CS 341: Foundations of CS II

Marvin K. Nakayama
Computer Science Department
New Jersey Institute of Technology
Newark, NJ 07102

Chapter 1
Regular Languages

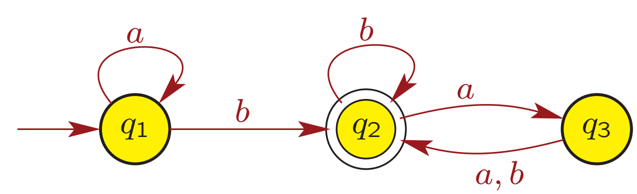
Contents

Introduction

- Now introduce a simple model of a computer having a finite amount of memory.
- This type of machine will be known as a **finite-state machine** or **finite automaton**.
- Basic idea how a finite automaton works:
  - It is presented an input string w over an alphabet Σ; i.e., $w \in \Sigma^*$.
  - It reads in the symbols of w from left to right, one at a time.
  - After reading the last symbol, it indicates if it accepts or rejects the string.
- These machines are useful for string matching, compilers, etc.

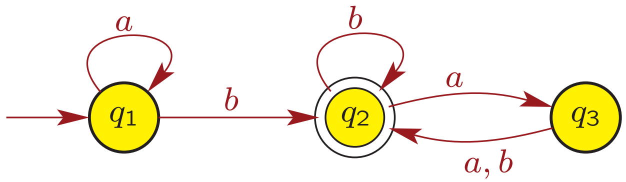
Deterministic Finite Automata (DFA)

Example: DFA with alphabet $\Sigma = \{a, b\}$:



- q_1, q_2, q_3 are the **states**.
- q_1 is the **start state** as it has an arrow coming into it from nowhere.
- q_2 is an **accept state** as it is drawn with a double circle.

Deterministic Finite Automata



- Edges tell how to move when in a state and a symbol from Σ is read.
- DFA is fed **input string** $w \in \Sigma^*$. After reading last symbol of w,
  - if DFA is in an accept state, then string is **accepted**
  - otherwise, it is **rejected**.
- Process the following strings over $\Sigma = \{a, b\}$ on above machine:
  - $abaa$ is accepted $\quad \to q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \xrightarrow{a} q_2$
  - aba is rejected $\quad \to q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$
  - ε is rejected $\quad \to q_1$

Formal Definition of DFA

Definition: A **deterministic finite automaton** (**DFA**) is a 5-tuple

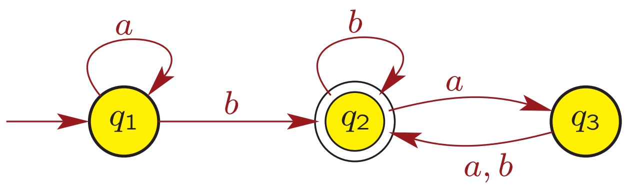
$$M = (Q, \Sigma, \delta, q_0, F),$$

where

1. Q is a **finite** set of states.
2. Σ is an alphabet, and the DFA processes strings over Σ.
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**.
   - δ defines label on each edge.
4. $q_0 \in Q$ is the **start state** (or **initial state**).
5. $F \subseteq Q$ is the set of **accept states** (or **final states**).

Remark: Sometimes refer to DFA as simply a **finite automaton (FA)**.

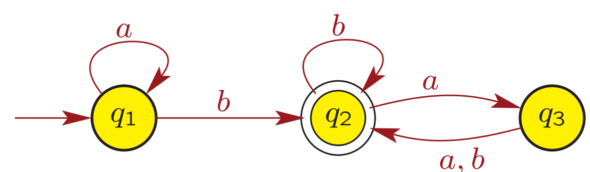
Transition Function of DFA



Transition function $\delta : Q \times \Sigma \to Q$ works as follows:

- For each state and for each symbol of the input alphabet, the function δ tells which (one) state to go to next.
- Specifically, if $r \in Q$ and $\ell \in \Sigma$, then $\delta(r, \ell)$ is the state that the DFA goes to when it is in state r and reads in ℓ, e.g., $\delta(q_2, a) = q_3$.
- For each pair of state $r \in Q$ and symbol $\ell \in \Sigma$,
  - there is **exactly one** arc leaving r labeled with ℓ.
- Thus, there is no choice in how to process a string.
  - So the machine is **deterministic**.

Example of DFA



$M = (Q, \Sigma, \delta, q_1, F)$ with

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- $\delta : Q \times \Sigma \to Q$ is described as

	a	b
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

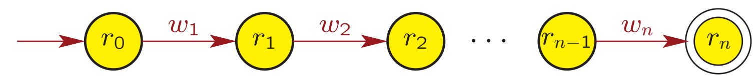
- q_1 is the start state
- $F = \{q_2\}$.

How a DFA Computes

- DFA is presented with an input string $w \in \Sigma^*$.
- DFA begins in the start state.
- DFA reads the string one symbol at a time, starting from the left.
- The symbols read in determine the sequence of states visited.
- Processing ends after the last symbol of w has been read.
- After reading the entire input string
  - if DFA ends in an accept state, then input string w is **accepted**;
  - otherwise, input string w is **rejected**.

Formal Definition of DFA Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
- String $w = w_1 w_2 \cdots w_n \in \Sigma^*$, where each $w_i \in \Sigma$ and $n \geq 0$.
- Then M **accepts** w if there exists a sequence of states $r_0, r_1, r_2, \ldots, r_n \in Q$ such that
  1. $r_0 = q_0$
     - first state r_0 in the sequence is the start state of DFA;
  2. $r_n \in F$
     - last state r_n in the sequence is an accept state;
  3. $\delta(r_i, w_{i+1}) = r_{i+1}$ for each $i = 0, 1, 2, \ldots, n-1$
     - sequence of states corresponds to valid transitions for string w.

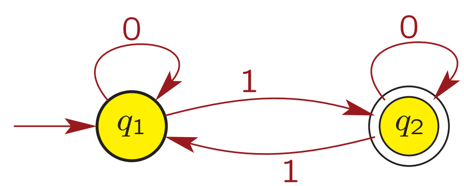


Language of Machine

- **Definition:** If A is the set of all strings that machine M accepts, then we say
  - $A = L(M)$ is the **language of machine** M, and
  - M **recognizes** A.
- If machine M has input alphabet Σ, then $L(M) \subseteq \Sigma^*$.
- **Definition:** A language is **regular** if it is recognized by some DFA.

Examples of Deterministic Finite Automata

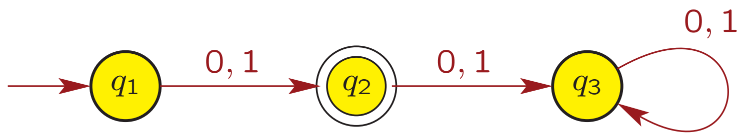
Example: Consider the following DFA M_1 with alphabet $\Sigma = \{0, 1\}$:



Remarks:

- 010110 is accepted, but 0101 is rejected.
- $L(M_1)$ is the language of strings over Σ in which the total number of 1's is odd.
- Can you come up with a DFA that recognizes the language of strings over Σ having an even number of 1's ?

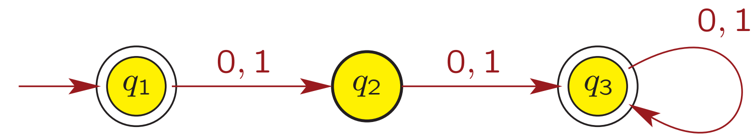
Example: Consider the following DFA M_2 with alphabet $\Sigma = \{0, 1\}$:



Remarks:

- $L(M_2)$ is language of strings over Σ that have length 1, i.e.,
$$L(M_2) = \{ w \in \Sigma^* \mid |w| = 1 \}$$
- Recall that $\overline{L(M_2)}$, the complement of $L(M_2)$, is the set of strings over Σ not in $L(M_2)$, i.e.,
$$\overline{L(M_2)} = \Sigma^* - L(M_2).$$
Can you come up with a DFA that recognizes $\overline{L(M_2)}$?

Example: Consider the following DFA M_3 with alphabet $\Sigma = \{0, 1\}$:



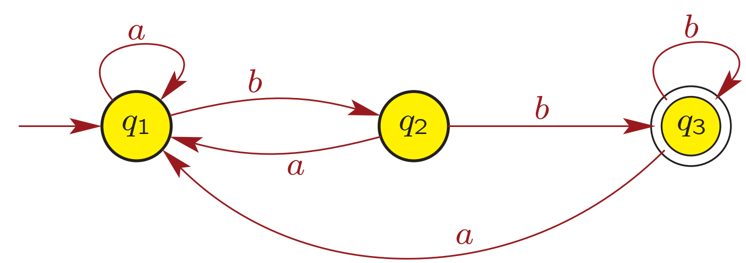
Remarks:

- $L(M_3)$ is the language of strings over Σ that **do not** have length 1, i.e.
$$L(M_3) \;=\; \overline{L(M_2)} \;=\; \{w \in \Sigma^* \mid |w| \neq 1\}$$
- DFA can have more than one accept state.
- Start state can also be an accept state.
- In general, a DFA accepts ε if and only if the start state is also an accept state.

Constructing DFA for Complement

- In general, given a DFA M for language A,
we can make a DFA $\overline{M}$ for $\overline{A}$ from M by
  - changing all accept states in M into non-accept states in $\overline{M}$,
  - changing all non-accept states in M into accept states in $\overline{M}$,
- More formally, suppose language A over alphabet Σ has a DFA
$$M = (\, Q,\ \Sigma,\ \delta,\ q_1,\ F\,).$$
- Then, a DFA for the complementary language $\overline{A}$ is
$$\overline{M} = (\, Q,\ \Sigma,\ \delta,\ q_1,\ Q - F\,).$$
where $Q, \Sigma, \delta, q_1, F$ are the same as in DFA M.
- Why does this work?

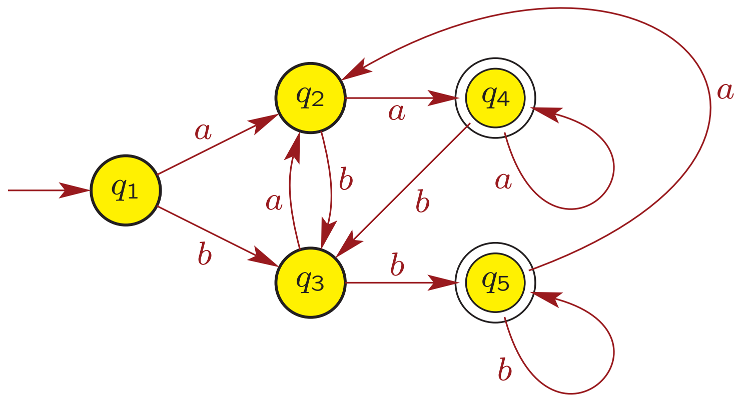
Example: Consider the following DFA M_4 with alphabet $\Sigma = \{a, b\}$:



Remarks:

- $L(M_4)$ is the language of strings over Σ that end with bb, i.e.,
$$L(M_4) = \{ w \in \Sigma^* \mid w = sbb \text{ for some } s \in \Sigma^* \}.$$
- Note that $abbb \in L(M_4)$ and $bba \notin L(M_4)$.

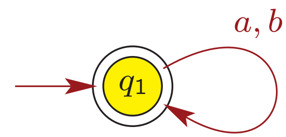
Example: Consider the following DFA M_5 with alphabet $\Sigma = \{a, b\}$:



$$L(M_5) = \{ w \in \Sigma^* \mid w = saa \text{ or } w = sbb \text{ for some string } s \in \Sigma^* \}.$$

Note that $abbb \in L(M_5)$ and $bba \notin L(M_5)$.

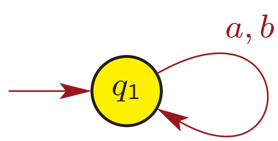
Example: Consider the following DFA M_6 with alphabet $\Sigma = \{a, b\}$:



Remarks:

- This DFA accepts all possible strings over Σ, i.e.,
$$L(M_6) = \Sigma^*.$$
- In general, any DFA in which all states are accept states recognizes the language Σ^*.

