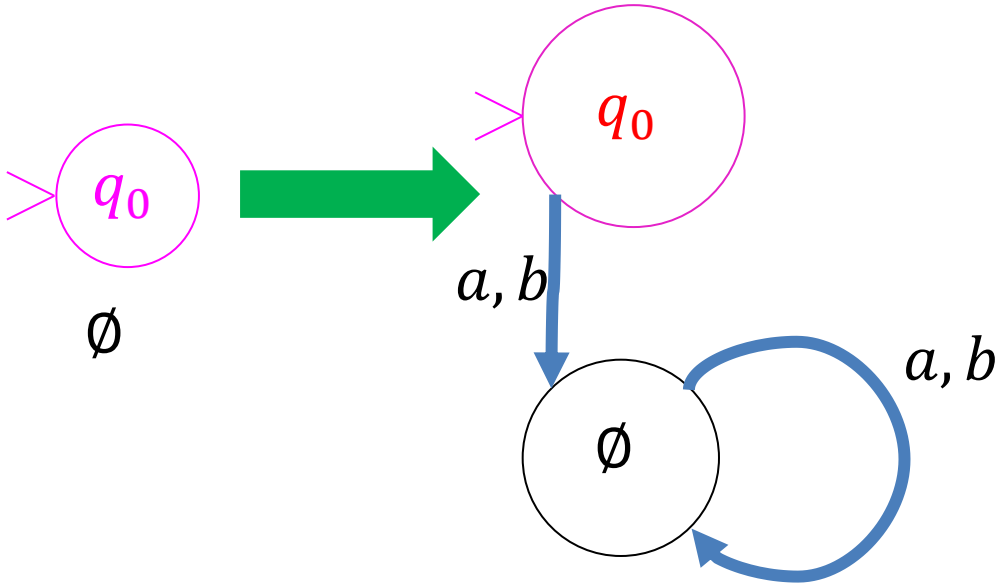
$\emptyset$

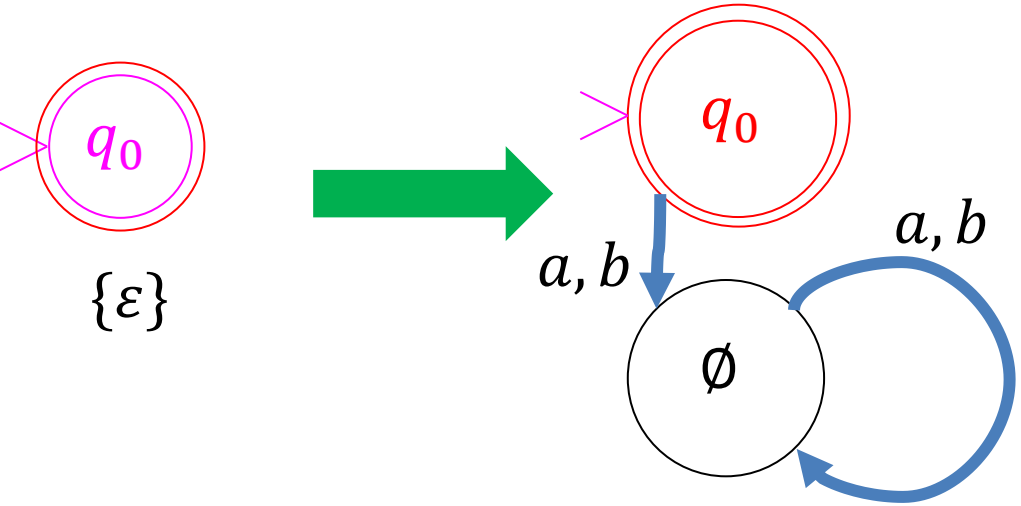


$\{0 \text{ or more } a's, 0 \text{ or more } b's, \text{one or more } a's\}$

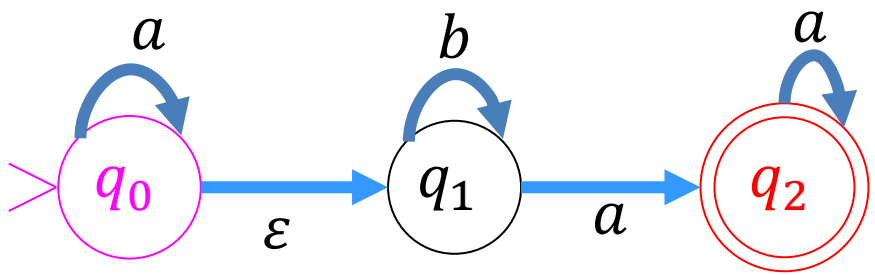
Powerset Construction



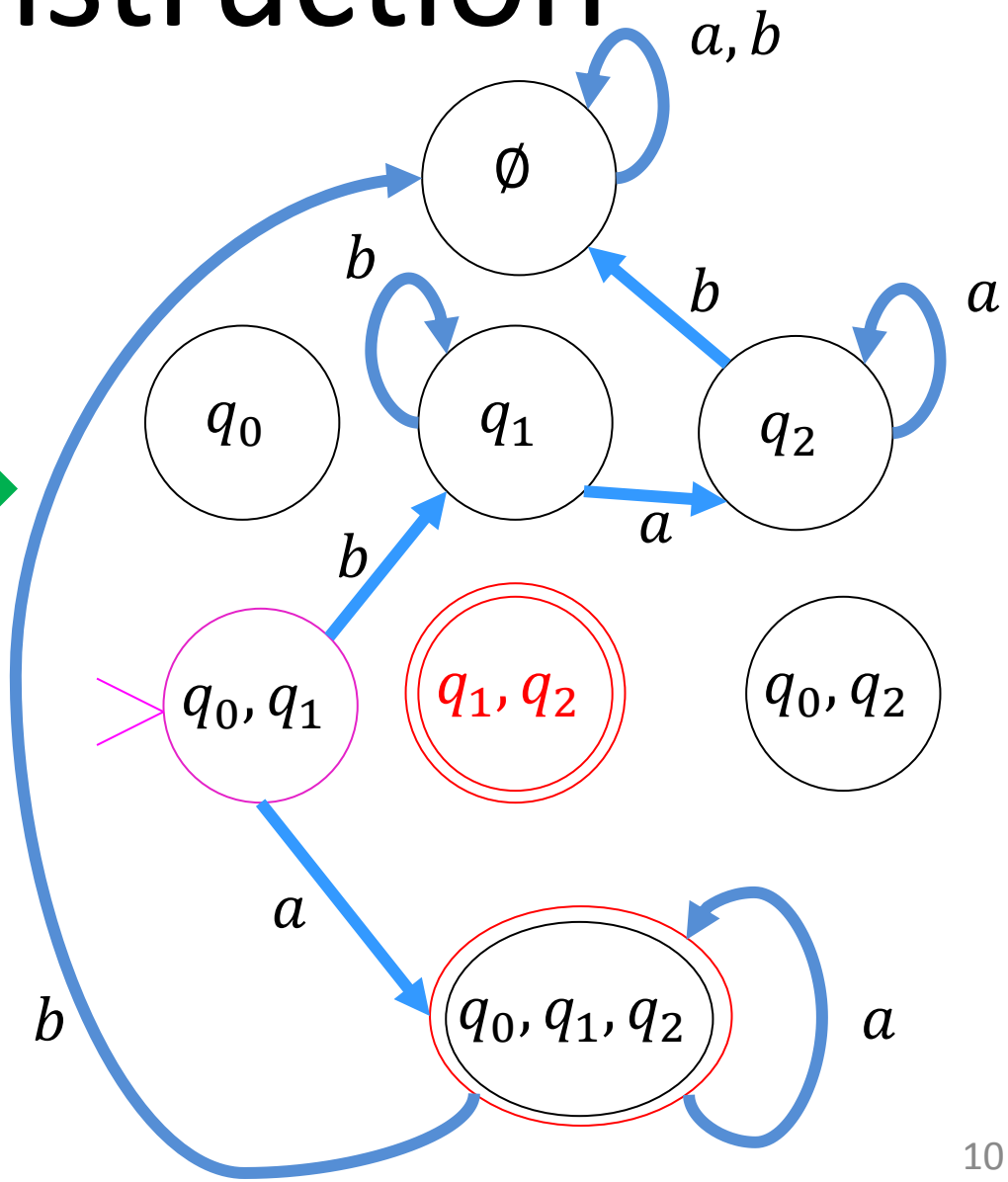




Powerset Construction



{0 or more a's, 0 or more b's, one or more a's}



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

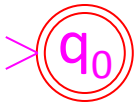
- Σ^*
- $\{\varepsilon\}$
- $EvenA$
- $EvenAOddB$
- $\{w \in \Sigma^* \mid aba \text{ appears somewhere in } w\}$
- $\{0 \text{ or more } a's, 0 \text{ or more } b's, \text{one or more } a's\}$

Regex to NFA

All literal regular expressions have an FSA:

