Chapter 1 - Regular Languages

Q: What is a computer?

- Von Neumann Architecture $\begin{cases} \text{CPU} \\ \text{Memory} \\ \text{I/O} \\ \text{Bus} \end{cases}$

- Real computers are too complicated for a manageable mathematical theory

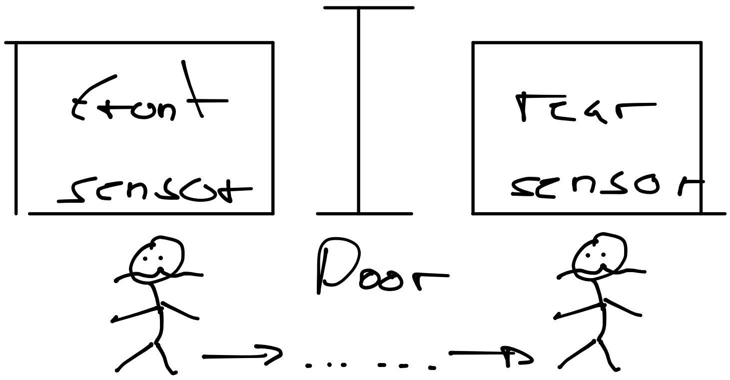
- Instead we will use an idealized computer called a computational model (our first one will be the finite state machine)

1.1 - Finite Automata

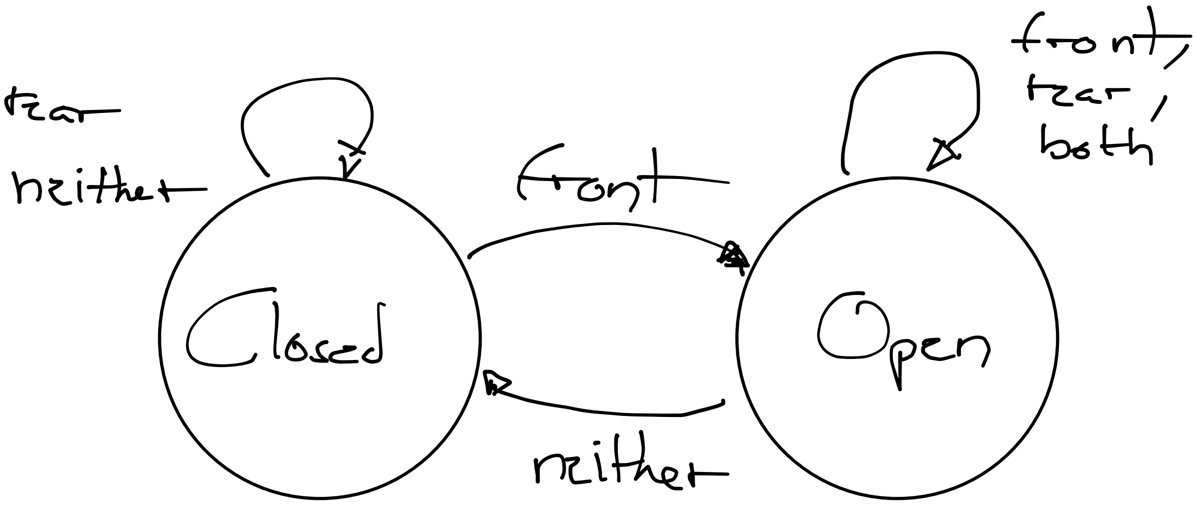
- Computer models with limited amount of memory (but still very useful)

- Lets have a look at an example...

- Automatic Door Example



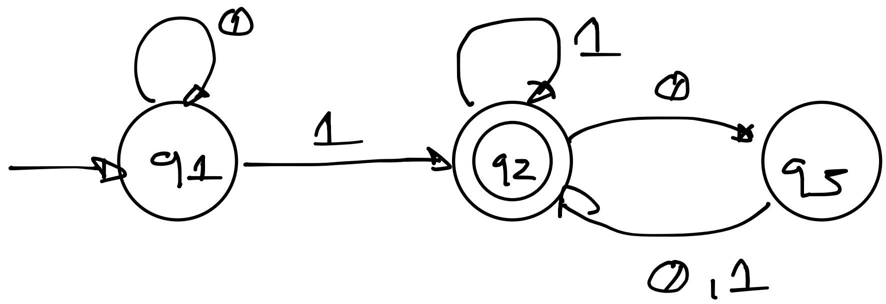
This behaviour can be represented by the following state diagram:



Q: How much memory does this state machine requires?

Only two states, a single bit would suffice (i.e. $\log_2 2 = 1$)

The following figure depicts a finite automaton M_1



- States: q_1, q_2, q_3
  - Start state: q_1
  - Accept state: q_2 (I prefer the term final states)
    (double circle)

- Consider the input string: 1 1 0 1

1. Start in q_1
2. Read 1, transition from q_1 to q_2
3. Read 1, transition from q_2 to q_2
4. Read 0, transition from q_2 to q_3
5. Read 1, transition from q_3 to q_2
6. Accept, because M_1 is in an accept state q_2 at the end of the input.

Formal definition of a finite automaton

> **Q:** So, what is the formal definiton of an automata?

A finite automaton has several parts:

- Set of states (1) and rules (2) for going from one state to another depending on the input symbol;
- An input alphabet (3) that indicates the allowed input symbols;
- It has a start state (4) and a set of final states (5)
- A finite automaton is a list of these 5 objects

Definition 1.5

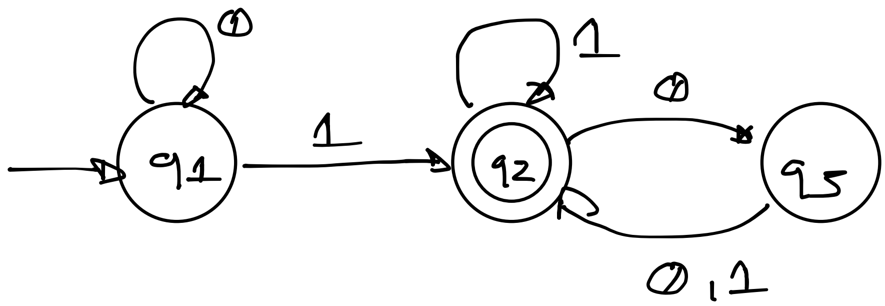
sigma character
delta character

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

1. Q is a finite set called the **states**
2. Σ is a finite set called the **alphabet**
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**
4. $q_0 \in Q$ is the **start state**
5. $F \subseteq Q$ is the set of **accept states**
   (subset)

Example

Lets return to finite automaton M_1:



Q_1: What is the formal definition of M_1?

$\to M_1 = (Q, \Sigma, \delta, q_1, F)$

$\boxed{Q_2:}$ $\boxed{\text{What is the set of states } Q?}$

$\hookrightarrow Q = \{q_1, q_2, q_3\}$

$\boxed{Q_3:}$ $\boxed{\text{What is the alphabet } \Sigma?}$

$\hookrightarrow \Sigma = \{0, 1\}$

$\boxed{Q_4:}$ $\boxed{\text{What is the start state?}}$

$\hookrightarrow q_1$

$\boxed{Q_5:}$ $\boxed{\text{What is the final state?}}$

$\hookrightarrow q_2$

$\boxed{Q_6:}$ $\boxed{\text{What is the transition function?}}$

δ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Q: If a machine recognizes strings that use symbols/characters from an alphabet, what should we name the set of all strings that are recognized by a machine?

If A is the set of all strings that machine M recognizes we say that A is the language of machine A and write $L(M) = A$. M recognizes A

Q7: What is the language recognized by M_1?

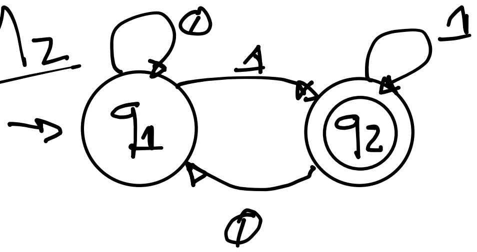
Useful to see examples of recognized strings:

- 1
- 01
- 001
- 100
- 0100
- 00100
- 001001

$L(M_1) = \{w \mid w$ contains at least one 1 and an even number of 0's follow the last 1$\}$

Example 1.7

M_2



Q: What is the definition?

$M_2 = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- $q_0 = q_1$
- $F = q_2$
-

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_2

Q: What is the language recognized?

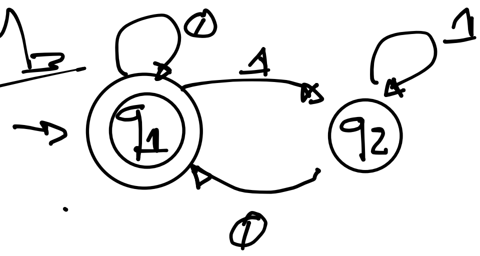
Useful to see some examples of recognized strings

- 1
- 01
- 001
- 0011
- 001101

$L(M_2) = \{w \mid w \text{ ends in } 1\}$

Example 1.9

M_3



$M_3 = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- $q_0 = q_1$
- $F = q_2$

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_2

M_3 is similar to M_3 except for the location of the accept state

Q :	What is the language recognized?

- Useful to see some examples of recognized strings

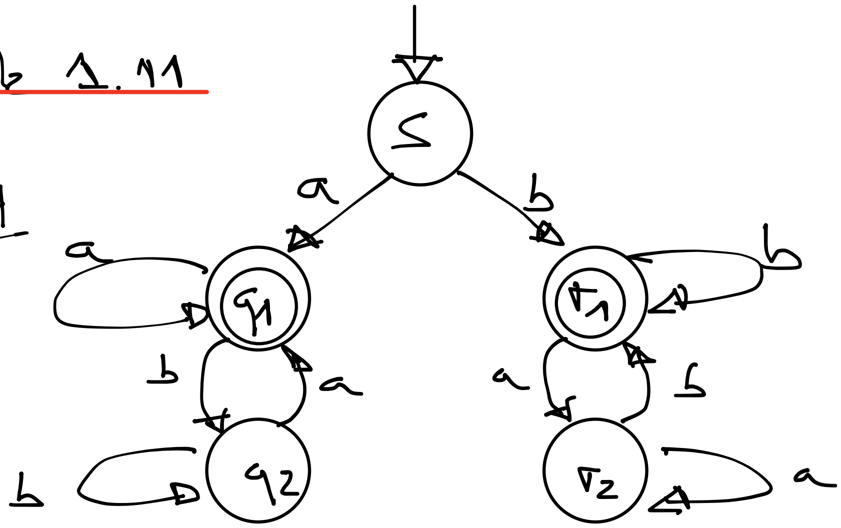
- Because the start state is also a final state M_3 accepts the empty string ε