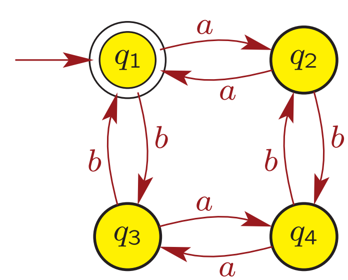
Example: Consider the following DFA M_7 with alphabet $\Sigma = \{a, b\}$:



Remarks:

- This DFA accepts no strings over Σ, i.e.,
$$L(M_7) = \emptyset.$$
- In general,
  - a DFA may have no accept states, i.e., $F = \emptyset \subseteq Q$.
  - any DFA with no accept states recognizes the language $\emptyset$.

Example: Consider the following DFA M_8 with alphabet $\Sigma = \{a, b\}$:



- DFA moves left or right on a.
- DFA moves up or down on b.

- This DFA recognizes the language of strings over Σ having
  - even number of a's and
  - even number of b's.
- Note that $ababaa \in L(M_8)$ and $bba \notin L(M_8)$.

Some Operations on Languages

- Let A and B be languages.
- Recall we previously defined the operations:
  - **Union**:
    $$A \cup B = \{\, w \mid w \in A \text{ or } w \in B \,\}.$$
  - **Concatenation**:
    $$A \circ B = \{\, vw \mid v \in A,\, w \in B \,\}.$$
  - **Kleene star**:
    $$A^* = \{\, w_1 w_2 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in A \,\}.$$

Closed under Operation

- Recall that a collection S of objects is **closed** under operation f if applying f to members of S always returns an object still in S.
  - e.g., $\mathcal{N} = \{1, 2, 3, \ldots\}$ is closed under addition but not subtraction.
- Previously saw that given a DFA M_1 for language A, can construct DFA M_2 for complementary language $\overline{A}$.
  - Make all accept states in M_1 into non-accept states in M_2.
  - Make all non-accept states in M_1 into accept states in M_2.
- Thus, the class of regular languages is closed under complementation.
  - i.e., if A is a regular language, then $\overline{A}$ is a regular language.

Regular Languages Closed Under Union

Theorem 1.25
The class of regular languages is closed under union.

- i.e., if A_1 and A_2 are regular languages, then so is $A_1 \cup A_2$.

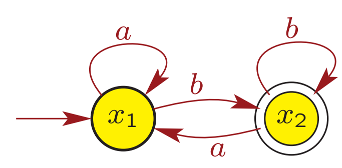
Proof Idea:

- Suppose A_1 is regular, so it has a DFA M_1.
- Suppose A_2 is regular, so it has a DFA M_2.
- $w \in A_1 \cup A_2$ if and only if $w \in A_1$ **or** $w \in A_2$.
- $w \in A_1 \cup A_2$ if and only if w is accepted by M_1 **or** M_2.
- Need DFA M_3 to accept a string w iff w is accepted by M_1 **or** M_2.
- Construct M_3 to keep track of where the input would be if it were simultaneously running on both M_1 and M_2.
- Accept string if and only if M_1 **or** M_2 accepts.

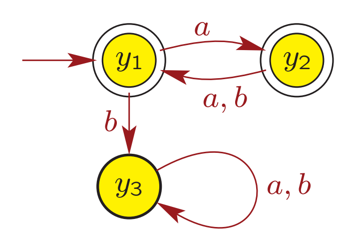
Example: Consider the following DFAs and languages over $\Sigma = \{a, b\}$:

- DFA M_1 recognizes language $A_1 = L(M_1)$
- DFA M_2 recognizes language $A_2 = L(M_2)$

DFA M_1 for A_1

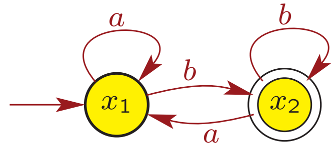


DFA M_2 for A_2

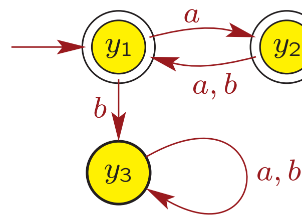


- We now want a DFA M_3 for $A_1 \cup A_2$.

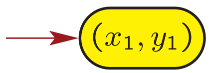
DFA M_1 for A_1



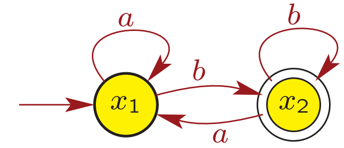
DFA M_2 for A_2



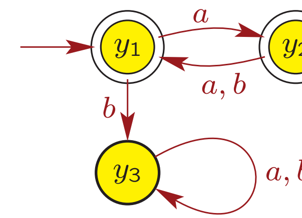
Step 1 to build DFA M_3 for $A_1 \cup A_2$: Begin in start states for M_1 and M_2



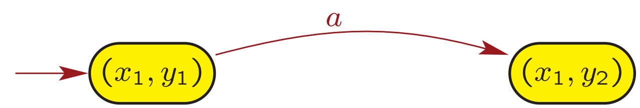
DFA M_1 for A_1



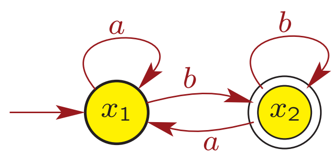
DFA M_2 for A_2



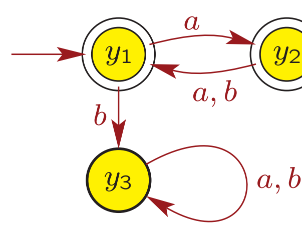
Step 2: From (x_1, y_1) on input a, M_1 moves to x_1, and M_2 moves to y_2.



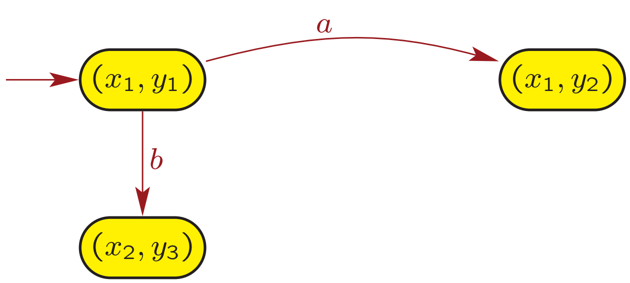
DFA M_1 for A_1



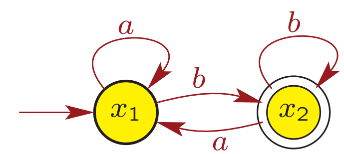
DFA M_2 for A_2



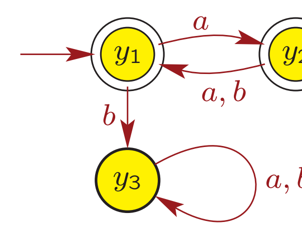
Step 3: From (x_1, y_1) on input b, M_1 moves to x_2, and M_2 moves to y_3.



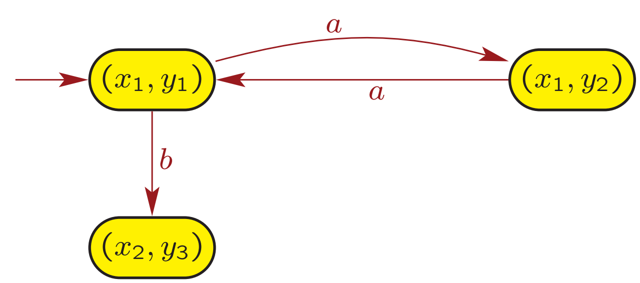
DFA M_1 for A_1



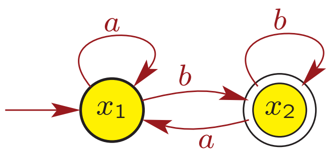
DFA M_2 for A_2



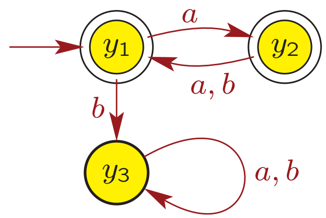
Step 4: From (x_1, y_2) on input a, M_1 moves to x_1, and M_2 moves to y_1.



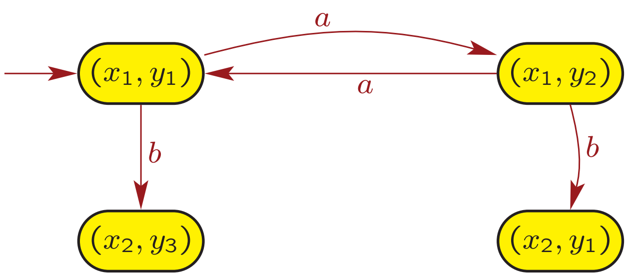
DFA M_1 for A_1



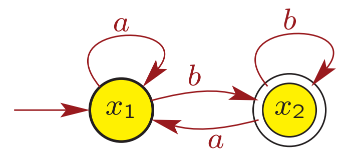
DFA M_2 for A_2



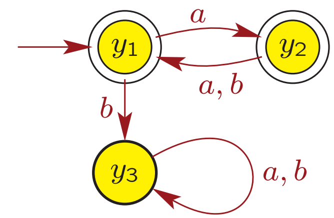
Step 5: From (x_1, y_2) on input b, M_1 moves to x_2, and M_2 moves to y_1, ...



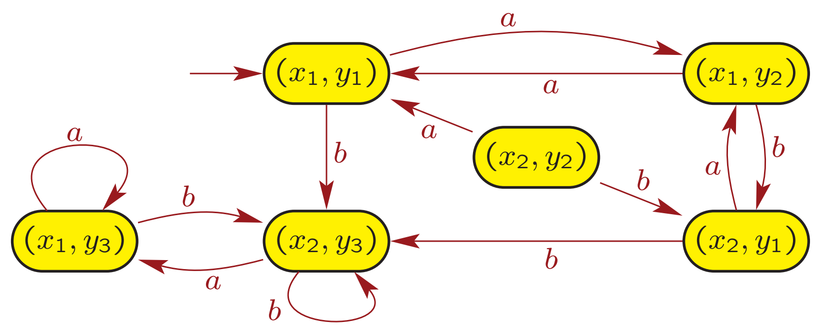
DFA M_1 for A_1



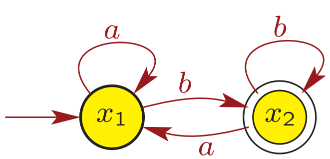
DFA M_2 for A_2



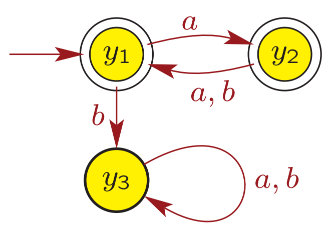
Continue until each state has outgoing edge for each symbol in Σ.



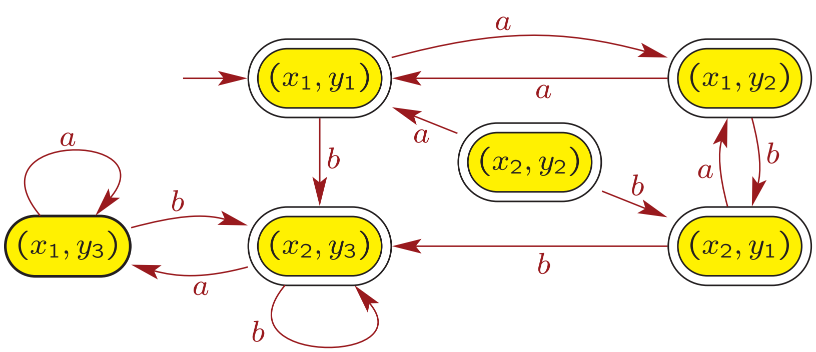
DFA M_1 for A_1



DFA M_2 for A_2



Accept states for DFA M_3 for $A_1 \cup A_2$ have accept state from M_1 or M_2



Proof that Regular Languages Closed Under Union

- Suppose A_1 and A_2 are defined over the same alphabet Σ.
- Suppose A_1 recognized by DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.
- Suppose A_2 recognized by DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
- Define DFA $M_3 = (Q_3, \Sigma, \delta_3, q_3, F_3)$ for $A_1 \cup A_2$ as follows:
  - Set of states of M_3 is
  $$Q_3 = Q_1 \times Q_2 = \{ (x, y) \mid x \in Q_1, \ y \in Q_2 \}.$$
  - The alphabet of M_3 is Σ.
  - M_3 has transition function $\delta_3 : Q_3 \times \Sigma \to Q_3$ such that for $x \in Q_1$, $y \in Q_2$, and $\ell \in \Sigma$,
  $$\delta_3(\,(x, y),\, \ell\,) \;=\; (\,\delta_1(x, \ell),\ \delta_2(y, \ell)\,)\,.$$
  - The start state of M_3 is
  $$q_3 = (q_1, q_2) \in Q_3.$$

- The set of accept states of M_3 is

$$F_3 = \{ (x,y) \in Q_1 \times Q_2 \mid x \in F_1 \text{ or } y \in F_2 \} = [F_1 \times Q_2] \cup [Q_1 \times F_2].$$

- Because $Q_3 = Q_1 \times Q_2$,
  - number of states in new machine M_3 is $|Q_3| = |Q_1| \cdot |Q_2|$.
- Thus, $|Q_3| < \infty$ because $|Q_1| < \infty$ and $|Q_2| < \infty$.