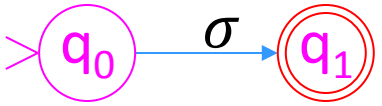
ε

Empty string



$\sigma \in \Sigma$

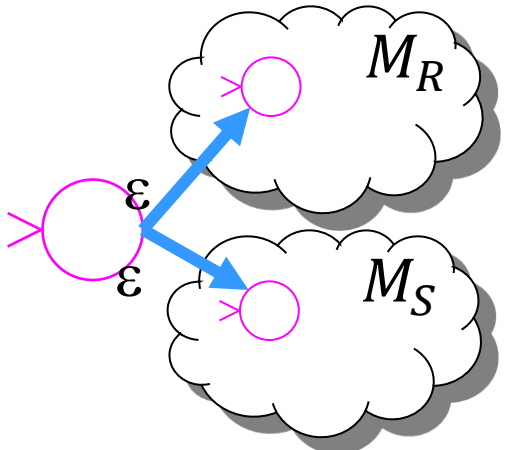
Literal characters



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

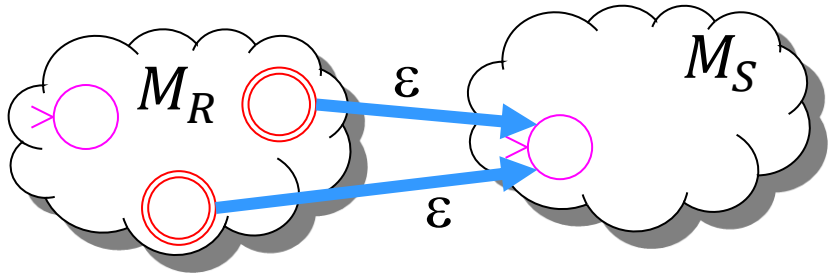
$(R + S)$

union



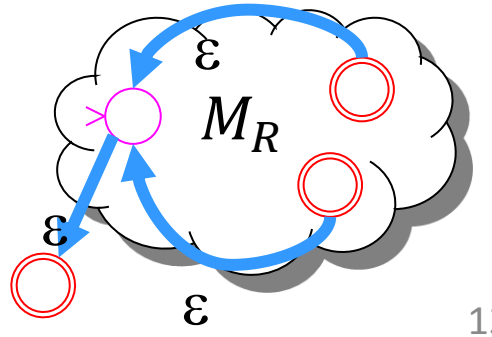
RS

concatenation



R^*

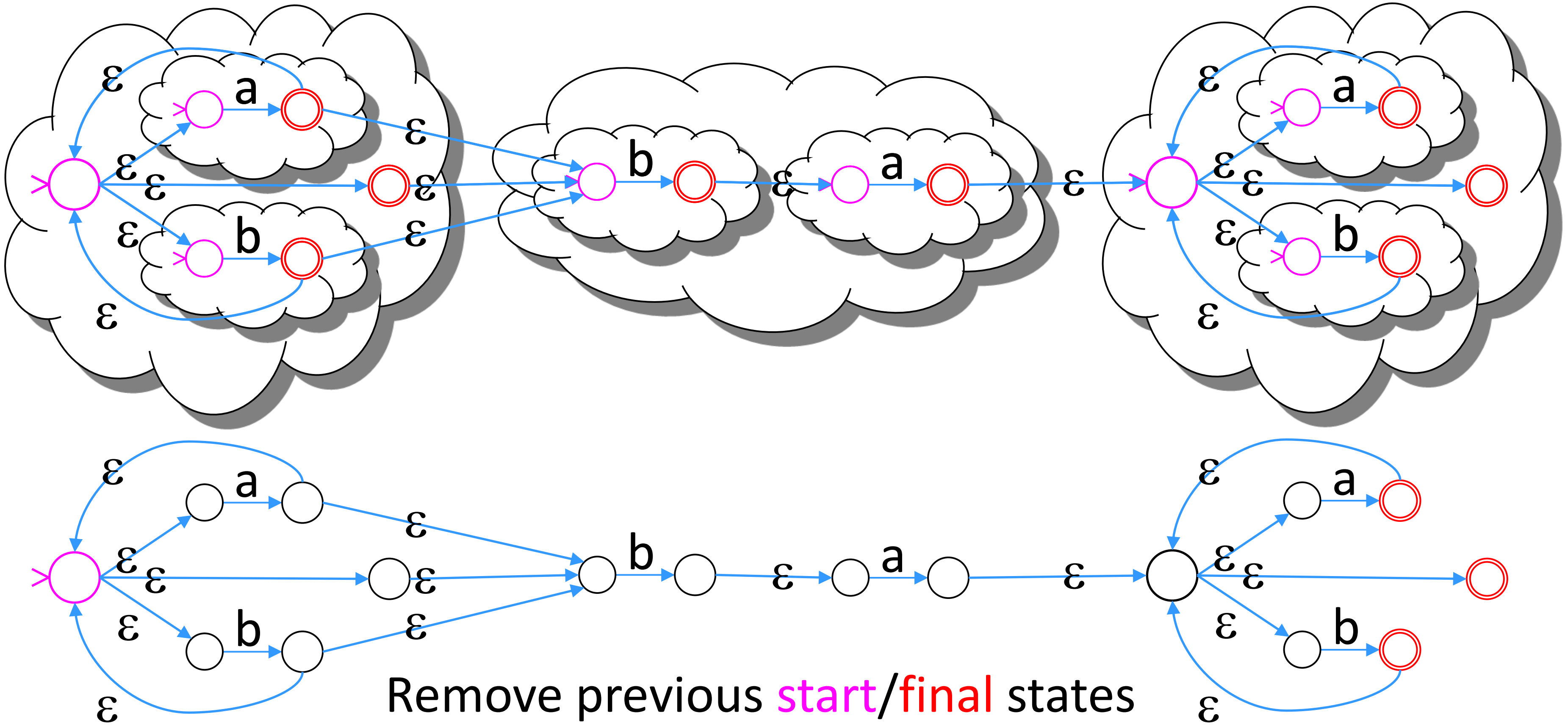
Kleene closure



Regex to NFA

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