- ε
- 0
- 010
- 0110
- 01100

$L(M_3) = \{w \mid w$ is the empty string ε or ends in a $0\}$

Example 1.11

M_4



$M_4 = \{ Q, \Sigma, \delta, q_0, F \}$

- $Q = \{ S, q_1, q_2, r_1, r_2 \}$
- $\Sigma = \{ a, b \}$
- $q_0 = S$
- $F = \{ q_1, r_1 \}$
- δ

δ	a	b
S	q_1	r_1
q_1	q_1	q_2
q_2	q_1	q_2
r_1	r_2	r_1
r_2	r_2	r_1

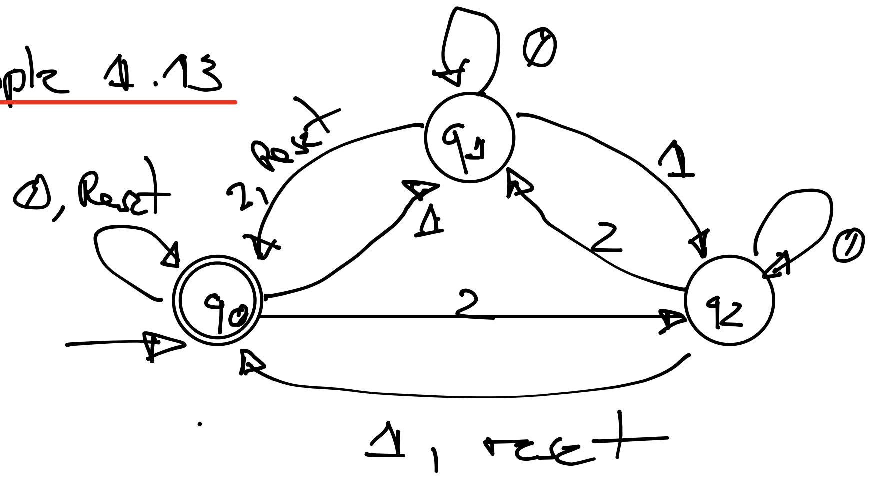
$\boxed{\text{Q: What is the language recognized?}}$

a	b
aba	bab
abba	baab

$L(M_4) = \{ w \mid w$ start and end with the same symbol $\}$

Example 1.13

M_5



$M_5 = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\text{Reset}, 0, 1, 2\}$
- $q_0 = q_0$
- $F = q_0$

δ	Reset	0	1	2
q_0	q_0	q_0	q_1	q_2
q_1	q_0	q_1	q_2	q_0
q_2	q_0	q_2	q_0	q_1

$\boxed{Q: \text{What is the language recognized?}}$

0

12

10101

1002

1000122

Sums the numbers read, if the total is a multiple of 3 it accepts. Every time it receives the RESET symbol it resets the count to zero

Formal Definition of Computation

Now we know:

- Informal definition (state diagram)
- Formal definition (5-tuple)

But we have not described formally the computation procedure

$\boxed{Q:}$ What is the formal definition of computation using finite automata?

Let:

- $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton
- $w = w_1 w_2 \dots w_n$ be a string where $w_i \in \Sigma$

Then:

- M accepts / recognizes w if a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$ (initial state)

2. $\delta(r_i, w_{i+1}) = r_{i+1} \quad \forall i \in [0, n-1]$

3. $r_n \in F$ (final state)

We say that M recognizes language A

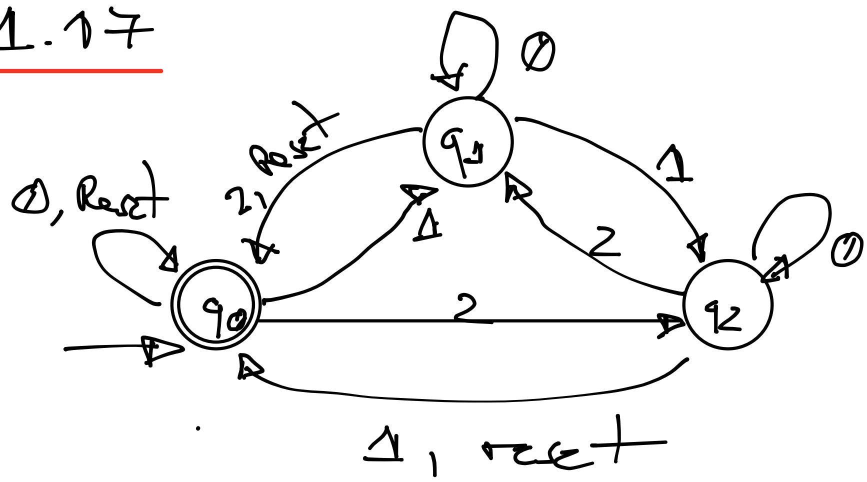
if $A = \{ w \mid M \text{ accepts } w \}$

Definition 1.6

A language is called a regular language if some finite automaton recognizes it

Example 1.17

M_5



- Let $w = 1\ 0\ \text{Reset}\ 2\ 2\ \text{Reset}\ 0\ 1\ 2$

- Sequence of states:

$$\underbrace{q_0}_{\in q_0}, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, \underbrace{q_0}_{\in q_0}$$

which satisfies the three conditions

- $L(M_5) = \{w \mid$ the sum of the symbols in w is 0 modulo 3, except that Reset resets the count to $0\}$

- As M_5 recognizes this language, it is a regular language

Designing Finite Automata

- Designing Automata is a creative process
- No simple design algorithm exists
- Useful: Put yourself in the place of the machine you are trying to design

Example

Design a finite automaton E_1 to recognize the language consisting of all strings over $\Sigma = \{0, 1\}$ with an odd number of 1s.

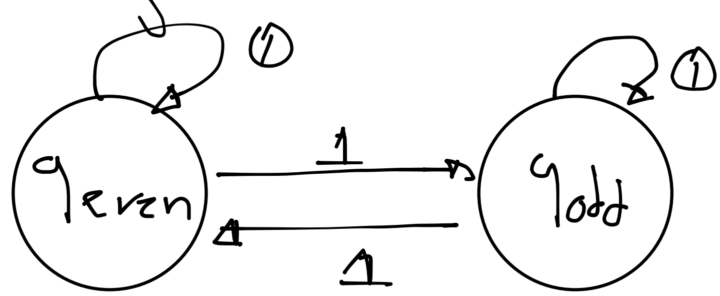
Q_1: How would you go about developing E_1?

Q_2: Do you need to remember the entire string seen?

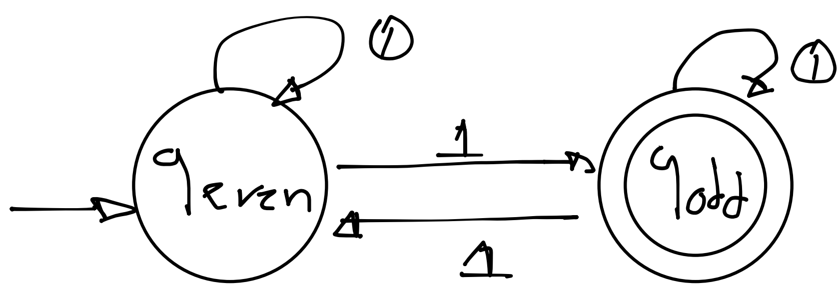
→ No! Simply remember if the machine has seen an odd number of 1's

1) This implies that two states exist:

2) Next, we assign the transition possibilities



3) Next, define the initial and final states:



Example 1.21

- Design a finite automaton E_2 for recognizing the regular of all strings that contain the string 001 as a substring

- Examples: 0010, 1001, 001, 11111111001111

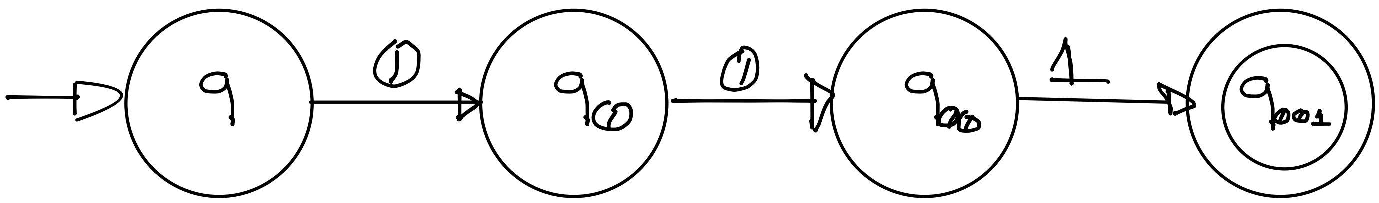
$\boxed{Q_1:}$ How would you go about developing E_2?

- If we see a 0, then we have seen the first of the symbols

- If we see another 0, then we have seen the second of the symbols

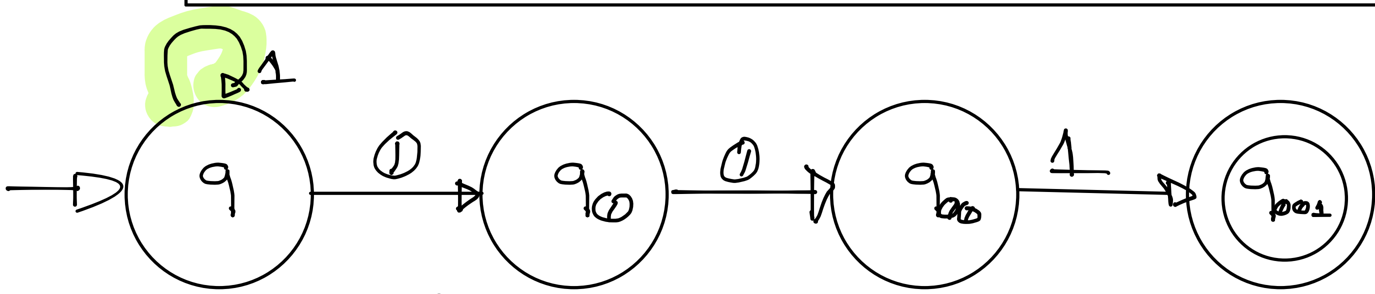
- If we see an 1, then we have seen the third (and final) symbol