∎

Remark:

- We can leave out a state $(x,y) \in Q_1 \times Q_2$ from Q_3 if (x,y) is not reachable from M_3's initial state (q_1, q_2).
- This would result in fewer states in Q_3, but still we have $|Q_1| \cdot |Q_2|$ as an upper bound for $|Q_3|$; i.e., $|Q_3| \leq |Q_1| \cdot |Q_2| < \infty$.

Regular Languages Closed Under Intersection

Theorem
The class of regular languages is closed under intersection.

- i.e., if A_1 and A_2 are regular languages, then so is $A_1 \cap A_2$.

Proof Idea:

- A_1 has DFA M_1.
- A_2 has DFA M_2.
- $w \in A_1 \cap A_2$ if and only if $w \in A_1$ **and** $w \in A_2$.
- $w \in A_1 \cap A_2$ if and only if w is accepted by both M_1 **and** M_2.
- Need DFA M_3 to accept string w iff w is accepted by M_1 **and** M_2.
- Construct M_3 to simultaneously keep track of where the input would be if it were running on both M_1 and M_2.
- Accept string if and only if both M_1 **and** M_2 accept.

Regular Languages Closed Under Concatenation

Theorem 1.26
Class of regular languages is closed under concatenation.

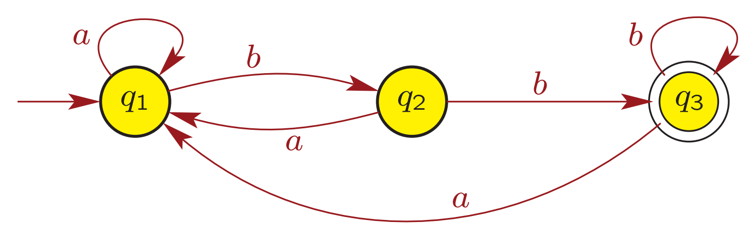
- i.e., if A_1 and A_2 are regular languages, then so is $A_1 \circ A_2$.

Remark:

- It is possible (but cumbersome) to directly construct a DFA for $A_1 \circ A_2$ given DFAs for A_1 and A_2.
- There is a simpler way if we introduce a new type of machine.

Nondeterministic Finite Automata

- In any DFA, the next state the machine goes to on any given symbol is uniquely determined.



- This is why these machines are **deterministic**.
- Remember that the transition function in a DFA is defined as

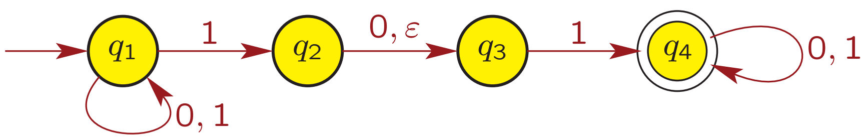
$$\delta : Q \times \Sigma \to Q.$$

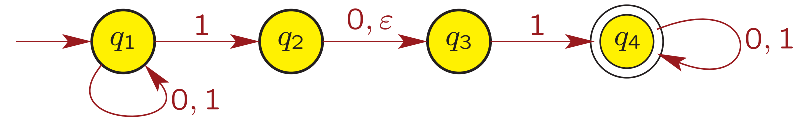
- Because range of δ is Q, fcn δ always returns a **single state**.
- DFA has exactly one transition leaving each state for each symbol.
  - $\delta(q, \ell)$ tells what state the edge out of q labeled with ℓ leads to.

Nondeterminism

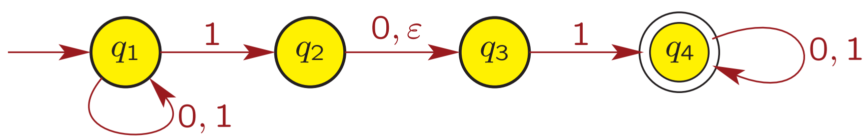
- **Nondeterministic finite automata (NFAs)** allow for several or no choices to exist for the next state on a given symbol.
- For a state q and symbol $\ell \in \Sigma$, NFA can have
  - multiple edges leaving q labelled with the same symbol ℓ
  - no edge leaving q labelled with symbol ℓ
  - edges leaving q labelled with ε
    - can take ε-edge without reading any symbol from input string.

Example: NFA N_1 with alphabet $\Sigma = \{0, 1\}$.



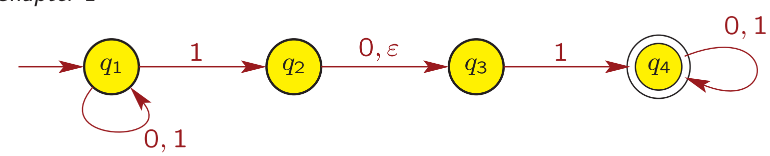


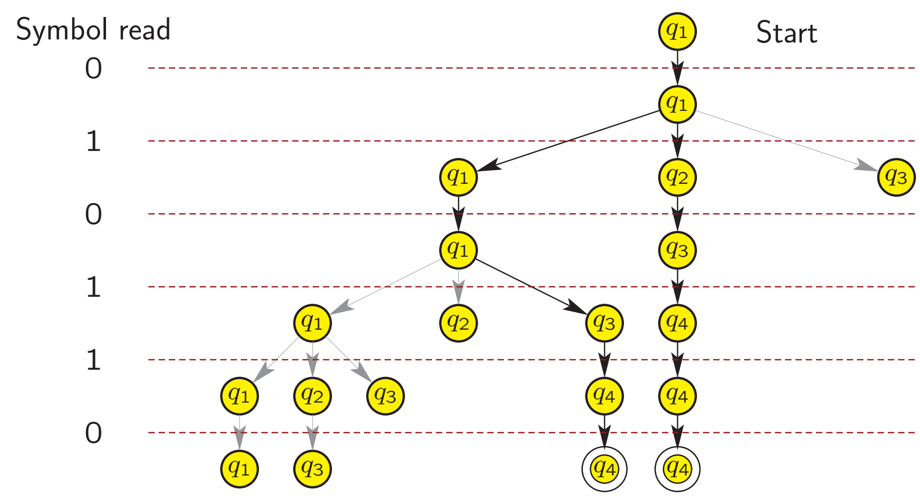
- Suppose NFA is in a state with multiple ways to proceed, e.g., in state q_1 and the next symbol in input string is 1.
- The machine splits into multiple copies of itself (threads).
  - Each copy proceeds with computation independently of others.
  - NFA may be in a **set of states**, instead of a single state.
  - NFA follows all possible computation paths in parallel.
  - If a copy is in a state and next input symbol doesn't appear on any outgoing edge from the state, then the copy **dies** or **crashes**.
- If **any** copy ends in an accept state after reading entire input string, the NFA **accepts** the string.
- If **no** copy ends in an accept state after reading entire input string, then NFA does not accept (**rejects**) the string.



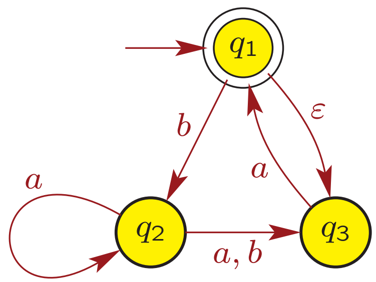
- Similarly, if a state with an ε-transition is encountered,
  - without reading an input symbol, NFA splits into multiple copies, each one following an exiting ε-transition (or staying put).
  - Each copy proceeds independently of other copies.
  - NFA follows all possible paths in parallel.
  - NFA proceeds **nondeterministically** as before.

- What happens on input string 010110 ?

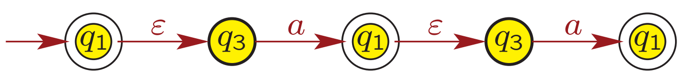




Example: NFA N



- N accepts strings ε, a, aa, baa, $baba$, $\ldots$.
  - e.g., $aa = \varepsilon a \varepsilon a$



- N does not accept (i.e., rejects) strings b, ba, bb, bbb, $\ldots$.

Formal Definition of NFA

Definition: For an alphabet Σ, define $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

- Σ_ε is set of possible labels on NFA edges.

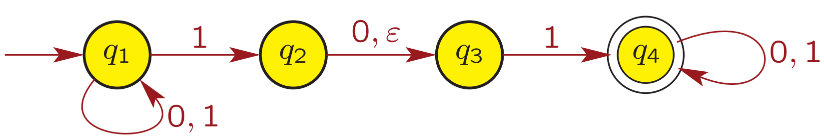
Definition: A **nondeterministic finite automaton (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states
2. Σ is an alphabet
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function, where
   - $\mathcal{P}(Q)$ is the power set of Q
   - δ defines label on each edge.
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states.

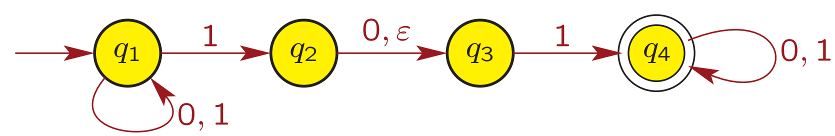
Difference Between DFA and NFA

- DFA has transition function $\delta : Q \times \Sigma \to Q$.

- NFA has transition function $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$.
  - Returns a **set of states** rather than a single state.
  - Allows for ε-transitions because $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
  - For state $q \in Q$ and $\ell \in \Sigma_\varepsilon$, $\delta(q, \ell)$ is set of states where edges out of q labeled with ℓ lead to.



- **Remark:** Note that every DFA is also an NFA.



Formal description of above NFA $N = (Q, \Sigma, \delta, q_1, F)$

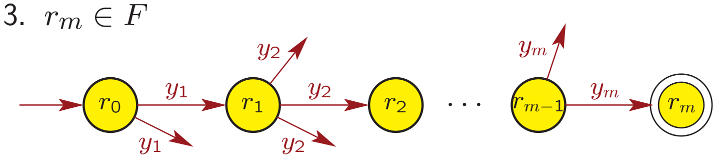
- $Q = \{q_1, q_2, q_3, q_4\}$ is the set of states
- $\Sigma = \{0, 1\}$ is the alphabet
- Transition function $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

- q_1 is the start state
- $F = \{q_4\}$ is the set of accept states

Formal Definition of NFA Computation

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w \in \Sigma^*$.
- Then N **accepts** w if
  - we can write w as $w = y_1 y_2 \cdots y_m$ for some $m \geq 0$, where each $y_i \in \Sigma_\varepsilon$, and
  - there is a sequence of states $r_0, r_1, r_2, \ldots, r_m$ in Q such that
    1. $r_0 = q_0$
    2. $r_{i+1} \in \delta(r_i, y_{i+1})$ for each $i = 0, 1, 2, \ldots, m-1$
    3. $r_m \in F$



Definition: The set of all input strings that are accepted by NFA N is the **language recognized by** N and is denoted by $L(N)$.

Equivalence of DFAs and NFAs

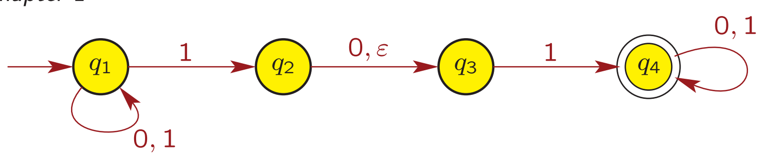
Definition: Two machines (of any types) are **equivalent** if they recognize the same language.