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



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

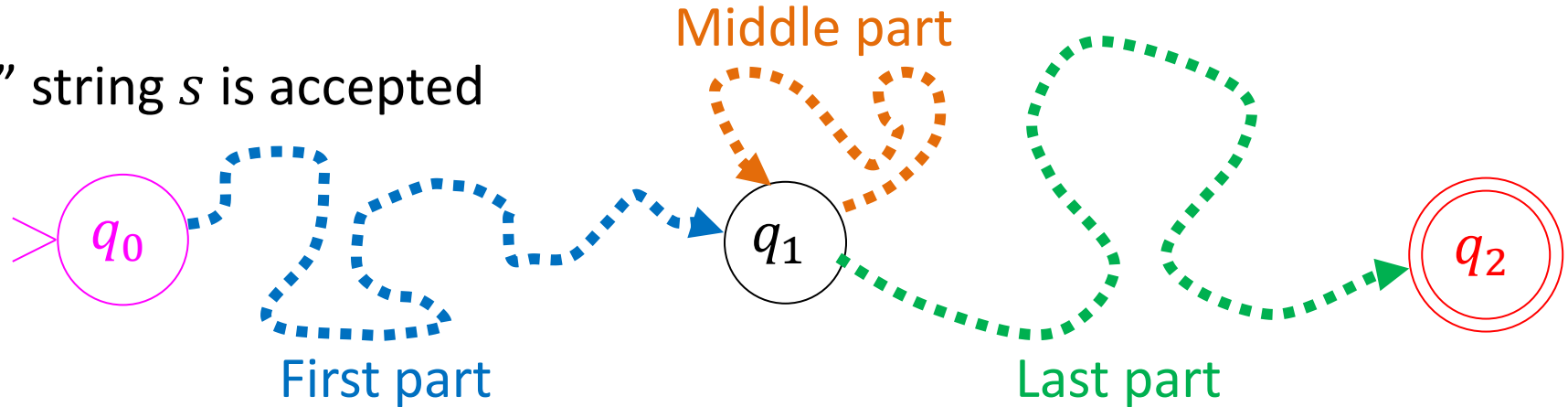
- A language is regular if:
  - There is a NFA/DFA which accepts exactly the strings in the language
  - There is a regular expression which describes exactly the strings in the language
- A lanugage is non-regular if:
  - There is no NFA/DFA which accepts exactly the strings in the language (there will 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)

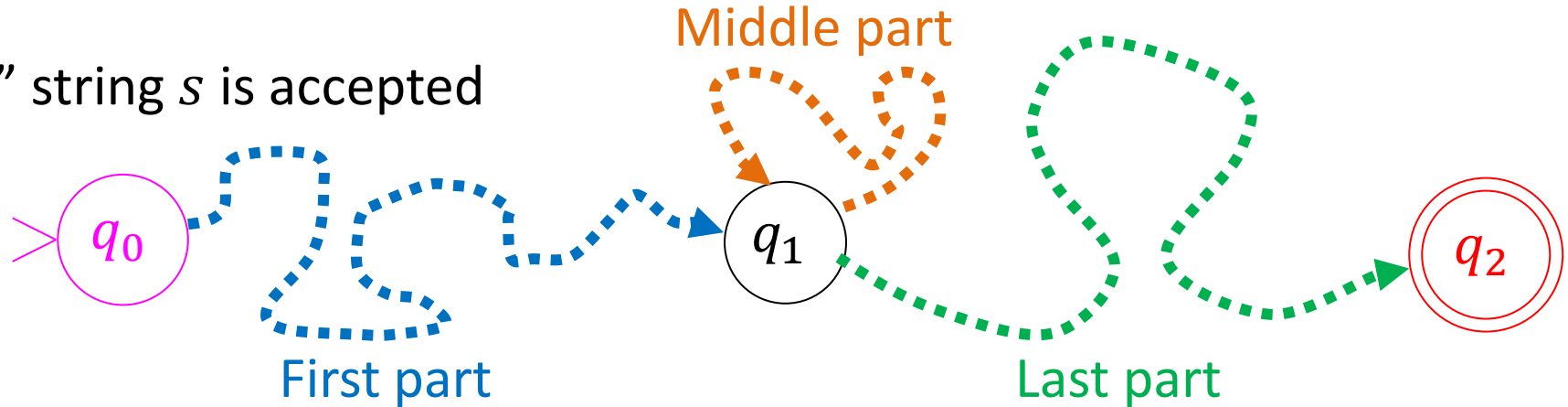
When "long" string s is accepted



“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, $xy^i z$ has too many a's
  2. $y \in b^+$
     - In this case, $xy^i z$ has too many b's
  3. $y \in a^+ b^+$
     - In this case, $xy^i z$ has a's and b's out of order
- Since $a^p b^p$ cannot be "pumped", L is not regular