1. This implies the following structure:

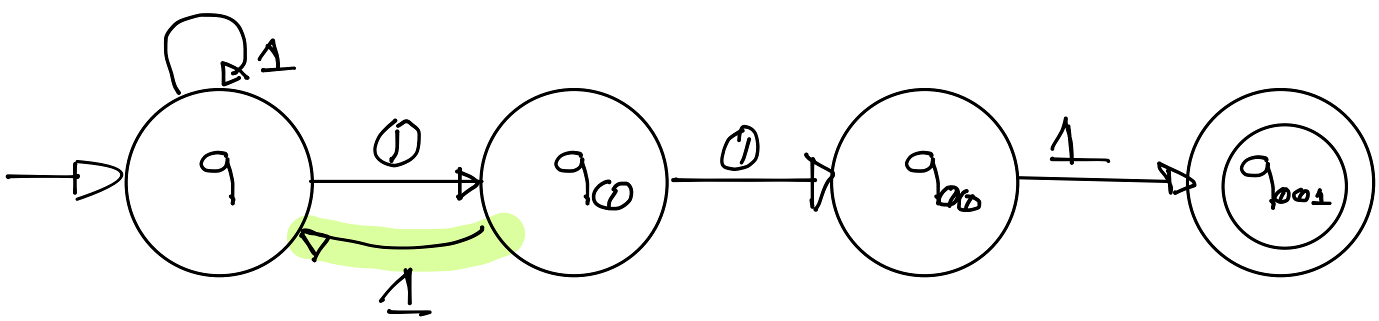


2. Now we need to consider all the remaining possibilities:

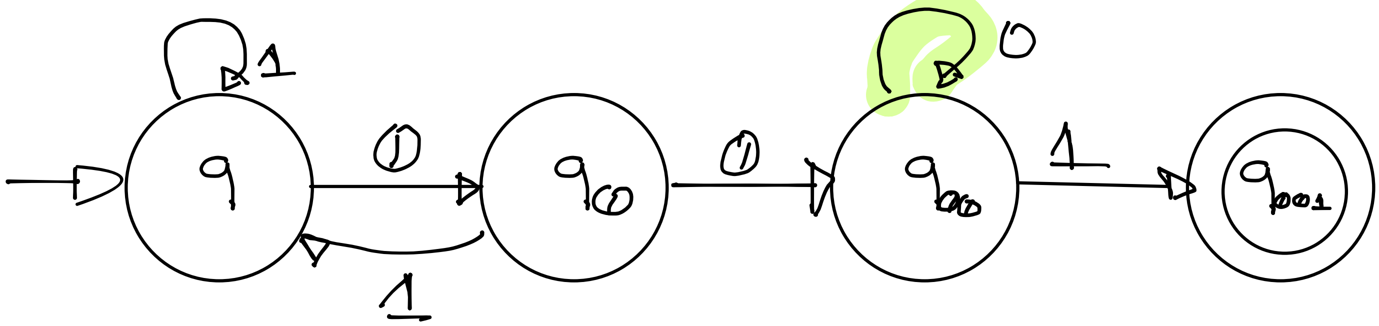
$Q_{2.1}$	What if the string starts with 1's?



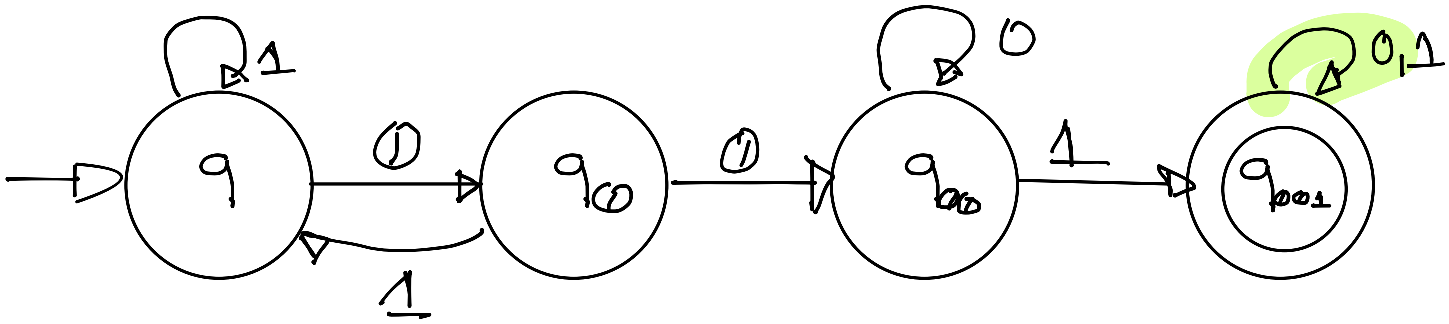
$Q_{2.2}$	What if after the first 0 we see an 1?



$Q_{2.3}$	What if after the second 0 we multiple 0's?



Q2.4 What if 001 other characters appear?



The regular operations

Idea: Investigate properties of finite automata and regular languages (this also implies the concept of non-regular languages)

Definition 1.23

Let $\underline{A}$ and $\underline{B}$ be languages. We define the regular operations union, concatenation and star as follows:

Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

Star: $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

(since $k \geq 0 \Rightarrow \varepsilon \in A^*$

Notes: A^* is the set of all strings over symbols in A including the empty string ε

Example 1.24

$\Sigma = \{a, b, c, \ldots, z\}$

$A = \{good, bad\}$ (Language A contains strings good and bad)

$B = \{boy, girl\}$ (Language B contains strings boy and girl)

Then:

$A \cup B = \{good, bad, boy, girl\}$

$A \circ B = \{goodboy, goodgirl, badboy, badgirl\}$

$A^* = \{\varepsilon, good, bad, goodgood, goodbad, badgood,$

$badbad, goodgoodgood, goodgoodbad, goodbadgood,$

$goodbadbad, \ldots\}$

Closed operation

A collection of objects is closed under some operation if applying the operation to members of the collection returns an object still in the collection

Example:

- Let $\mathbb{N} = \{1, 2, 3, \dots\}$ be the set of natural numbers
- $\mathbb{N}$ is closed under multiplication because $x \times y$ still returns a number $\in \mathbb{N}$
- $\mathbb{N}$ is not closed under division since x / y may produce a number $\notin \mathbb{N}$

First property we can learn:

Theorem 1.25

The class of regular languages is closed under the union operation. In other words, if A_1 and A_2 are regular languages so is $A_1 \cup A_2$

Proof Idea:

1. We have regular languages A_1 and A_2 and we want to show that $A_1 \cup A_2$ is also regular

2. Because A_1 and A_2 are regular:
   Some finite automaton M_1 recognizes A_1
   Some finite automaton M_2 recognizes A_2

3. To prove that $A_1 \cup A_2$ is regular:
   Demonstrate a finite automaton recognizing $A_1 \cup A_2$

(4) We construct $\underline{M}$ from $\underline{M_1}$ and $\underline{M_2}$.
Machine $\underline{M}$ must accept its input when either $\underline{M_1}$ and M_2 would accept (M simulates M_1 and M_2)

(5) How can M simulate M_1 and M_2?
↳ Simultaneous simulation with each input symbol
Just need to remember the state of each machine

Proof:

Let $\begin{cases} M_1 \text{ recognize } A_1 \text{ where } M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \\ M_2 \text{ recognize } A_2 \text{ where } M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \end{cases}$

Construct M to recognize $A_1 \cup A_2$ where $M = (Q, \Sigma, \delta, q_0, F)$

(1) $Q = \{ (r_1, r_2) \mid r_1 \in \overset{\in M_1}{Q_1} \text{ and } r_2 \in \overset{\in M_2}{Q_2} \}$
(set of all pairs, the $1^{st} \in Q_1$ and $2^{nd} \in Q_2$)

(2) We assume for simplicity that Σ is the same for M_1 and M_2

(3) δ is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$ let

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_1, a))$$

④ q_0 is the pair (q_1, q_2)

⑤ $F = \{ (r_1, r_2) \mid r_1 \in F_1$ or $r_2 \in F_2 \}$

(same as $(F_1 \times Q_2) \cup (Q_1 \times F_2)$)

∎ This concludes the construction of M.

Q:	So, what is the main conclusion that you can draw from Theorem 1.25?

∴ <u>The union of two regular languages is regular</u>

Let's see additional properties ☝

Theorem 1.26

The class of regular languages is closed under the concatenation operation

In other words: If A_1 and A_2 are regular languages so is $A_1 \circ A_2$

Proof idea (similar to the union proof)

① Start with finite automata M_1 and M_2 recognizing the regular languages A_1 and A_2.

② Instead of constructing automaton $\underline{M}$ to accept its input if either M_1 or M_2 accept:

Must accept if the input can be broken into two pieces:
- M_1 accepts 1^{st} piece
- M_2 accepts 2^{nd} piece

③ The problem is that M does not know where to break its input:

- Where does the 1^{st} part end?
- Where does the 2^{nd} part end?

To solve this problem we need to introduce nondeterminism.

1.2 Nondeterminism

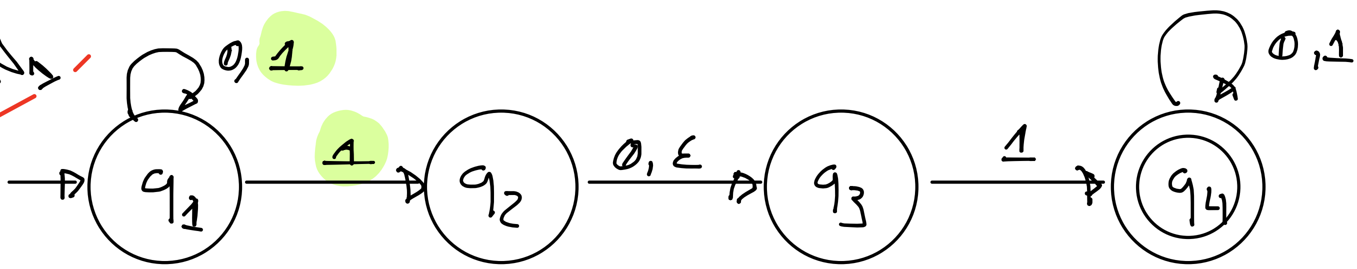
- **Determinism** – when a machine transitions to a single state based on an input (the machines we have seen so far)
- **Non-Determinism** – when a machine may transition to more than one state based on an input

Notes:

Non-determinism is a generalization of determinism, so every deterministic finite automaton (DFA) is automatically a nondeterministic finite automaton (NFA)

Q: Can you see anything peculiar with the next image?

NFA N_1



- State q_1 has two possible transitions for input 1
- State q_2 does not have a transition for input 1
- State q_2 has a transition for ε

Non-determinism:

$\because_1$ In a NFA, a state may have zero, one or many exiting arrows for each symbol

$\because_2$ A NFA may have arrows labeled with members of the alphabet or ε. Zero, one or many arrows may exit from each state with the label ε.

Q₁: But wait... How does a NFA compute? Imagine we are in q_1 of the previous example and the next input symbol is 1

1. Machine splits into multiple copies of itself and follows all possibilities in parallel
2. Each copy proceeds normally
3. If there are subsequent choices the machine splits again
4. If the next symbol does not appear on any of the arrows exiting the state (for any copy) then the machine dies
5. If any copy is in a final state, the NFA accepts string

Q_2: But what happens if a state with an arrow ε is encountered?