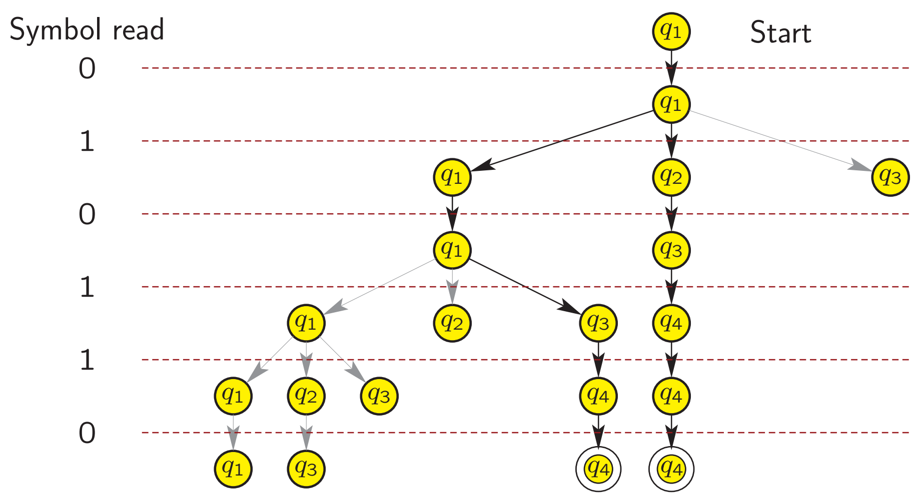
Theorem 1.39
Every NFA N has an equivalent DFA M.

- i.e., if N is some NFA, then $\exists$ DFA M such that $L(M) = L(N)$.

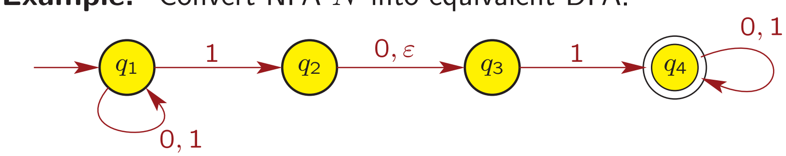
Proof Idea:

- NFA N splits into multiple copies of itself on nondeterministic moves.
- NFA can be in a **set of states** at any one time.
- Build DFA M whose set of states is the **power set** of the set of states of NFA N, keeping track of where N can be at any time.

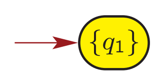




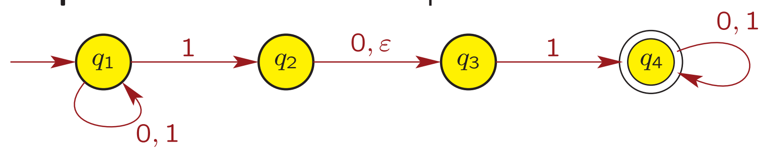
Example: Convert NFA N into equivalent DFA.



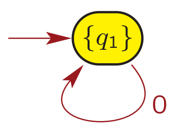
N's start state q_1 has no ε-edges out, so DFA has start state $\{q_1\}$.



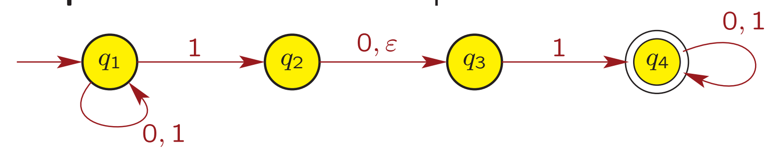
Example: Convert NFA N into equivalent DFA.



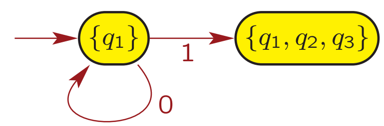
On reading 0 from states in $\{q_1\}$, can reach states $\{q_1\}$.



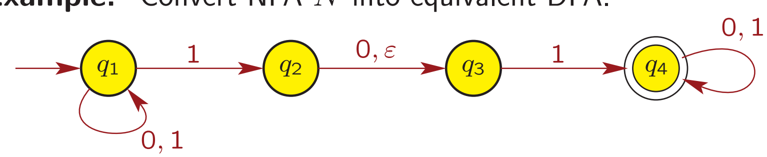
Example: Convert NFA N into equivalent DFA.



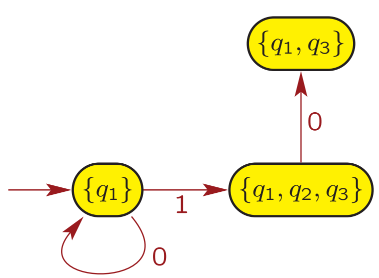
On reading 1 from states in $\{q_1\}$, can reach states $\{q_1, q_2, q_3\}$.



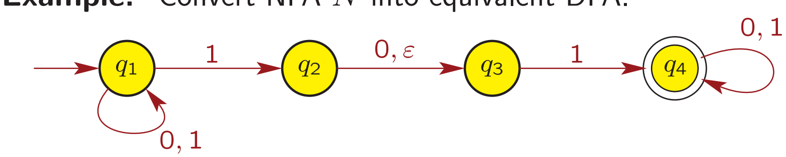
Example: Convert NFA N into equivalent DFA.



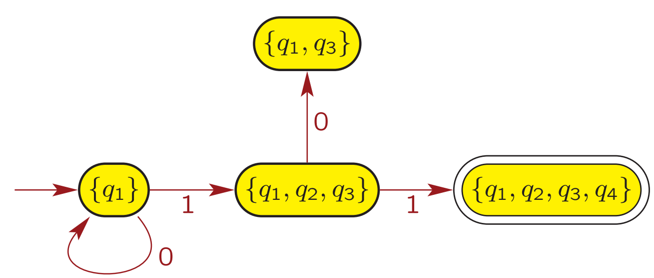
On reading 0 from states in $\{q_1, q_2, q_3\}$, can reach states $\{q_1, q_3\}$.



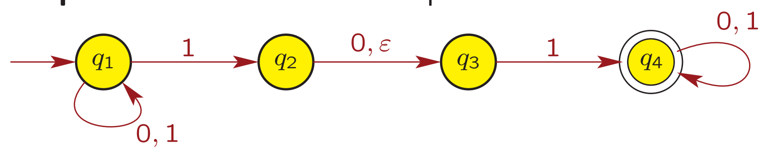
Example: Convert NFA N into equivalent DFA.



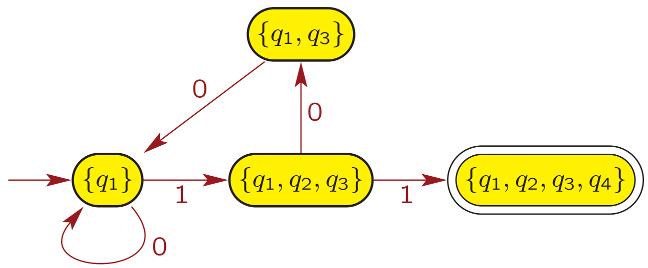
On reading 1 from states in $\{q_1, q_2, q_3\}$, can reach $\{q_1, q_2, q_3, q_4\}$.



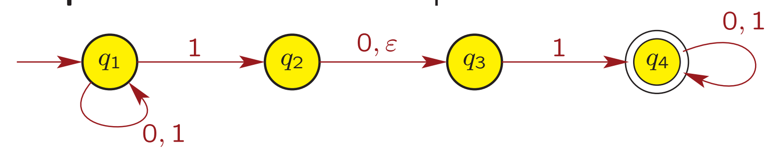
Example: Convert NFA N into equivalent DFA.



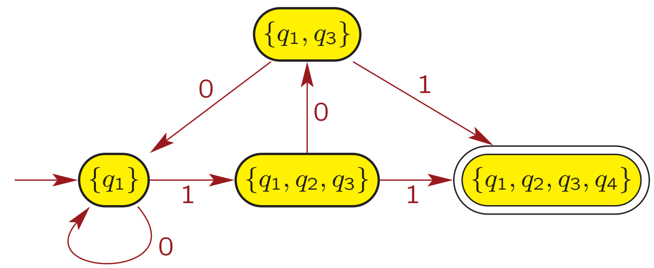
On reading 0 from states in $\{q_1, q_3\}$, can reach states $\{q_1\}$.



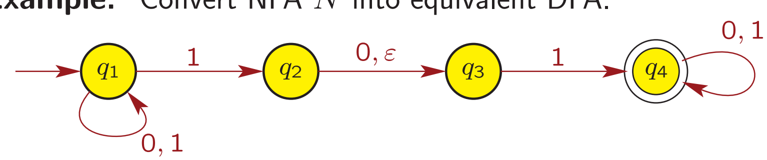
Example: Convert NFA N into equivalent DFA.



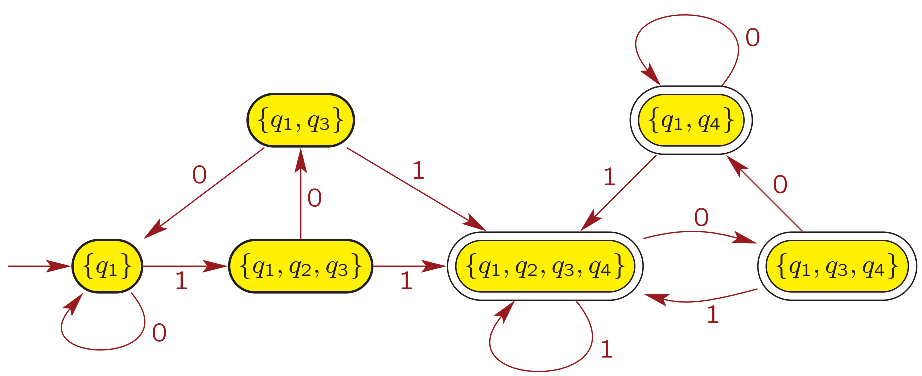
On reading 1 from states in $\{q_1, q_3\}$, can reach states $\{q_1, q_2, q_3, q_4\}$.



Example: Convert NFA N into equivalent DFA.

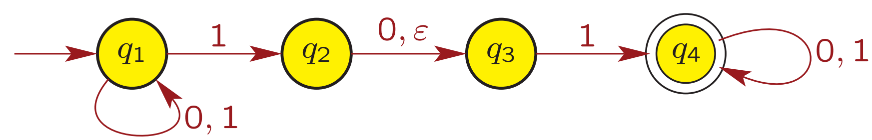


Continue until each DFA state has a 0-edge and a 1-edge leaving it.
DFA accept states have ≥ 1 accept states from N.



Proof. (Theorem 1.39)

- Consider NFA $N = (Q, \Sigma, \delta, q_0, F)$:



- **Definition:** The ε-**closure** of a set of states $R \subseteq Q$ is

$$E(R) \;=\; \{\, q \mid q \text{ can be reached from } R \text{ by travelling over 0 or more } \varepsilon \text{ transitions} \,\}.$$

  - e.g., $E(\{q_1, q_2\}) = \{q_1, q_2, q_3\}$.

Convert NFA to Equivalent DFA

Given NFA $N = (Q, \Sigma, \delta, q_0, F)$, build an equivalent DFA $M = (Q', \Sigma, \delta', q'_0, F')$ as follows:

1. Calculate the ε-closure of every subset $R \subseteq Q$.
2. Define DFA M's set of states $Q' = \mathcal{P}(Q)$.
3. Define DFA M's start state $q'_0 = E(\{q_0\})$.
4. Define DFA M's set of accept states F' to be all DFA states in Q' that include an accept state of NFA N; i.e.,
$$F' = \{\, R \in Q' \mid R \cap F \neq \emptyset \,\}.$$
5. Calculate DFA M's transition function $\delta' : Q' \times \Sigma \to Q'$ as
$$\delta'(R, \ell) = \{\, q \in Q \mid q \in E(\delta(r, \ell)) \text{ for some } r \in R \,\}$$
for $R \in Q' = \mathcal{P}(Q)$ and $\ell \in \Sigma$.
6. Can leave out any state $q' \in Q'$ not reachable from q'_0, e.g., $\{q_2, q_3\}$ in our previous example.

Regular $\Longleftrightarrow$ NFA

Corollary 1.40
Language A is regular if and only if some NFA recognizes A.

Proof.
($\Rightarrow$)

- If A is regular, then there is a DFA for it.
- But every DFA is also an NFA, so there is an NFA for A.

($\Leftarrow$)

- Follows from previous theorem (1.39), which showed that every NFA has an equivalent DFA.