1. Without reading any input, the machine splits into multiple copies:

   1.1 One copy for each ε-transition

   1.2 One copy staying at the same state

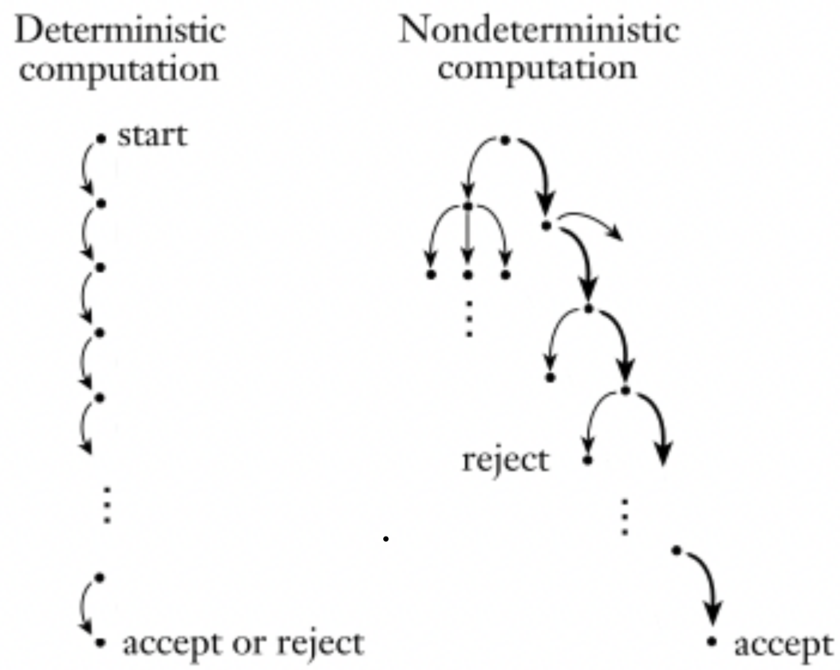
2. The continues nondeterminiically

Q: Does this behavior remind you of anything from the Operating Systems course?

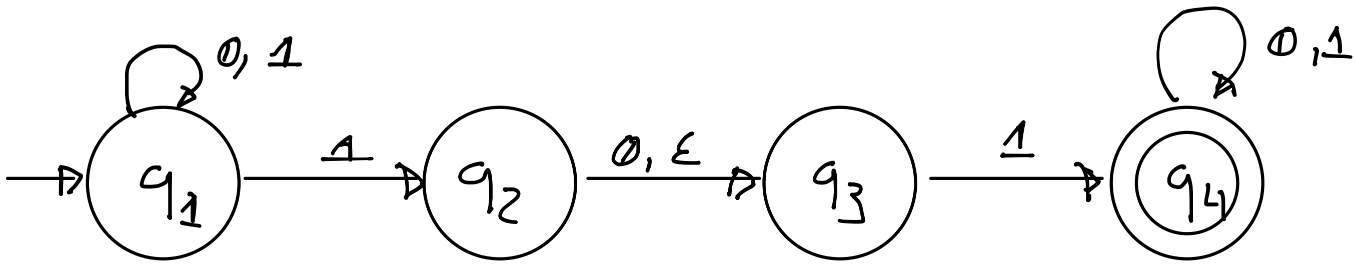
- Processes and Threads

- Every time an NFA splits: fork/new thread

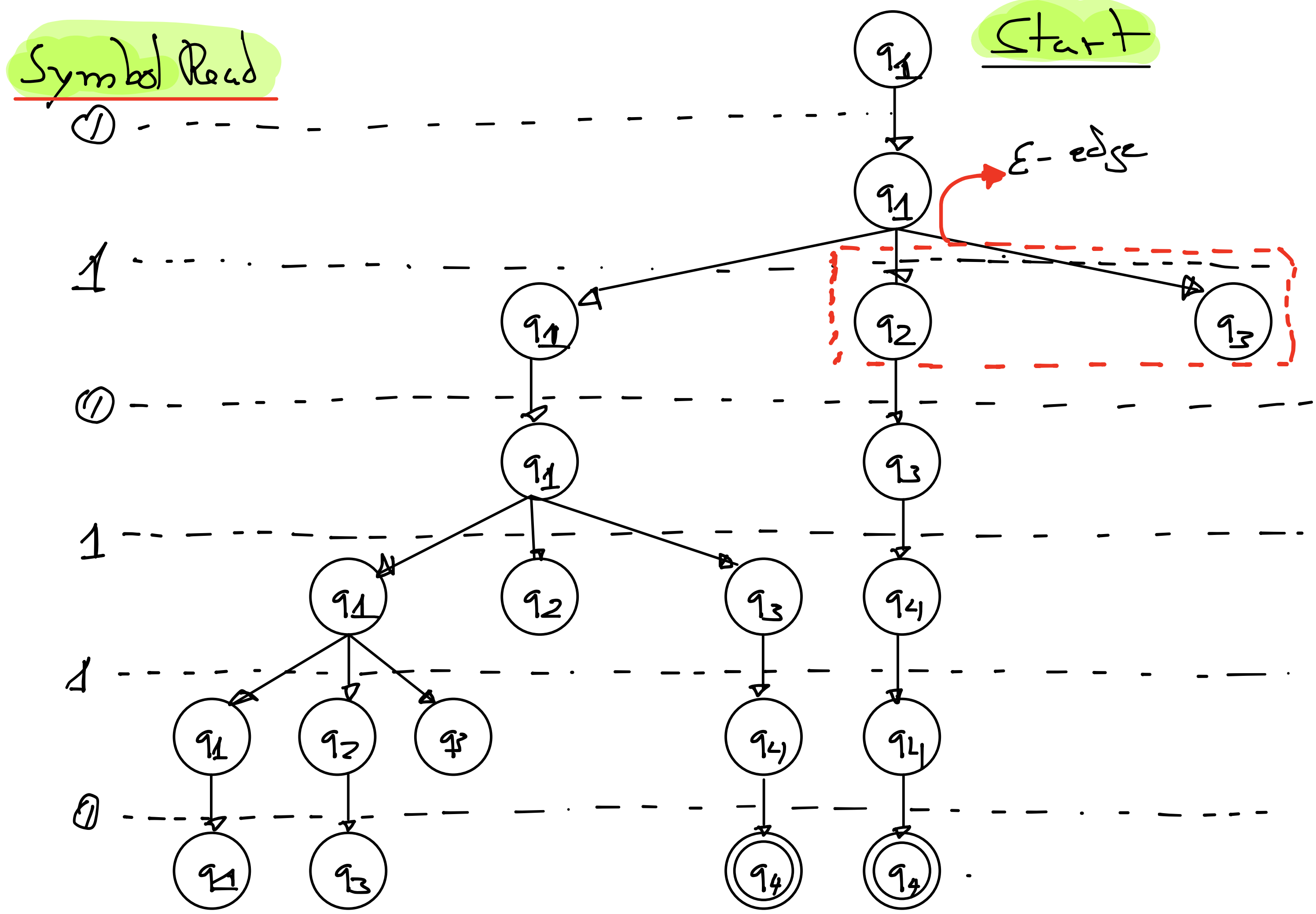
- Simple way to visualize determinism vs. nondeterminism



Let's go back to the previous example (N_1):



Consider input string: 0 1 0 1 1 0



Q: What is the language recognized by N_1?

↳ N_1 accepts all strings that contain 101 or 11 as a substring

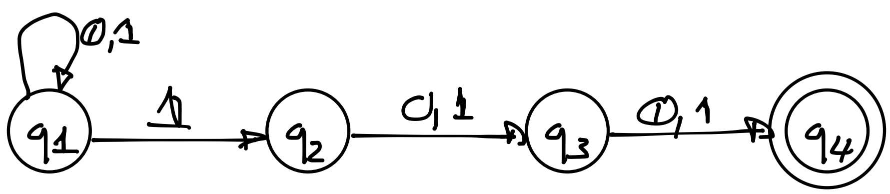
NFA are useful in $\neq$ manners:

- Every NFA can be converted to an equivalent DFA
- Constructing NFA may be easier than building an equivalent DFA

Example 1.30

Let A be the language consisting of all strings over $\{0,1\}$ containing a 1 in the third position from the end (e.g. $000100 \in A$, but $0011 \notin A$)

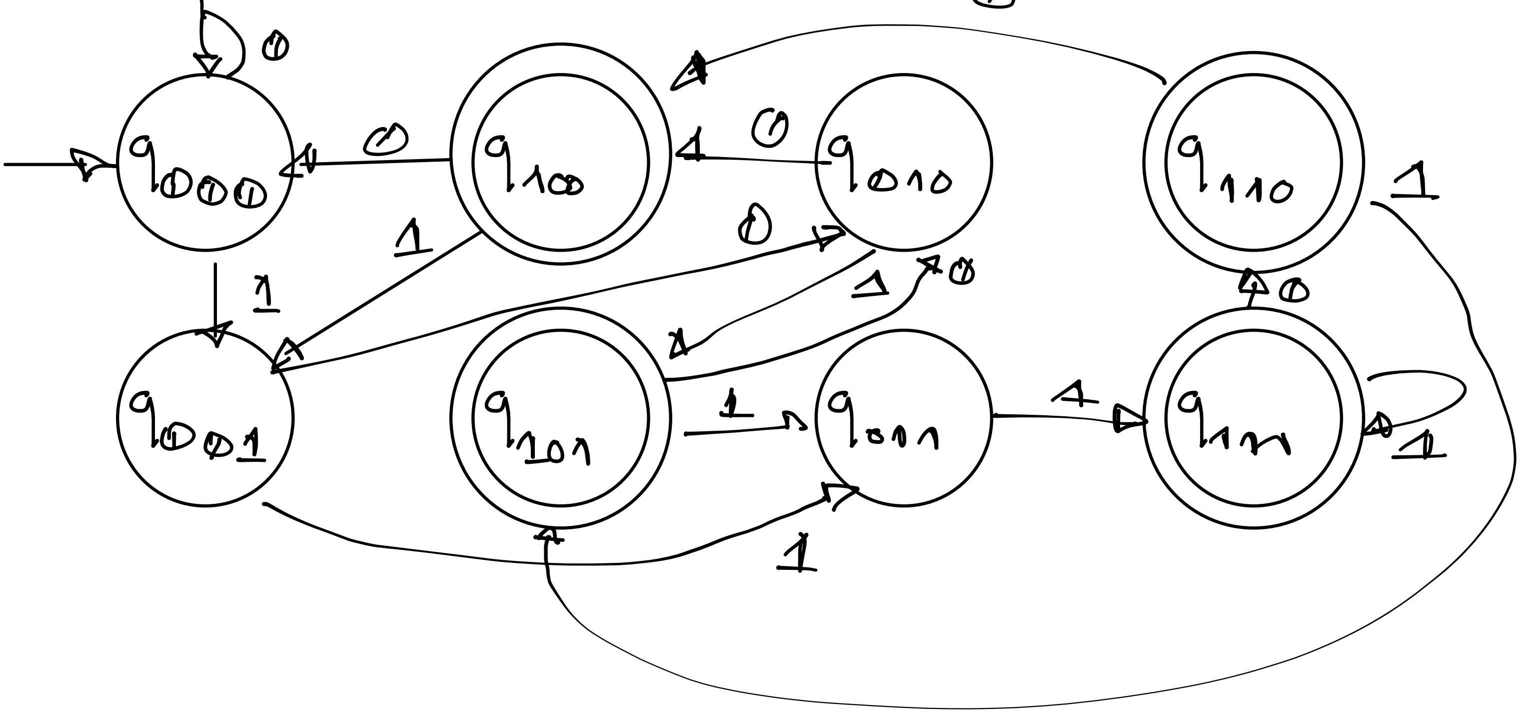
The following NFA N_2 recognizes A:



Notes:

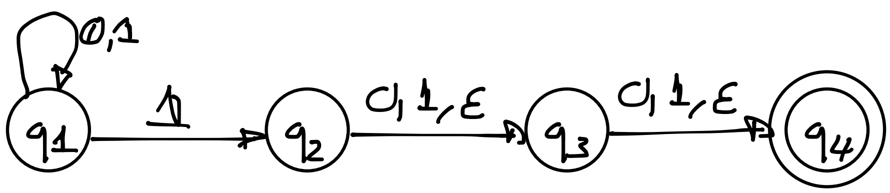
N_2 stays in the initial state until it 'guesses' that it is three positions from the end

The equivalent DFA:

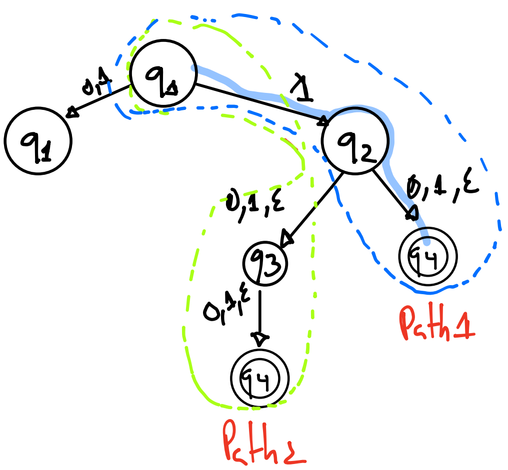


Example [Optional]

Let us alter N_2 in the following manner:



Q:	What language does this NFA recognize?



Possibilities:

Path 1: 1
- 0 --- 10
- 1 --- 11
- ε --- 1ε=1

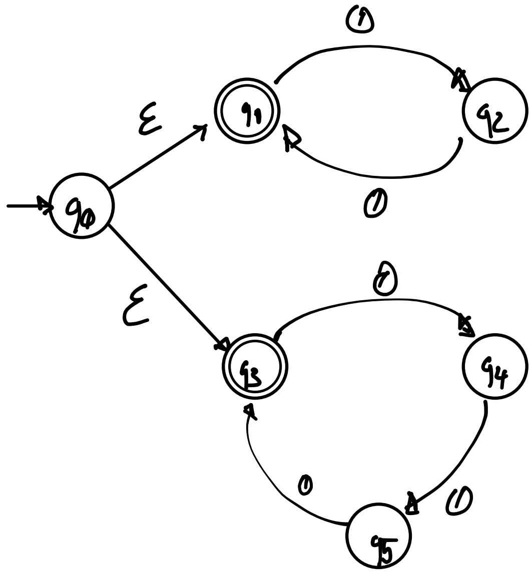
Path 2: 1
- 0
  - 0 ... 100
  - 1 — 101
  - ε — 10ε=10
- 1
  - 0 — 110
  - 1 — 111
  - ε — 11ε=11
- ε
  - 0 — 1ε0=10
  - 1 — 1ε1=11
  - ε — 1εε=1

∴ The altered NFA recognizes the language consisting of all strings over {0,1} that contain at least a 1 in of the last three positions

Example 1.33

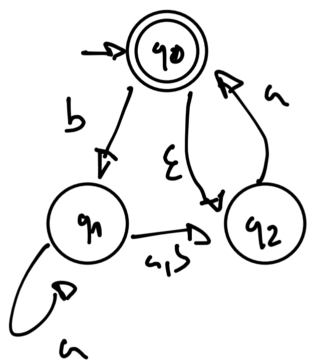
Construct a NFA capable of recognizing 0^k where k is a multiple of 2 or 3 and $0^0 = \varepsilon$. Example: ε, 00, 0000, 000000, ...
ε, 000, 000000, 000000000

[resolution:]



Example 1.35

Give examples of strings accepted and not accepted by the following NFA



[resolution:]

Accepted: ε, a, baba, baa

Not accepted: b, bb, babba

Formal Definition of NFA

- Similar to DFA except for trancition function (δ)

  - DFA: δ receives a state and an input and produces next state

  - NFA: δ receives a state an an input symbol or ε and produces set of possible states

Additional notation:

- $P(Q)$ represent the collection of all subsets of Q (power set)

- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Definition 1.37

A NFA is a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$:

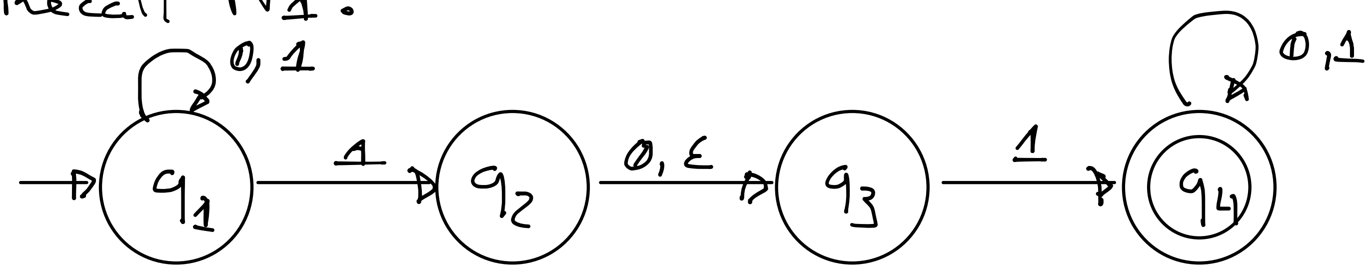
1. Q is a set of finite states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ (transition function)
   ("maps to a combination of Q")
4. $q_0 \in Q$ (initial state)
5. $F \subseteq Q$ (final states)

N accepts w if we can write w as $y_1 y_2 \dots y_m$ ($y_i \in \Sigma_\varepsilon$) and a sequence of states $r_0, r_1, \dots, r_m$ ($r_i \in Q$) with three conditions:

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1}) \ \forall i \in [0, m-1]$
3. $r_m \in F$

Example 1.38

Recall N_1:



The formal definition of $N_1 = (Q, \Sigma, \delta, q_0, F)$:

1. $Q = \{q_1, q_2, q_3, q_4\}$

2. $\Sigma = \{0, 1\}$

3.

δ	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	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. $q_0 = q_1$ is the start state

5. $F = \{q_4\}$

Equivalence of NFA and DFA

- Both recognize same class of languages

Surprising since NFA appear to be more powerful than DFA

Theorem 1.39

Every NFA has an equivalent DFA
(recognizes same language)

Proof Idea:

1. Convert NFA to a DFA that simulates the NFA

2. We need to know if the computation is in one of the states. If multiple copies of the machine exist we only need to keep track if there is an instance active of the state.

- If k states exist, then 2^k subsets of the states exist

$$\underset{q_1}{(A/\bar{A})} \times \underset{q_2}{(A/\bar{A})} \times \cdots \underset{q_k}{(A/\bar{A})} = 2^k$$

3. DFA will need to have 2^k states

Proof

Let:

- $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing language A

- $M = (Q', \Sigma, \delta', q_0', F')$ be the DFA recognizing language A

Case 1: N has no ε-edges

① $Q' = P(Q)$ (power set of Q, $|P(Q)| = 2^k$)
Every state of M is a set of states of N

R represents a set of states of N

② For $R \in Q'$ and $a \in \Sigma$ let

Notation 1: $\delta'(R, a) = \{ q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R \}$

Notation 2: $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$

③ $q_0' = \{ q_0 \}$

④ $F' = \{ R \in Q' \mid R \text{ contains a final state of } N \}$
Accept if one of the states of R is a final state of N

Case 2: $\underline{N}$ has ε-edges

- Let $E(R)$ be the collection of states that can be reached from members of $\underline{R}$ by going only along ε-edges, including the member of R themselves. For $R \subseteq Q$:

$$E(R) = \{ q \mid q \text{ can be reached from } R \text{ by traveling along } 0 \text{ or more } \varepsilon\text{-edges} \}$$

- The transition function δ' needs to be updated:

$$\delta'(R, a) = \{ q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R \}$$

$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$$

- Additionally: start state of M needs to contemplate ε-edges

$$q_0' = \{ E(q_0) \}$$

Q: What is the main conclusion that you can draw from Theorem 1.39?

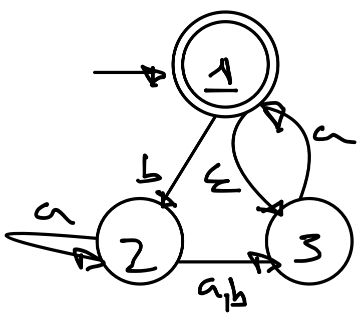
∴ Every NFA can be converted into an equivalent DFA.