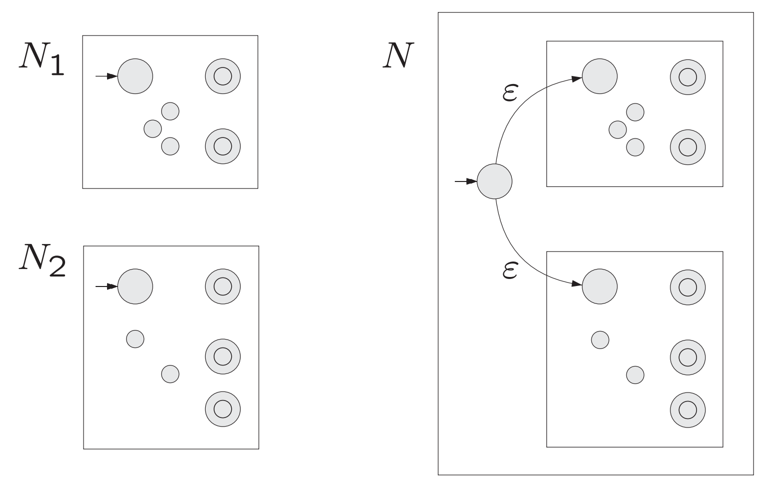
Class of Regular Languages Closed Under Union

Remark: Can use fact that every NFA has an equivalent DFA to simplify the proof that the class of regular languages is closed under union.

Remark: Recall **union**:
$$A_1 \cup A_2 = \{\, w \mid w \in A_1 \text{ or } w \in A_2 \,\}.$$

Theorem 1.45
The class of regular languages is closed under union.

Proof Idea: Given NFAs N_1 and N_2 for A_1 and A_2, resp., construct NFA N for $A_1 \cup A_2$ as follows:



Construct NFA for $A_1 \cup A_2$ from NFAs for A_1 and A_2

- Let A_1 be language recognized by NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.
- Let A_2 be language recognized by NFA $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
- Construct NFA $N = (Q, \Sigma, \delta, q_0, F)$ for $A_1 \cup A_2$:
  - $Q = \{q_0\} \cup Q_1 \cup Q_2$ is set of states of N.
  - q_0 is start state of N.
  - Set of accept states $F = F_1 \cup F_2$.
  - For $q \in Q$ and $a \in \Sigma_\varepsilon$, transition function δ satisfies

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1, \\ \delta_2(q,a) & \text{if } q \in Q_2, \\ \{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \varepsilon, \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$

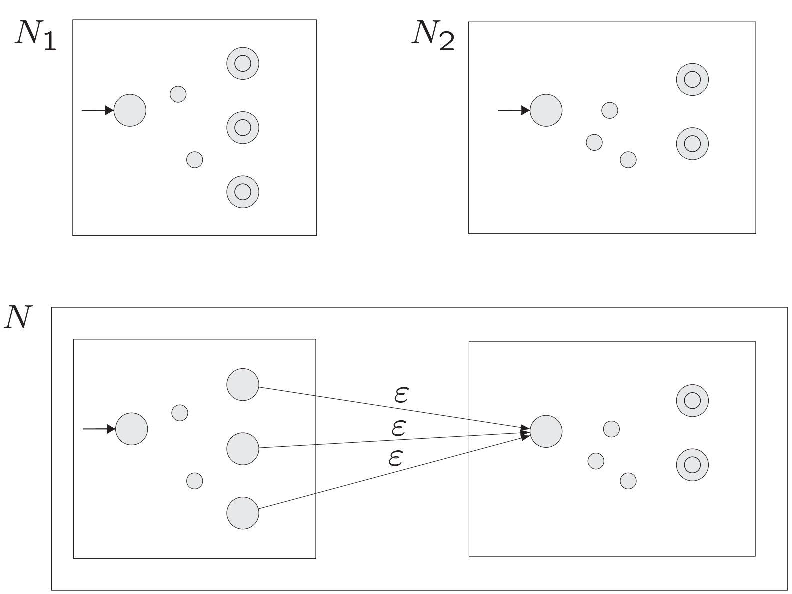
Class of Regular Languages Closed Under Concatenation

Remark: Recall **concatenation**:

$$A \circ B = \{\, vw \mid v \in A,\ w \in B \,\}.$$

Theorem 1.47
The class of regular languages is closed under concatenation.

Proof Idea: Given NFAs N_1 and N_2 for A_1 and A_2, resp., construct NFA N for $A_1 \circ A_2$ as follows:



Construct NFA for $A_1 \circ A_2$ from NFAs for A_1 and A_2

- Let A_1 be language recognized by NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.
- Let A_2 be language recognized by NFA $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
- Construct NFA $N = (Q, \Sigma, \delta, q_1, F_2)$ for $A_1 \circ A_2$:
  - $Q = Q_1 \cup Q_2$ is set of states of N.
  - Start state of N is q_1, which is start state of N_1.
  - Set of accept states of N is F_2, which is same as for N_2.
  - For $q \in Q$ and $a \in \Sigma_\varepsilon$, transition function δ satisfies

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 - F_1, \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \varepsilon, \\ \delta_1(q,a) \cup \{q_2\} & \text{if } q \in F_1 \text{ and } a = \varepsilon, \\ \delta_2(q,a) & \text{if } q \in Q_2. \end{cases}$$

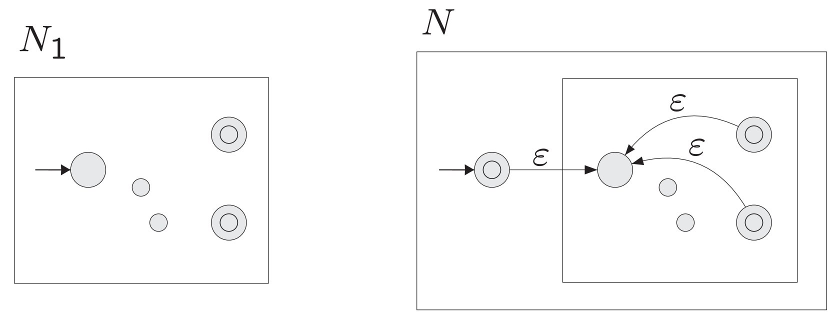
Class of Regular Languages Closed Under Star

Remark: Recall **Kleene star**:

$$A^* = \{\, x_1 x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A \,\}.$$

Theorem 1.49
The class of regular languages is closed under the Kleene-star operation.

Proof Idea: Given NFA N_1 for A,
construct NFA N for A^* as follows:



Construct NFA for A^* from NFA for A

- Let A be language recognized by NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.
- Construct NFA $N = (Q, \Sigma, \delta, q_0, F)$ for A^* :
  - $Q = \{q_0\} \cup Q_1$ is set of states of N.
  - q_0 is start state of N.
  - $F = \{q_0\} \cup F_1$ is the set of accept states of N.
  - For $q \in Q$ and $a \in \Sigma_\varepsilon$, transition function δ satisfies

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 - F_1, \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \varepsilon, \\ \delta_1(q,a) \cup \{q_1\} & \text{if } q \in F_1 \text{ and } a = \varepsilon, \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \varepsilon, \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$

Regular Expressions

- Regular expressions are a way of describing certain languages.
- Consider alphabet $\Sigma = \{0, 1\}$.
- Shorthand notation:
  - 0 means $\{0\}$
  - 1 means $\{1\}$
- Regular expressions use above shorthand notation and operations
  - union $\cup$
  - concatenation $\circ$
  - Kleene star $*$
- When using concatenation, will often leave out operator "$\circ$".

Interpreting Regular Expressions

Example: $0 \cup 1$ means $\{0\} \cup \{1\}$, which equals $\{0, 1\}$.

Example:

- Consider $(0 \cup 1)0^*$, which means $(0 \cup 1) \circ 0^*$.
- This equals $\{0, 1\} \circ \{0\}^*$.
- Recall $\{0\}^* = \{\, \varepsilon, 0, 00, 000, \ldots \}$.
- Thus, $\{0, 1\} \circ \{0\}^*$ is the set of strings that
  - start with symbol 0 or 1, and
  - followed by zero or more 0's.

Another Example of a Regular Expression

Example:

- $(0 \cup 1)^*$ means $(\{0\} \cup \{1\})^*$.
- This equals $\{0, 1\}^*$, which is the set of all possible strings over the alphabet $\Sigma = \{0, 1\}$.
- When $\Sigma = \{0, 1\}$, often use shorthand notation Σ to denote regular expression $(0 \cup 1)$.

Hierarchy of Operations in Regular Expressions

- In most programming languages,
  - multiplication has precedence over addition

$$2 + 3 \times 4 = 14$$

  - parentheses change usual order

$$(2 + 3) \times 4 = 20$$

  - exponentiation has precedence over multiplication and addition

$$4 + 2 \times 3^2 = \underline{\qquad}, \qquad 4 + (2 \times 3)^2 = \underline{\qquad}.$$

- Order of precedence for the regular operations:
  1. Kleene star
  2. concatenation
  3. union
- Parentheses change usual order.

More Examples of Regular Expressions

Example: $00 \cup 101^*$ is language consisting of

- string 00
- strings that begin with 10 and followed by zero or more 1's.

Example: $0(0 \cup 101)^*$ is the language consisting of strings that

- start with 0
- concatenated to a string in $\{0, 101\}^*$.

For example, 0101001010 is in the language because

$$0101001010 = 0 \circ 101 \circ 0 \circ 0 \circ 101 \circ 0.$$

Formal Definition of Regular Expression

Definition: R is a **regular expression** with alphabet Σ if R is

1. a for some $a \in \Sigma$
2. ε
3. $\emptyset$
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
5. $(R_1) \circ (R_2)$, also denoted by $(R_1)(R_2)$, where R_1 and R_2 are regular expressions
6. $(R_1)^*$, where R_1 is a regular expression
7. (R_1), where R_1 is a regular expression.

Can remove redundant parentheses, e.g., $((0) \cup (1))(1) \longrightarrow (0 \cup 1)1$.

Definition: If R is a regular expression, then $L(R)$ is the language **generated** (or **described** or **defined**) by R.

Examples of Regular Expressions

Examples: For $\Sigma = \{0, 1\}$,

1. $(0 \cup 1) \;=\; \{0, 1\}$
2. $0^*10^* \;=\; \{\, w \mid w \text{ has exactly a single } 1 \,\}$
3. $\Sigma^*1\Sigma^* \;=\; \{\, w \mid w \text{ has at least one } 1 \,\}$
4. $\Sigma^*001\Sigma^* \;=\; \{\, w \mid w \text{ contains } 001 \text{ as a substring} \,\}$
5. $(\Sigma\Sigma)^* \;=\; \{\, w \mid |w| \text{ is even} \,\}$
6. $(\Sigma\Sigma\Sigma)^* \;=\; \{\, w \mid |w| \text{ is a multiple of three} \,\}$
7. $0\Sigma^*0 \;\cup\; 1\Sigma^*1 \;\cup\; 0 \;\cup\; 1 \;=$ $\{\, w \mid w \text{ starts and ends with the same symbol} \,\}$
8. $1^*\emptyset \;=\; \emptyset$, anything concatenated with $\emptyset$ is equal to $\emptyset$.
9. $\emptyset^* \;=\; \{\varepsilon\}$

Examples:

1. $R \cup \emptyset \;=\; \emptyset \cup R \;=\; R$
2. $R \circ \varepsilon \;=\; \varepsilon \circ R \;=\; R$
3. $R \circ \emptyset \;=\; \emptyset \circ R \;=\; \emptyset$
4. $R_1(R_2 \cup R_3) \;=\; R_1R_2 \cup R_1R_3$. Concatenation distributes over union.

Example:

- Define EVEN-EVEN over alphabet $\Sigma = \{a, b\}$ as strings with an even number of a's and an even number of b's; see slide 1-20 for a DFA.
- For example, $aababbaaababab \in$ EVEN-EVEN.
- Regular expression:

$$\big(\; aa \;\cup\; bb \;\cup\; (ab \cup ba)(aa \cup bb)^*(ab \cup ba) \;\big)^*$$

Kleene's Theorem

Theorem 1.54
Language A is regular iff A has a regular expression.

Lemma 1.55
If a language is described by a regular expression, then it is regular.

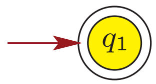
Proof. Procedure to convert regular expression R into NFA N :

1. If $R = a$ for some $a \in \Sigma$, then $L(R) = \{a\}$, which has NFA

$N = (\{q_1, q_2\},\ \Sigma,\ \delta,\ q_1,\ \{q_2\})$ where transition function δ

- $\delta(q_1, a) = \{q_2\}$,
- $\delta(r, b) = \emptyset$ for any state $r \neq q_1$ or any $b \in \Sigma_\varepsilon$ with $b \neq a$.

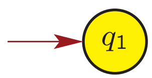
2. If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$, which has NFA



$N = (\{q_1\},\ \Sigma,\ \delta,\ q_1,\ \{q_1\})$ where

- $\delta(r, b) = \emptyset$ for any state r and any $b \in \Sigma_\varepsilon$.