Corollary 1.40

A language is regular if and only if some NFA recognizes it

Example 1.41

Convert NFA N_4 to a DFA:



[Resolution:]

$N_4 = (Q, \Sigma, \delta, q_0, F)$
(NFA)

- $Q = \{1, 2, 3\}$
- $\Sigma = \{a, b\}$
- $q_0 = 1$
- $F = \{1\}$

Let $D = (Q', \Sigma_E, \delta', q_0', F)$

$- Q' = P(Q)$

$= \{ \emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\} \}$

Binary Encoding: (000 , 001 , 010 , 011 , 100 , 101 , 110 , 111)

$- q_0' = E(q_0) = E(1) = \{1,3\}$

$- F' = \{ \{1\}, \{1,2\}, \{1,3\}, \{1,2,3\} \}$

$$- \delta'(R,a) = \bigcup_{r \in R} E(\delta(r,a))$$

1. $\delta'(\emptyset, a) = \emptyset$
 $\delta'(\emptyset, b) = \emptyset$ } Special case state for when no transitions exist

2. $\delta'(1, a) = E(\delta(1,a)) = E(\emptyset) = \emptyset$
 $\delta'(1, b) = E(\delta(1,b)) = E(2) = \{2\}$

3. $\delta'(2, a) = E(\delta(2,a)) = E(\{2,3\}) = \{2,3\}$
 $\delta'(2, b) = E(\delta(2,b)) = E(\{3\}) = \{3\}$ → ε-edge

4. $\delta'(3, a) = E(\delta(3,a)) = E(\{1\}) = \{1,3\}$
 $\delta'(3, b) = E(\delta(3,b)) = E(\emptyset) = \emptyset$

5. $\delta'(\{1,2\}, a) = E(\delta(1,a)) \cup E(\delta(2,a)) = E(\emptyset) \cup E(\{2,3\}) = \emptyset \cup \{2,3\} = \{2,3\}$
 $\delta'(\{1,2\}, b) = E(\delta(1,b)) \cup E(\delta(2,b)) = E(\{2\}) \cup E(\{3\}) = \{2,3\}$

6. $\delta'(\{1,3\}, a) = E(\delta(1,a)) \cup E(\delta(3,a)) = E(\emptyset) \cup E(\{1\}) = \emptyset \cup \{1,3\} = \{1,3\}$
 $\delta'(\{1,3\}, b) = E(\delta(1,b)) \cup E(\delta(3,b)) = E(\{2\}) \cup E(\emptyset) = \{2\} \cup \emptyset = \{2\}$

(7) $\delta'(\{2,3\}, a) = E(\delta(2,a)) \cup E(\delta(3,a)) = E(\{2,3\}) \cup E(\{1\}) = \{2,3\} \cup \{1,3\} = \{1,2,3\}$

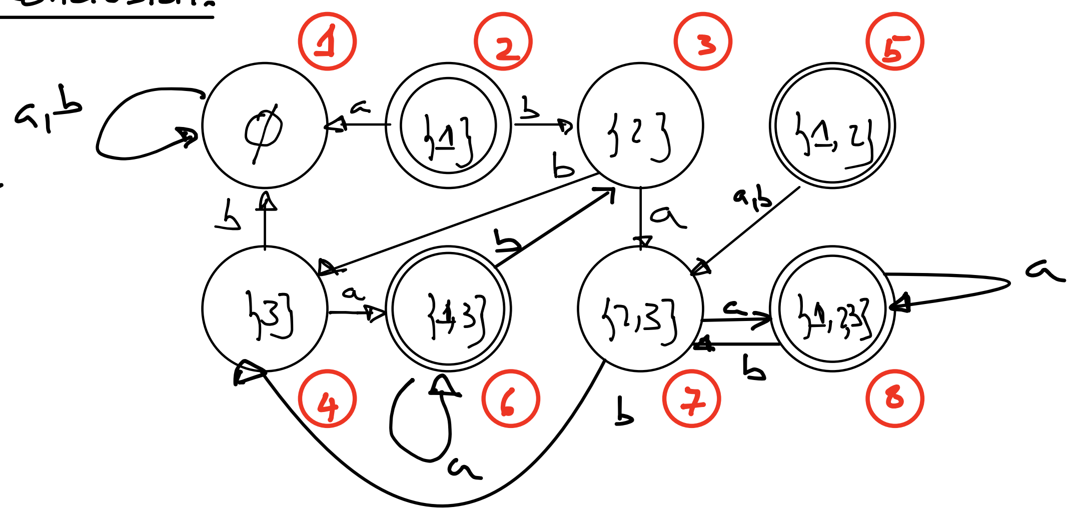
$\delta'(\{2,3\}, b) = E(\delta(2,b)) \cup E(\delta(3,b)) = E(\{3\}) \cup E(\emptyset) = \{3\}$

(8) $\delta'(\{1,2,3\}, a) = E(\delta(1,a)) \cup E(\delta(2,a)) \cup E(\delta(3,a)) = E(\emptyset) \cup E(\{2,3\}) \cup E(\{1\}) = \emptyset \cup \{2,3\} \cup \{1,3\} = \{1,2,3\}$

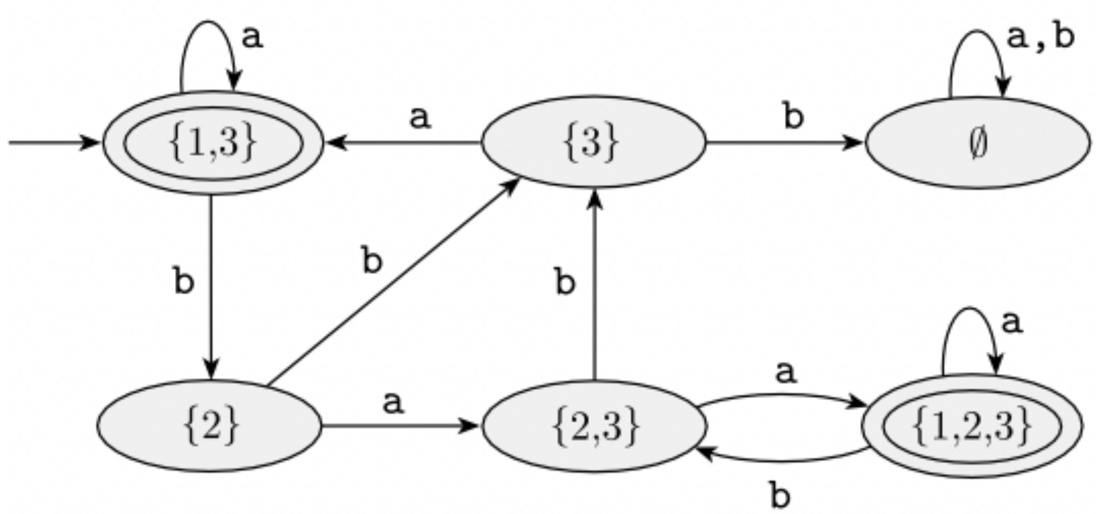
$\delta'(\{1,2,3\}, b) = E(\delta(1,b)) \cup E(\delta(2,b)) \cup E(\delta(3,b)) = E(\{2\}) \cup E(\{3\}) \cup E(\emptyset) = \{2\} \cup \{3\} \cup \emptyset = \{2,3\}$

In conclusion:

$\therefore_1$



$\therefore_2$ Because there are no arrows pointing to states {1} and {1,2} they can be removed. Why? Because there is no way of reaching those states. New figure:



Closure under the regular operations

- We return to the original proofs of union, concatenation and star operations of regular languages are still regular (closure)
  (continuation of pages 26/27 of this PDF)

- At the time the proof was abandoned because it was too complicated

- We can now attempt to prove using NFA
  (in fact, lets prove for the union also)

Theorem 1.45

The class of regular languages is closed under the union operation

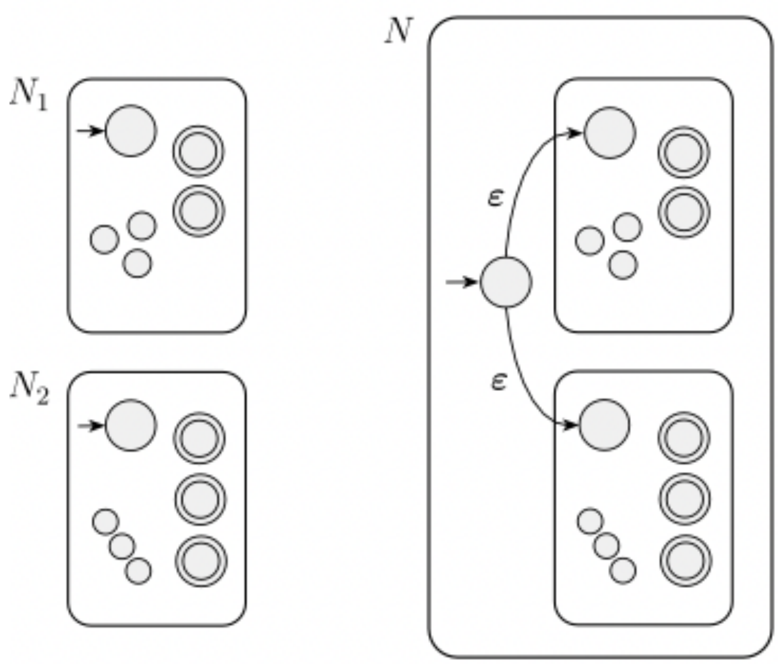
Proof Idea:

1. We have regular languages A_1 and A_2 and we want to prove that $A_1 \cup A_2$ is also regular

2. Idea: Take two NFA, N_1 and N_2 for A_1 and A_2 and combine them into one new NFA N

3. N must accept the input if either N_1 or N_2 accepts this input

4. The new machine has a new start state that branches to the start states of the old machines with ε-edges. This starts N_1 and N_2 simultaneously:



Proof:

Let $\begin{cases} N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \text{ recognize } A_1 \\ N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \text{ recognize } A_2 \end{cases}$

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:

① $Q = \{q_0\} \cup Q_1 \cup Q_2$

② q_0 is the initial state of Q

③ $F = F_1 \cup F_2$ (N accepts if N_1 or N_2 accepts)

④ $$\delta(q, a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

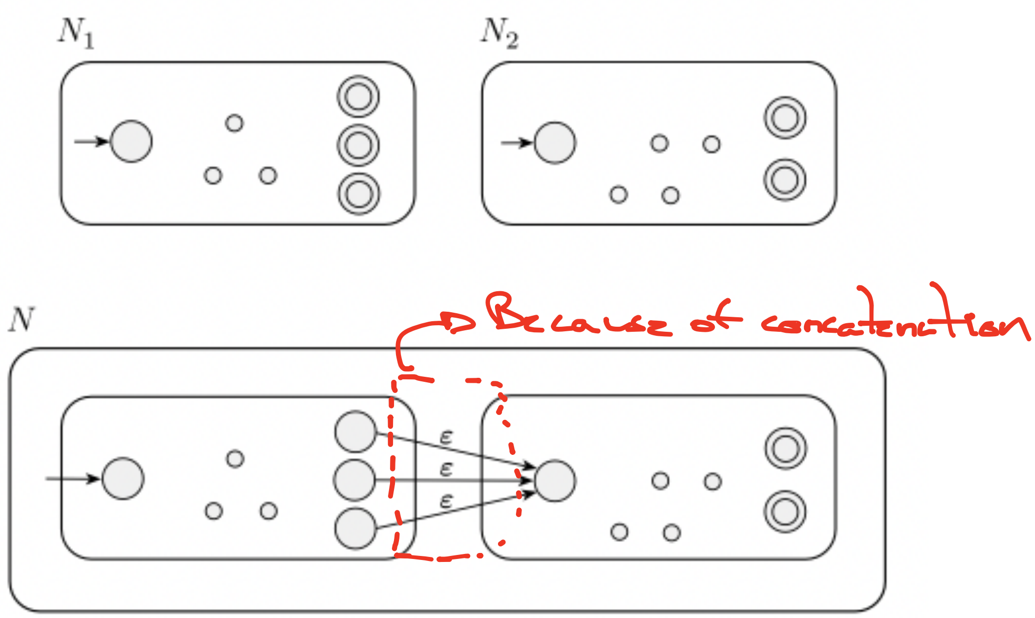
Theorem 1.47

The class of regular languages is closed under the concatenation operation.

Proof Idea:

① We have regular languages A_1 and A_2 and want to prove $A_1 \circ A_2$ is regular

② Idea: Take two NFA, N_1 and N_2 for A_1 and A_2, and combine them into a new NFA N as we did for the case of the union but in a different manner:



③ The initial state of N must be the initial of N_1
The final states of N_1 have ε-edges to N_2.
Whenever N_1 is in a final state the machine automatically branches to N_2. This means that the machine has found an initial piece of the input that constitutes a ~~string~~ in A_1

④ The final states of N are the final states of N_2
This means that the input is accepted if it can be split into two parts:

1st - recognized by N_1
2nd - recognized by N

Proof:

Let
$$\begin{cases} N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \text{ recognize } A_1 \\ N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \text{ recognize } A_2 \end{cases}$$

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \circ A_2$:

① $Q = Q_1 \cup Q_2$

② The initial state of N is q_1 of N_1

③ The final state of N are those of N_2: $F = F_2$

④
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & , q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & , q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & , q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a) & , q \in Q_2 \end{cases}$$

Theorem 1.49

The class of regular languages is closed under the star operation

Q: Do you remember what is the star operation?

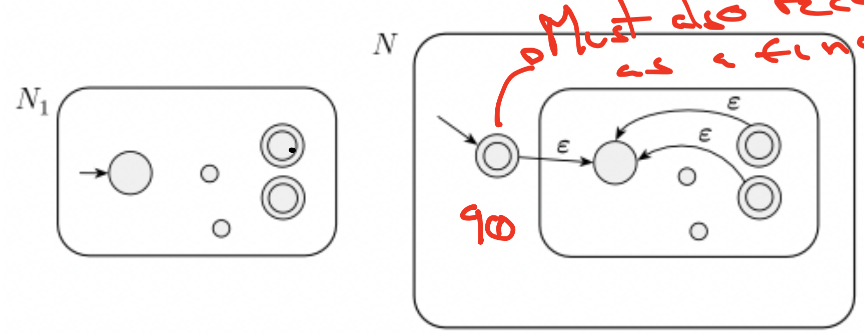
Notes: A^* is the set of all strings over symbols in A including the empty string ε

Proof Idea:

① We have a regular language A_1 and want to prove that A_1^* is also regular

② Idea: Take a NFA N_1 for A_1 and modify it to recognize A_1^*:

Must also recognize ε as a final state



③ N will accept its input whenever it can be broken into several pieces and N_1 accepts each piece

(4) N can be constructed like N_1 with additional ε-edges to the initial state from the final states. Thus way, when processing gets to the end of a piece that N_1 accepts, the machine N has the option of jumping back to the initial state to ~~try~~ to read another piece that N_1 accepts

(5) N must be modified to accept $\underline{\varepsilon}$ which $\in A_1^*$

Proof

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct N

$N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*:

1. $Q = \{q_0\} \cup Q_1$

2. q_0 is the initial state of N

3. $F = \{q_0\} \cup F_1$ (since A_1^* must include ε)

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

1.3 Regular Expressions

Q:	What is a regular expression?

→ Mechanism to describe languages

Example: $(0 \cup 1)0^*$ → star Operator (see page 20)

Q:	What is the language of this regular expression?

→ Strings that start with a 0 or a 1 followed by zero or more (*) 0's.

Example 1.54.a $(0 \cup 1)^*$

Q:	What is the language of this regular expression?

→ Language consisting of all possible strings of 0s and 1s.

Example 1.54.b

Let $\Sigma = \{0, 1\}$ then we can write Σ as shorthand for the regular expression $(0 \cup 1)$

Q:	What is the language of this regular expression?

language consisting of all string of length 1 over Σ

Example 1.51.c

Q:	What is the language of the regular expression Σ^*?

language consisting of all string over Σ

Example 1.51.d

Q:	What is the language of the regular expression $\Sigma^* 1$?

language consisting of all string over Σ that terminate in 1

Example 1.51.e

Q:	What is the language of the regular expression $(0\Sigma^*) \cup (\Sigma^* 1)$

→ language consisting of all strings over Σ that start with a zero or terminate with a 1.

Important Notes about precedence

Order is important:

$1^{\underline{st}}$ Star operation
$2^{\underline{nd}}$ Concatenation
$3^{\underline{rd}}$ Union

} Unless there is parentheses

Q:	So what is the formal definition of a regular expression?

Definition 1.52

R is a regular expression if R is:

1. A symbol $a \in \Sigma$
2. ε (empty string, $\neq$ from empty language)
3. $\emptyset$ (empty language, contains zero string)
4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
   I.e. R can be represented as a union of two regular expressions
5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
   I.e. R can be represented as a concatenation of two regular expressions
6. (R_1^*) where R_1 is a regular language
   I.e. R is the star operation of a regular expression

Important Definitions:

R^* - zero or more concatenations of strings from R

R^+ - one or more concatenations of strings from R, i.e. $R^+ = RR^*$

R^k - k concatenations of strings from R

EXAMPLE 1.53

In the following instances, we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w|\ w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w|\ w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w|\ w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w|\ \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w|\ w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w|\ \text{the length of } w \text{ is a multiple of } 3\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w|\ w \text{ starts and ends with the same symbol}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
11. $1^*\emptyset = \emptyset$.
    Concatenating the empty set to any set yields the empty set.
12. $\emptyset^* = \{\varepsilon\}$.
    The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

■

Equivalence with Finite Automata

Fun fact: R.E are equivalent to finite automata

Q: But what does this mean that they are equivalent

Any R.E. can be converted to a finite automaton (and vice versa)

Recall: Language is regular if we can build an automaton for it

Theorem 1.54

A language is regular iff some regular expression describes it

Because of the iff the proof needs to be divided onto two parts. Each part will be a different Lemma (Lemma 1.55 and Lemma 1.60)

Lemma 1.55

If a language is described by a R.E. then it is regular

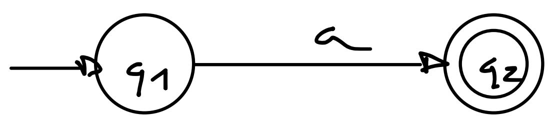
Proof Idea

1. R is a R.E describing language A
2. Show how to convert R into an NFA recognizing A
3. By Corollary 1.40: If an NFA recognizes A, then A is regular

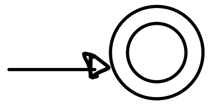
Proof: Let's convert R into an NFA N.

We consider the six cases in the formal definition of R.E

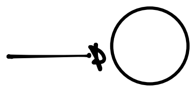
1. $R = a, a \in \Sigma$. Then $L(R) = \{a\}$ and the following NFA recognizes $L(R)$



(2) $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$ and the following NFA recognizes $L(R)$



(3) $R = \emptyset$. Then $L(R) = \emptyset$ and the following NFA recognizes $L(R)$



(4) $R = R_1 \cup R_2$

Class of regular languages is closed for union

(5) $R = R_1 \circ R_2$

Class of regular languages is closed for concatenation

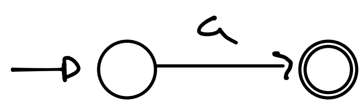
(6) $R = R_1^*$

Class of regular languages is closed for star

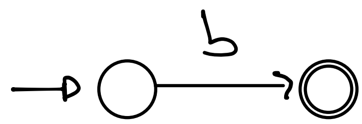
Example 1.56

Convert the R.E. $(ab \cup a)^*$ to a NFA.

1 (N_1) a



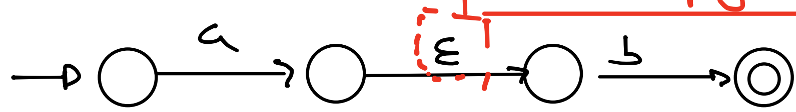
2 (N_2) b



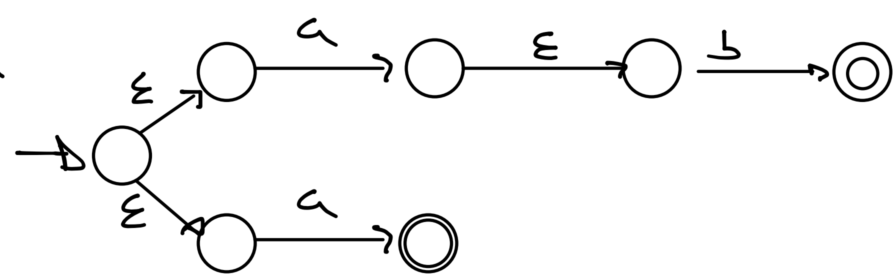
Notes:

Why the ε-edge?
We are combining two NFA N_1 and N_2
(see page 48)

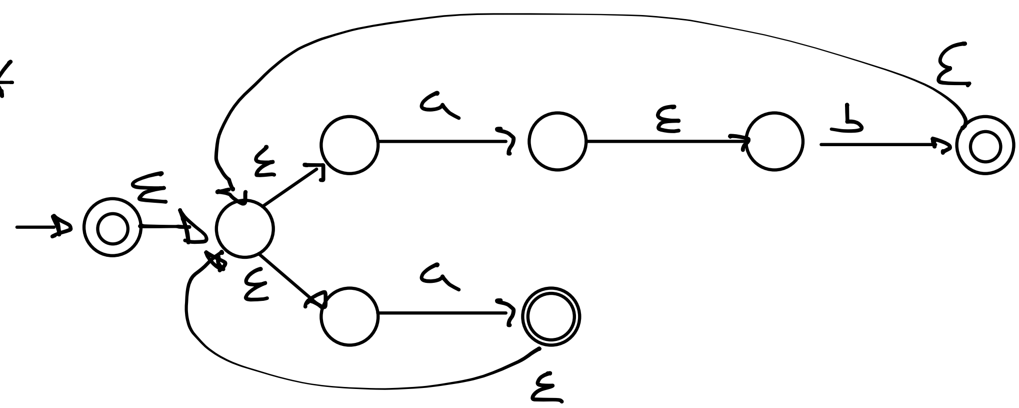
3 ab



4 $ab \cup a$



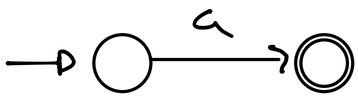
5 $(ab \cup a)^*$



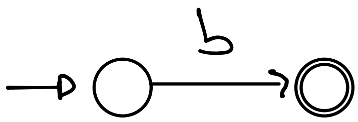
Example 1.58

Convert the R.E. $(a \cup b)^* aba$ to a NFA.