3. If $R = \emptyset$, then $L(R) = \emptyset$, which has NFA

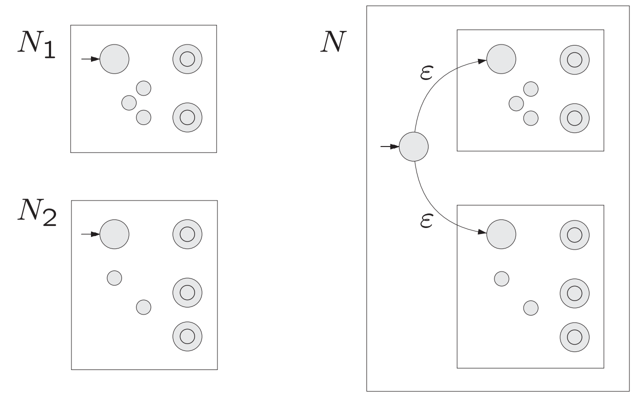


$N = (\{q_1\},\ \Sigma,\ \delta,\ q_1,\ \emptyset)$ where

- $\delta(r, b) = \emptyset$ for any state r and any $b \in \Sigma_\varepsilon$.

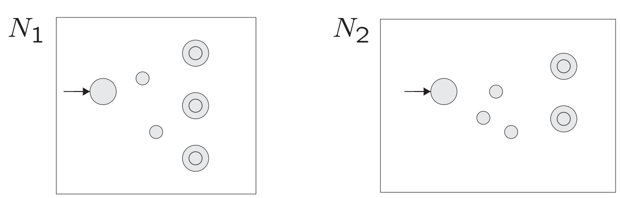
4. If $R = (R_1 \cup R_2)$ and
   - $L(R_1)$ has NFA N_1
   - $L(R_2)$ has NFA N_2,

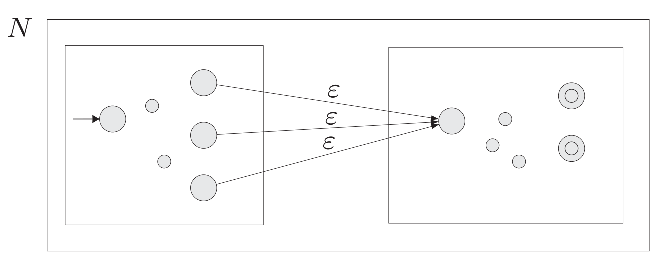
   then $L(R) = L(R_1) \cup L(R_2)$ has NFA N below:



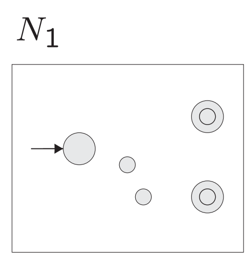
5. If $R = (R_1) \circ (R_2)$ and
   - $L(R_1)$ has NFA N_1
   - $L(R_2)$ has NFA N_2,

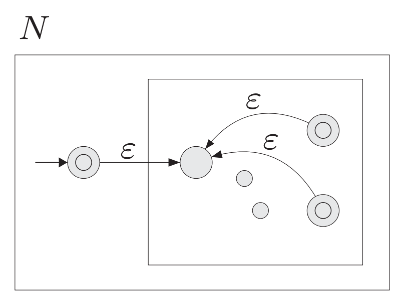
   then $L(R) = L(R_1) \circ L(R_2)$ has NFA N below:





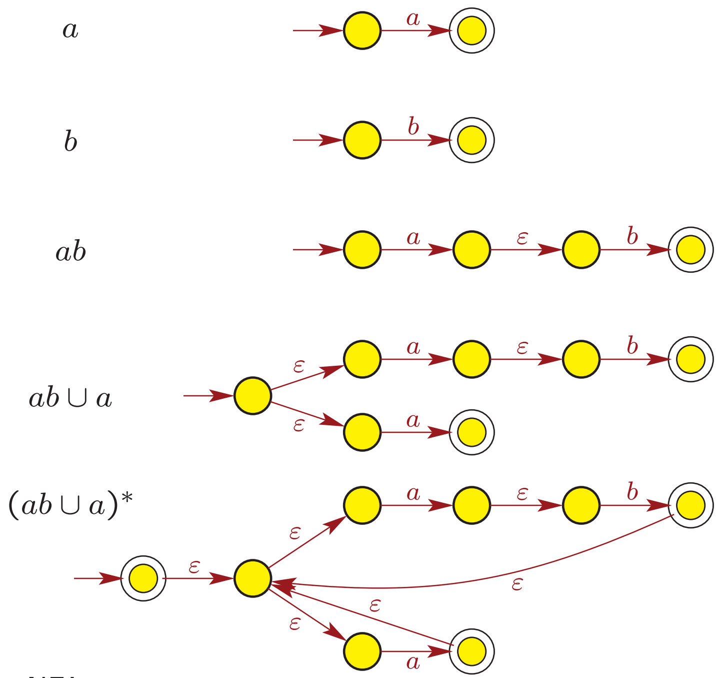
6. If $R = (R_1)^*$ and $L(R_1)$ has NFA N_1,
   then $L(R) = (L(R_1))^*$ has NFA N below:





- Thus, can convert any regular expression R into an NFA.
- Hence, Corollary 1.40 implies that the language $L(R)$ is regular.

Ex: Build NFA for $(ab \cup a)^*$



$\exists$ other correct NFAs

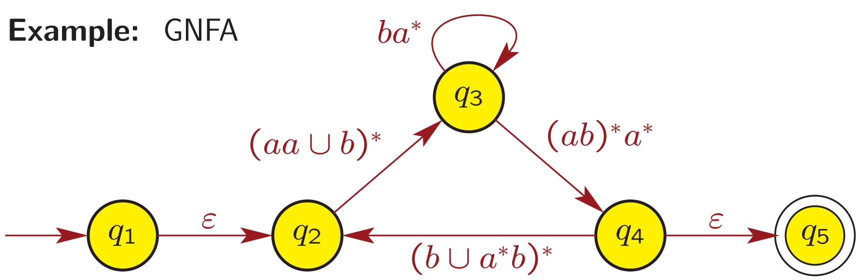
More of Kleene's Theorem

Lemma 1.60
If a language is regular, then it has a regular expression.

Proof Idea:

- Convert DFA into regular expression.
- Use **generalized NFA (GNFA)**, which is an NFA with following modifications:
  - no edges into start state.
  - single accept state, with no edges out of it.
  - labels on edges are **regular expressions** instead of elements from Σ_ε.
    - can traverse edge on any string generated by its regular expression.

Example: GNFA



- Can move from
  - q_1 to q_2 on string ε.
  - q_2 to q_3 on string $aabaa$.
  - q_3 to q_3 on string b or $baaa$.
  - q_3 to q_4 on string ε.
  - q_4 to q_5 on string ε.
- GNFA accepts string $\varepsilon \circ aabaa \circ b \circ baaa \circ \varepsilon \circ \varepsilon = aabaabbaaa$.

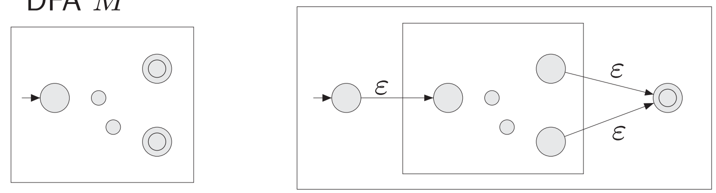
Method to convert DFA into regular expression

1. First convert DFA into equivalent GNFA.
2. Apply following iterative procedure:
   - In each step, eliminate one state from GNFA.
     - When state is eliminated, need to account for every path that was previously possible.
     - Can eliminate states in any order but end result will be different.
     - Never delete start or (unique) accept state.
   - Done when only 2 states remaining: start and accept.
     - Label on remaining arc between start and accept states is a regular expression for language of original DFA.

Remark: Method also can convert NFA into a regular expression.

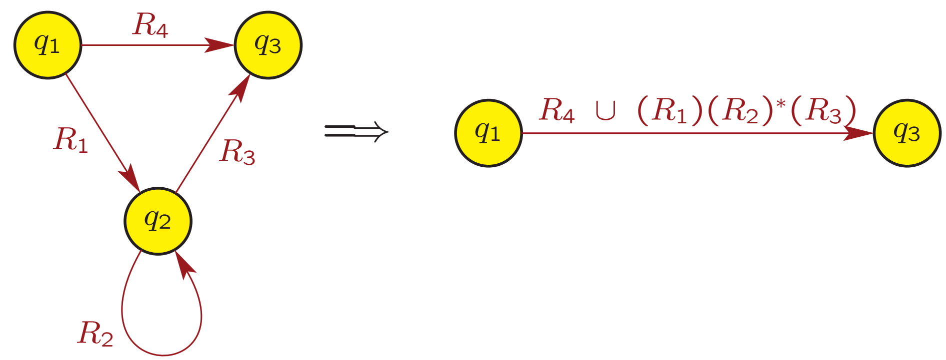
1. Convert DFA $M = (Q, \Sigma, \delta, q_1, F)$ into equivalent GNFA G.
   - Introduce new start state s.
     - Add edge from s to q_1 with label ε.
     - Make q_1 no longer the start state.
   - Introduce new accept state t.
     - Add edge with label ε from each state $q \in F$ to t.
     - Make each state originally in F no longer an accept state.
   - Change edge labels into regular expressions.
     - e.g., "a, b" becomes "$a \cup b$".

DFA M GNFA G

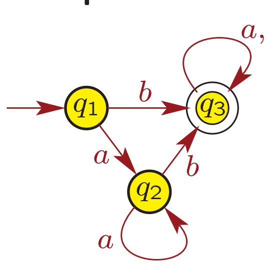


2. Iteratively eliminate a state from GNFA G.
   - Need to take into account all possible previous paths.
   - Never eliminate new start state s or new accept state t.

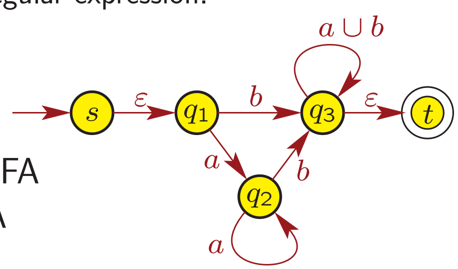
Example: Eliminate state q_2, which has no other in/out edges.



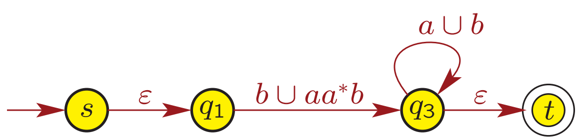
Example: Convert DFA M into regular expression.



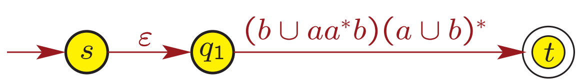
1) Convert DFA into GNFA



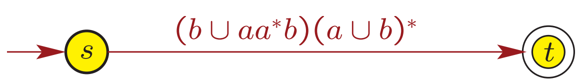
2.1) Eliminate state q_2



2.2) Eliminate state q_3

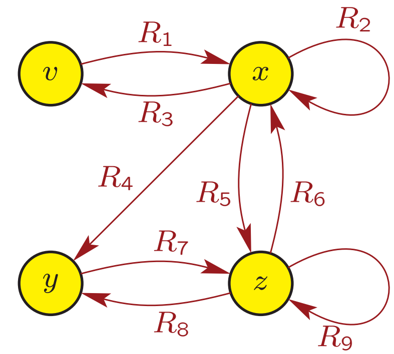


2.3) Eliminate state q_1



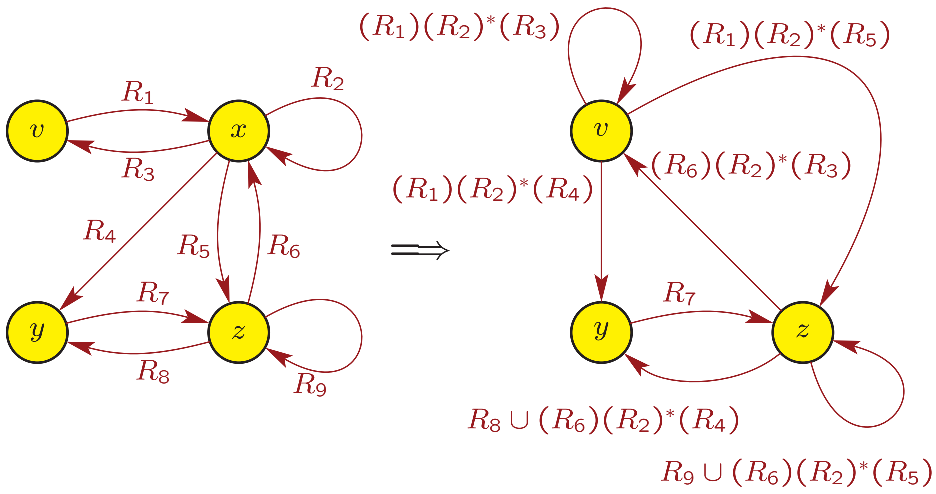
Example:

Eliminate state x, which has no other in/out edges



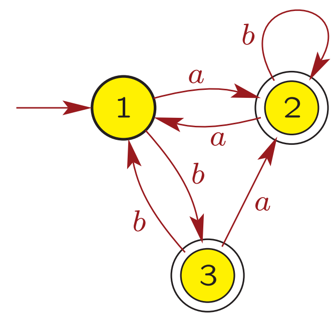
- Let $C = \{v, z\}$, which are states with arcs **into** x (except for x).
- Let $D = \{v, y, z\}$, which are states with arcs **from** x (except for x).
- When we eliminate x, need to account for paths
  - from each state in C directly into x
  - then from x directly to x
  - finally from x directly to each state in D

- Recall $C = \{v, z\}$ and $D = \{v, y, z\}$.
- So eliminating state x gives

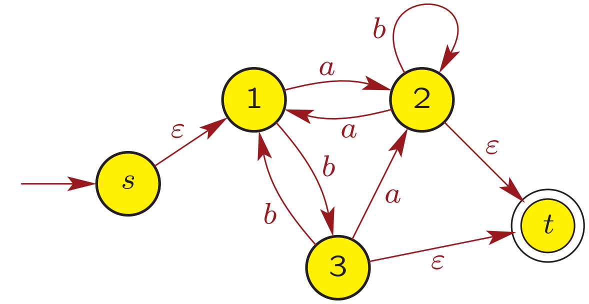


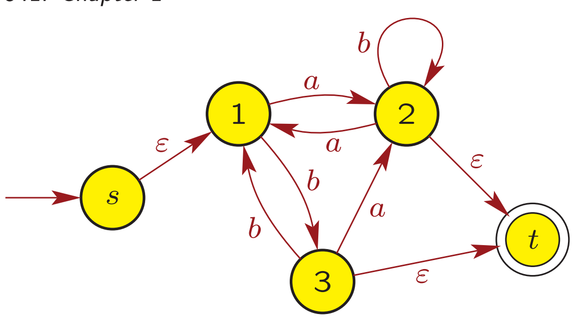
- e.g., for path $v \to x \to y$, add arc from v to y with label $(R_1)(R_2)^*(R_4)$

Example: Convert DFA into Regular Expression



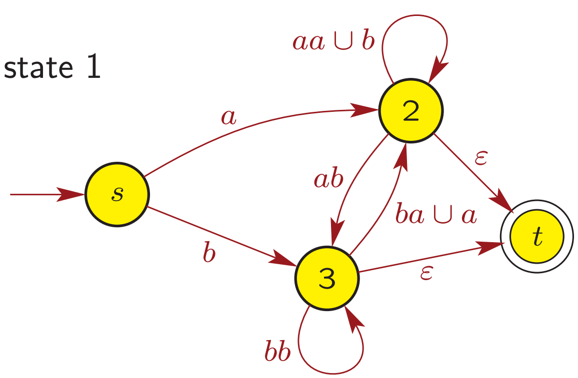
Step 1. Convert DFA into GNFA

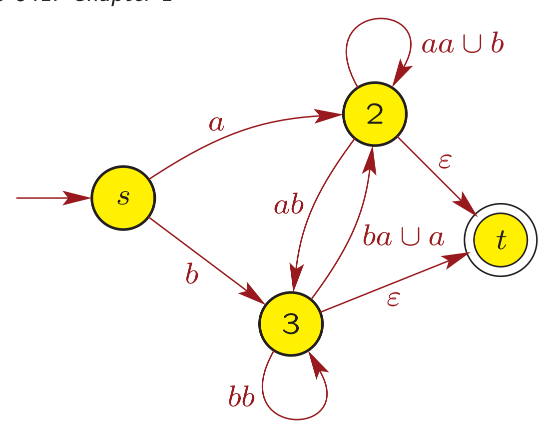




Step 2.1. Eliminate state 1

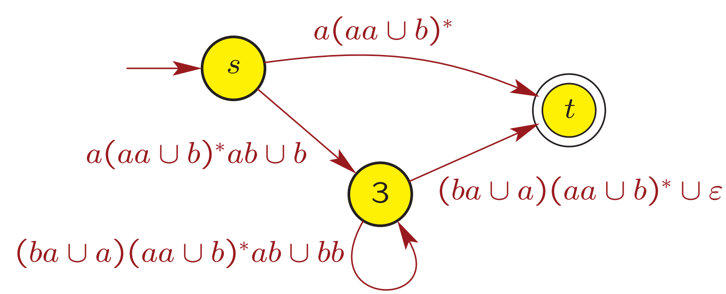
$$C = \{s, 2, 3\}$$
$$D = \{2, 3\}$$

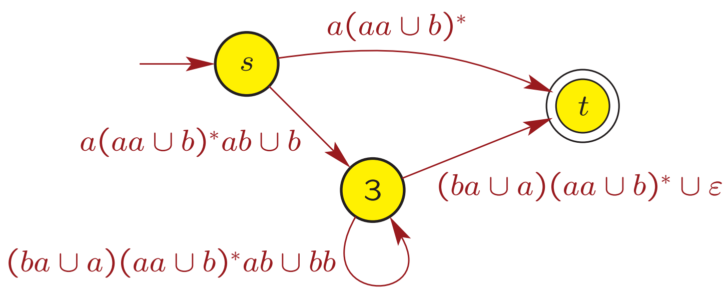




Step 2.2. Eliminate state 2

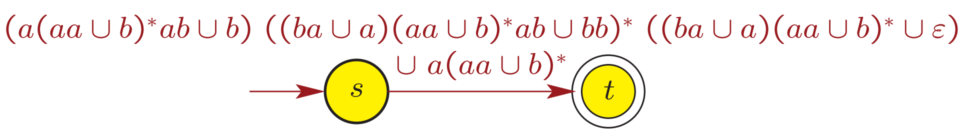
$$C = \{s, 3\}$$
$$D = \{3, t\}$$

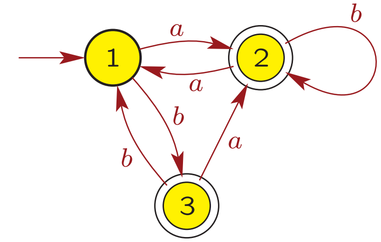




Step 2.3. Eliminate state 3

$$C = \{s\}, \quad D = \{t\}$$





$$\underbrace{\Big(a(aa \cup b)^*ab \cup b\Big)}_{\text{first visit to 3}}\ \underbrace{\Big((ba \cup a)(aa \cup b)^*ab \cup bb\Big)^*}_{\text{0 or more returns to 3}}\ \underbrace{\Big((ba \cup a)(aa \cup b)^* \cup \varepsilon\Big)}_{\text{end in 2 or stay in 3}}$$

$$\cup\ \underbrace{a(aa \cup b)^*}_{\text{ends in 2 with no visits to 3}}$$

- Regular expression accounts for all paths starting in start state 1 and ending in accepting state (2 or 3):
  - visit state 3 at least once (ending in 2 or 3), **or**
  - never visit state 3 (ending in 2).

Finite Languages are Regular

Theorem
If A is a finite language, then A is regular.

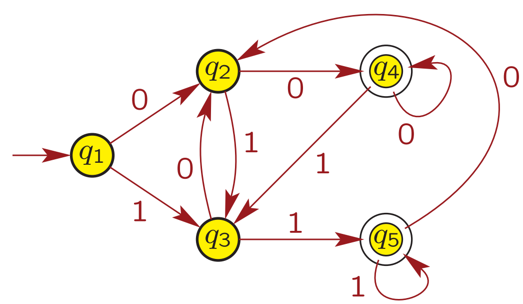
Proof.

- Because A finite, we can write
$$A = \{\, w_1, w_2, \ldots, w_n \,\}$$
for some $n < \infty$.
- A regular expression for A is then
$$R = w_1 \cup w_2 \cup \cdots \cup w_n$$
- Kleene's Theorem then implies A has a DFA, so A is regular.

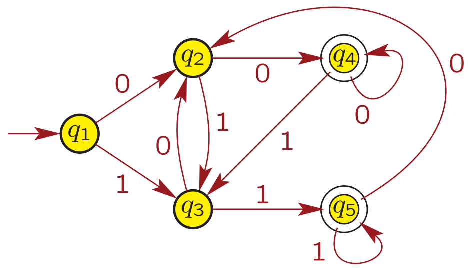
Remark: The converse is **not** true.
e.g., 1^* generates a regular language, but it's infinite.

Pumping Lemma for Regular Languages

Example: DFA with alphabet $\Sigma = \{0, 1\}$ for language A.



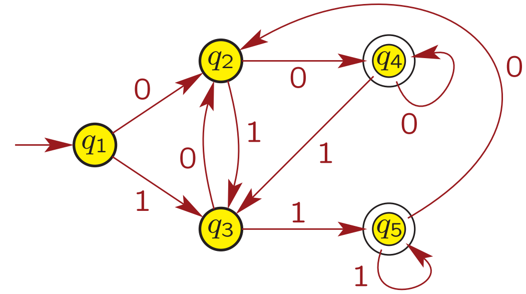
- DFA has 5 states.
- DFA accepts string $s = 0011$, which has length 4.
- On $s = 0011$, DFA visits all of the states.



- For any string s with $|s| \geq 5$, guaranteed to visit some state twice by the **pigeonhole principle**.
- String $s = 0011011$ is accepted by DFA, i.e., $s \in A$.

$$\to q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_4 \xrightarrow{1} q_3 \xrightarrow{1} q_5 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_5$$

- q_2 is first state visited twice.
- Using q_2, divide string s into 3 parts x, y, z such that $s = xyz$.
  - $x = 0$, the symbols read until first visit to q_2.
  - $y = 0110$, the symbols read from first to second visit to q_2.
  - $z = 11$, the symbols read after second visit to q_2.



- Recall DFA accepts string

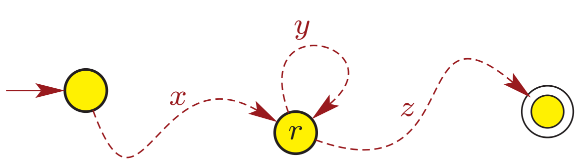
$$s = \underbrace{0}_{x}\underbrace{0110}_{y}\underbrace{11}_{z}.$$

- DFA also accepts strings

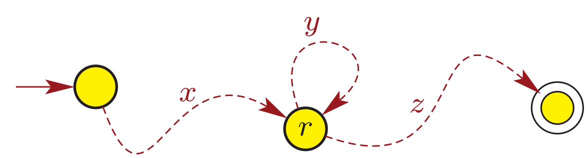
$$xyyz = \underbrace{0}_{x}\underbrace{0110}_{y}\underbrace{0110}_{y}\underbrace{11}_{z},$$
$$xyyyz = \underbrace{0}_{x}\underbrace{0110}_{y}\underbrace{0110}_{y}\underbrace{0110}_{y}\underbrace{11}_{z},$$
$$xz = \underbrace{0}_{x}\underbrace{11}_{z}.$$

- String $xy^iz \in A$ for each $i \geq 0$.

- More generally, consider
  - language A with DFA M having p states,
  - string $s \in A$ with $|s| \geq p$.
- When processing s on M, guaranteed to visit some state twice.
- Let r be first state visited twice.
- Using state r, can divide s as $s = xyz$.
  - x are symbols read until first visit to r.
  - y are symbols read from first to second visit to r.
  - z are symbols read from second visit to r to end of s.



Pumping y



- Because y corresponds to starting in r and returning to r,

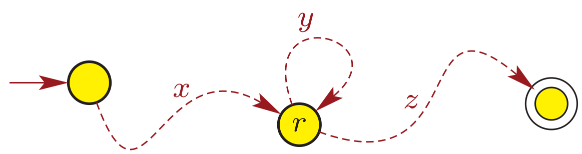
$$xy^iz \in A \text{ for each } i \geq 1.$$

- Also, note $xy^0z = xz \in A$, so

$$xy^iz \in A \text{ for each } i \geq 0.$$

- $|y| > 0$ because
  - y corresponds to starting in r and coming back;
  - this consumes at least one symbol (because DFA), so y can't be empty.

Length of xy



- $|xy| \leq p$, where p is number of states in DFA, because
  - xy are symbols read up to second visit to r.
  - Because r is the first state visited twice, all states visited before second visit to r are unique.
  - So just before visiting r for second time, DFA visited at most p states, which corresponds to reading at most $p - 1$ symbols.
  - The second visit to r, which is after reading 1 more symbol, corresponds to reading at most p symbols.

Pumping Lemma

Theorem 1.70
If A is regular language, **then** $\exists$ number p (pumping length) where,
if $s \in A$ with $|s| \geq p$, **then** s can be split into 3 pieces, $s = xyz$, satisfying the properties

1. $xy^iz \in A$ for each $i \geq 0$,
2. $|y| > 0$, **and**
3. $|xy| \leq p$.

Remarks:

- y^i denotes i copies of y concatenated together, and $y^0 = \varepsilon$.
- $|y| > 0$ means $y \neq \varepsilon$.
- $|xy| \leq p$ means x and y together have no more than p symbols total.

Understanding the Pumping Lemma

If $\overbrace{A \text{ is regular language}}^{M_1}$, **then** $\overbrace{\exists \text{ number } p \text{ (pumping length)}}^{M_2}$ where,

if $\overbrace{s \in A \text{ with } |s| \geq p}^{M_3}$, **then**

$$\left.\begin{array}{l} s \text{ can be split into 3 pieces, } s = xyz, \text{ satisfying properties} \\ \quad 1.\ xy^iz \in A \text{ for each } i \geq 0, \\ \quad 2.\ |y| > 0, \textbf{ and} \\ \quad 3.\ |xy| \leq p. \end{array}\right\} M_4$$

---

if (M_1 is true), **then**
  M_2 is true
  if (M_3 is true), **then**
    M_4 is true
  endif
endif

Nonregular Languages

Definition: Language is **nonregular** if there is no DFA for it.

Remarks:

- Pumping Lemma (PL) is a result about regular languages.
- But PL mainly used to prove that certain language A is **nonregular**.
- Typically done using **proof by contradiction**.
  - Assume language A is regular.
  - PL says that all strings $s \in A$ that are at least a certain length must satisfy some properties
  - By appropriately choosing $s \in A$, will eventually get contradiction.
  - PL: **can** split s into $s = xyz$ satisfying **all** of Properties 1–3.
  - To get contradiction, show **cannot** split $s = xyz$ satisfying 1–3.
    - Show **all** splits satisfying 2–3 violate Property 1.
  - Because Property 3 of PL states $|xy| \leq p$, often choose $s \in A$ so that all of its first p symbols are the same.

Language $A = \{\, 0^n 1^n \mid n \geq 0 \,\}$ is Nonregular

Proof.

- Suppose A is regular, so PL implies A has "pumping length" p.
- Consider string $s = 0^p 1^p \in A$.
- $|s| = 2p \geq p$, so Pumping Lemma will hold.
- So **can** split s into 3 pieces $s = xyz$ satisfying properties
  1. $xy^i z \in A$ for each $i \geq 0$,
  2. $|y| > 0$, and
  3. $|xy| \leq p$.
- To get contradiction, must show **cannot** split $s = xyz$ satisfying 1–3.
  - Show **all** splits $s = xyz$ satisfying Properties 2 and 3 will violate 1.
- Because the first p symbols of $s = \underbrace{00\cdots0}_{p}\underbrace{11\cdots1}_{p}$ are all 0's
  - Property 3 implies that x and y consist of only 0's.
  - z will be the rest of the 0's, followed by all p 1's.
- **Key:** y has some 0's, and z contains all the 1's (and maybe some 0's), so pumping y changes # of 0's but not # of 1's.

- So we have

$$\begin{aligned} x &= 0^j \quad \text{for some } j \geq 0, \\ y &= 0^k \quad \text{for some } k \geq 0, \\ z &= 0^m 1^p \quad \text{for some } m \geq 0 \end{aligned}$$

- $s = xyz$ implies

$$0^p 1^p = 0^j 0^k 0^m 1^p = 0^{j+k+m} 1^p,$$

  so $j + k + m = p$.
- Property 2 states that $|y| > 0$, so $k > 0$.
- Property 1 implies $xyyz \in A$, but

$$\begin{aligned} xyyz &= 0^j 0^k 0^k 0^m 1^p \\ &= 0^{j+k+k+m} 1^p \\ &= 0^{p+k} 1^p \notin A \end{aligned}$$

  because $j + k + m = p$ and $k > 0$.
- **Contradiction**, so $A = \{\, 0^n 1^n \mid n \geq 0 \,\}$ is nonregular.

Language $B = \{\, ww \mid w \in \{0,1\}^* \,\}$ is Nonregular

Proof.

- Suppose B is regular, so PL implies B has "pumping length" p.
- Consider string $s = 0^p 1 0^p 1 \in B$. ($0^p 0^p \in B$ won't work. Why?)
- $|s| = 2p + 2 \geq p$, so Pumping Lemma will hold.
- So **can** split s into 3 pieces $s = xyz$ satisfying properties
  1. $xy^i z \in B$ for each $i \geq 0$,
  2. $|y| > 0$, and
  3. $|xy| \leq p$.
- For contradiction, show **cannot** split $s = xyz$ so that 1–3 hold.
  - Show **all** splits $s = xyz$ satisfying Properties 2 and 3 will violate 1.
- Because first p symbols of $s = \underbrace{00\cdots0}_{p}1\underbrace{00\cdots0}_{p}1$ are all 0's,
  - Property 3 implies that x and y consist only of 0's.
  - z will be the rest of first set of 0's, followed by $1\,0^p\,1$.
- **Key:** y has some of first 0's, and z has all of second 0's, so pumping y changes only # of first 0's.

- So we have

$$\begin{aligned} x &= 0^j \quad \text{for some } j \geq 0, \\ y &= 0^k \quad \text{for some } k \geq 0, \\ z &= 0^m 1 0^p 1 \quad \text{for some } m \geq 0 \end{aligned}$$

- $s = xyz$ implies

$$0^p 1 0^p 1 = 0^j 0^k 0^m 1 0^p 1 = 0^{j+k+m} 1 0^p 1,$$

  so $j + k + m = p$.
- Property 2 states that $|y| > 0$, so $k > 0$.
- Property 1 implies $xyyz \in B$, but

$$\begin{aligned} xyyz &= 0^j 0^k 0^k 0^m 1 0^p 1 \\ &= 0^{j+k+k+m} 1 0^p 1 \\ &= 0^{p+k} 1 0^p 1 \notin B \end{aligned}$$

  because $j + k + m = p$ and $k > 0$.
- **Contradiction**, so $B = \{\, ww \mid w \in \{0,1\}^* \,\}$ is nonregular.

Important Steps in Proving Language is Nonregular

Pumping Lemma (PL):
If A is a regular language, **then** $\exists$ number p (pumping length) where,
if $s \in A$ with $|s| \geq p$, **then** s can be split into 3 pieces, $s = xyz$, with

1. $xy^iz \in A$ for each $i \geq 0$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Remarks:

- Must choose **appropriate** string $s \in A$ to get contradiction.
  - Some strings $s \in A$ might not lead to contradiction.
- Because Property 3 of PL states $|xy| \leq p$, often choose $s \in A$ so that all of its first p symbols are the same.
- Once appropriate s is chosen, need to show **every** possible split of $s = xyz$ leads to contradiction.

Pumping Lemma (PL):
If A is a regular language, **then** $\exists$ number p (pumping length) where,
if $s \in A$ with $|s| \geq p$, **then** s can be split into 3 pieces, $s = xyz$, with

1. $xy^iz \in A$ for each $i \geq 0$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Examples:

1. Let $C = \{ w \in \{a,b\}^* \mid w = w^{\mathcal{R}} \}$, where $w^{\mathcal{R}}$ is the reverse of w.
   - To show C is nonregular, can choose $s = a^p\, b\, a^p \in C$.
   - Choosing $s = a^p \in C$ does **not** work. Why?
2. To show $D = \{ a^{2n}\, b^{3n}\, a^n \mid n \geq 0 \}$ is nonregular, can choose $s = a^{2p}\, b^{3p}\, a^p \in D$.
3. Consider language $E = \{ w \in \{a,b\}^* \mid w \text{ has more } a\text{'s than } b\text{'s} \}$. For example, $baaba \in E$.
   - To show E is nonregular, can choose $s = b^p\, a^{p+1} \in E$.

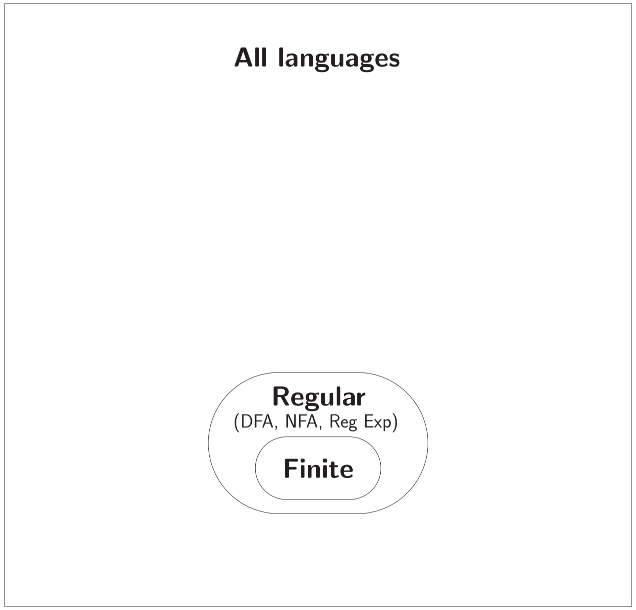
Common Mistake

- Consider $D = \{ a^{2n}\, b^{3n}\, a^n \mid n \geq 0 \}$.
- To show D is nonregular, can choose $s = a^{2p}\, b^{3p}\, a^p \in D$.
- **Common mistake:** try to apply Pumping Lemma with
$$x = a^{2p}, \qquad y = b^{3p}, \qquad z = a^p.$$
- For this split, $|xy| = 5p \not\leq p$.
- But Pumping Lemma states "If D is a regular language, then … **can** split $s = xyz$ satisfying Properties 1–3."
- To get contradiction, need to show **cannot** split $s = xyz$ satisfying Properties 1–3.
  - Need to show **every** split $s = xyz$ doesn't satisfy all of 1–3.
  - Every split $s = xyz$ satisfying Properties 2 and 3 must have
  $$x = a^j, \qquad y = a^k, \qquad z = a^m\, b^{3p}\, a^p,$$
  where $j + k \leq p$, $j + k + m = 2p$, and $k \geq 1$.

$F = \{ w \mid \#$ of 0's in w equals $\#$ of 1's in $w \}$ is Nonregular

- Note that, e.g., $101100 \in F$.
- Need to be careful when choosing string $s \in F$ for Pumping Lemma.
  - If $xyz \in F$ with $y \in F$, then $xy^iz \in F$, so no contradiction.
- **Another Approach:** If F and G are regular, then $F \cap G$ is regular.
- **Solution:** Suppose that F is regular.
  - Let $G = \{ 0^n 1^m \mid n, m \geq 0 \}$.
    - G is regular: it has regular expression 0^*1^*.
  - Then $F \cap G = \{ 0^n 1^n \mid n \geq 0 \}$.
  - But know that $F \cap G$ is not regular.
- **Conclusion:** F is not regular.

Hierarchy of Languages (so far)



Examples

$\{\, 0^n1^n \mid n \geq 0 \,\}$

$(0 \cup 1)^*$

$\{\, 110, 01 \,\}$

Summary of Chapter 1

- DFA is a deterministic machine for recognizing certain languages.
- A language is **regular** if it has a DFA.
- The class of regular languages is closed under union, intersection, concatenation, Kleene-star, complementation.
- NFA can be **nondeterministic**: allows choice in how to process string.
- Every NFA has an equivalent DFA.
- Regular expression is a way of generating certain languages.
- Kleene's Theorem: Language A has DFA iff A has regular expression.
- Every finite language is regular, but not every regular language is finite.
- Use pumping lemma to prove certain languages are not regular.