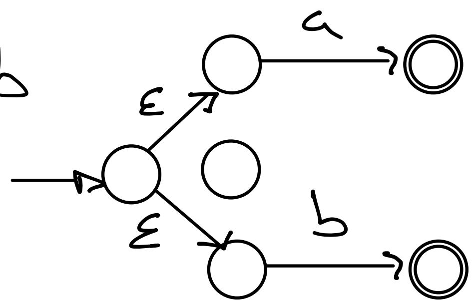
1. a



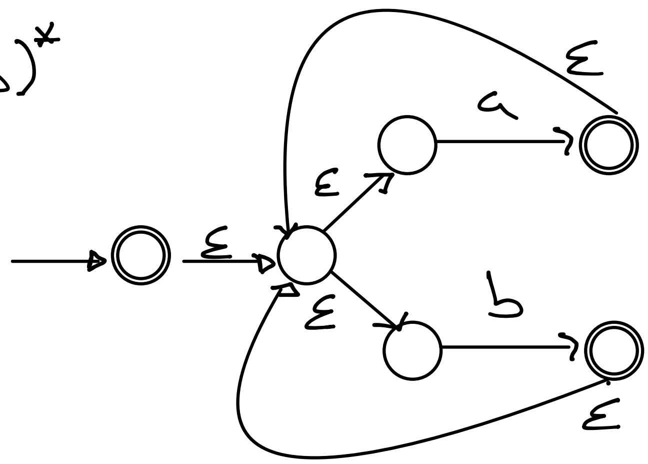
2. b



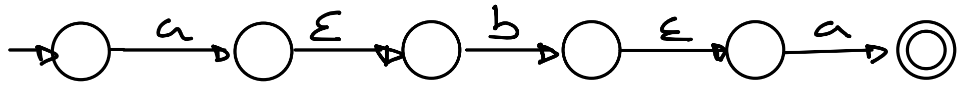
3. $a \cup b$



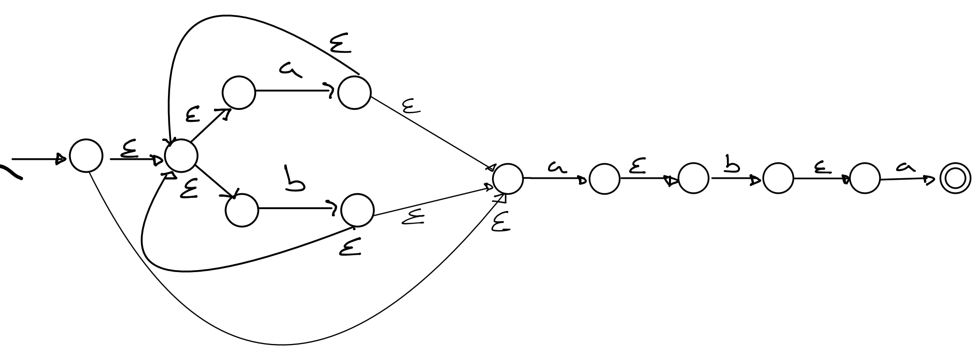
4. $(a \cup b)^*$



5. aba



6. $(a \cup b)^* aba$



Lemma 1.60

If a language is regular then it's described by a RE

Proof Idea:

1. We need to show that if a language $\underline{A}$ is regular, a R.E describes it.

2. Because $\underline{A}$ is regular then it is accepted by a DFA. We describe a procedure for converting DFAs into equivalent R.E.

3. This procedure is broken into two parts. The first part will use a generalized nondeterministic finite automaton (GNFA). ① First we show how to convert DFAs into GNFAs, and ② then GNFAs into R.E. (see page 66 and 67)

Q: So, what is a GNFA?

1. NFA where transitions can be R.E. instead of only members of Σ or ε.

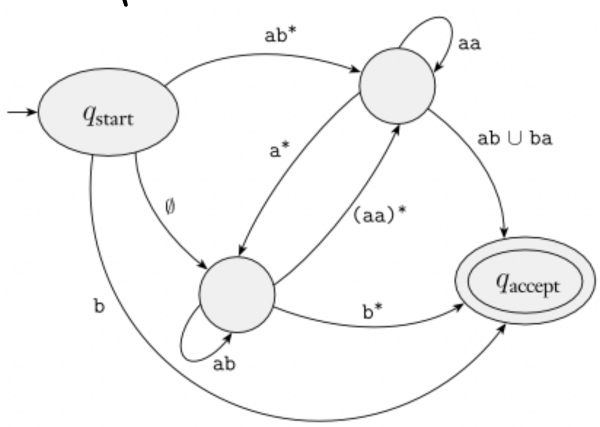
2. GNFA reads blocks of symbols (not necessarily just one symbol at a time as in an NFA)

3. GNFA moves along a transition arrow connecting two states by reading a block of symbols from the input

4. A GNFA is nondeterministic and may have $\neq$ ways to process the same string

5. A GNFA accepts / recognizes if it is in an end state at the end of the input.

Example



GNFAs have a special form:

- Start state has transitions to every other state but no incoming edges
- There is only a single final state with arrows from every other state but no outgoing edges. Furthermore, the final state is not the initial state
- Except for the initial and final states, one arrow goes from every state to every other state and also from each state to itself.

(1) We can easily convert a DFA into a GNFA: (see page 64)

(1) Simply add a new start state with an ε-arrow to the old start state and a new final state with ε arrows

(2) If multiple arrows with $\neq$ labels exist between two states then it is possible to replace with a single arrow whose label is the union of the previous labels.

(3) Finally, we add arrows labeled $\emptyset$ between states that had no arrows. This last step won't change the language recognized because a transition labeled with $\emptyset$ can never be used.

② We can then convert a GNFA into a RE

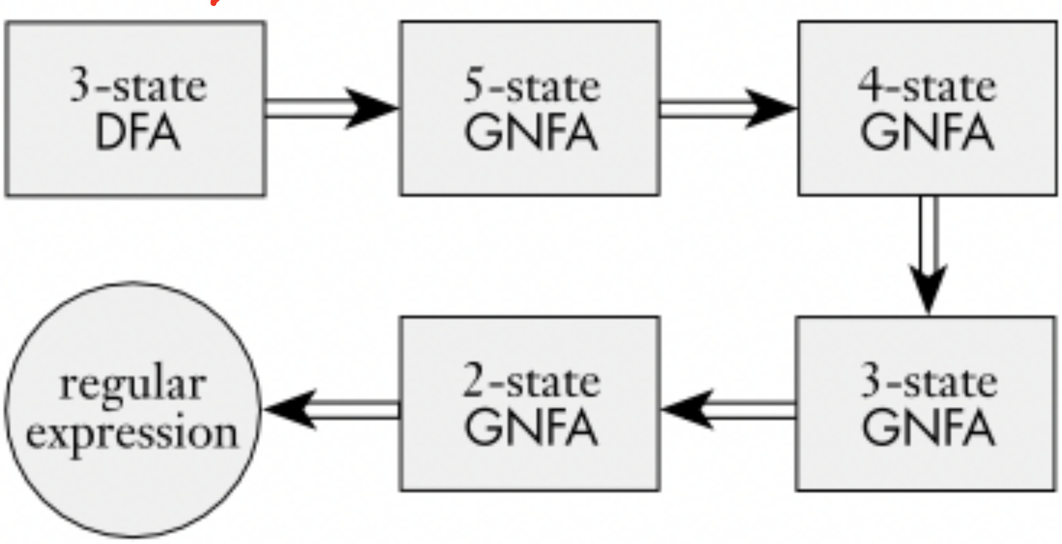
(2.1) Assume the GNFA has k states. Because a GNFA must have an initial and a final state and they have to be $\neq$ then $k \geq 2$

(2.2) If $k > 2$ then we construct an equivalent GNFA with $k-1$ states (it is possible to do so we just haven't seen it yet)

(2.3) If $k = 2$ the GNFA has a single arrow that goes from the start state to the final state. The label of this arrow is the equivalent RE.

Why? See page 64. Basically we need to add a new initial state and a new final state

Example



Q: So, how can we construct an equivalent GNFA with one fewer state?

(2.4) We do so by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized.

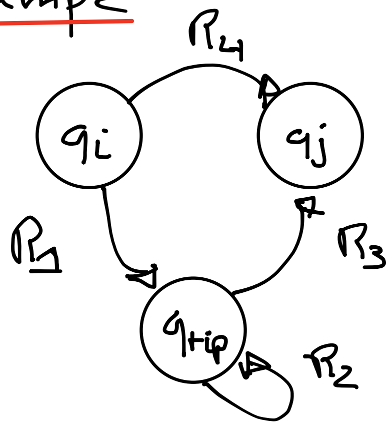
(2.5) Any state will do, provided it is not the initial state nor the final state.

(2.6) Because $k > 2$ this state will always exist. Let's call it q_{rip} (removed state)

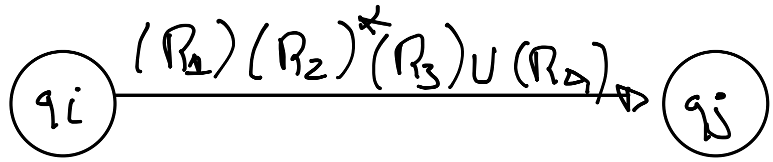
(2.7) After removing q_{rip} we repair the machine by altering the RE that label each of the remaining arrows. The new labels compensate for the absence of q_{rip} by adding back the lost computations.

(2.4) The new label going from a state q_i to q_j is a RE that describes all strings that would take the machine from q_i to q_j either directly or via q_{rip}

Example



(before)



(after)

$\therefore$ The new machine recognizes the original language.

Proof

① Start by formally defining the GNFA, which is similar to a NFA except for δ:

$$\delta : \underbrace{(Q - \{q_{accept}\})}_{\text{state } q_i} \times \underbrace{(Q - \{q_{start}\})}_{\text{state } q_j} \to R$$

- R := collection of all REs over alphabet Σ

- $\delta(q_i, q_j) = R$ the arrow from state q_i to q_j has the RE R as its label.

Definition 1.64

A GNFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$:

1. Q is the finite set of states
2. Σ is the input alphabet
3. $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow R$ is the transition function
4. q_{start} is the initial state
5. q_{accept} is the final state

star operation

A GNFA accepts a string w in $\boxed{\Sigma^*}$ if $w = w_1 w_2 \dots w_k$ ($w_i \in \Sigma^*$) and a sequence of states $q_0 q_1 \dots q_k$ exists such that:

1. $q_0 = q_{start}$ is the initial state
2. $q_k = q_{accept}$ is the accept state
3. Language of the RE R_i
   $\forall i \; w_i \in \boxed{L(R_i)}$ where $R_i = \delta(q_{i-1}, q_i)$
   (i.e. R_i is the RE on the arrow from q_{i-1} to q_i

We are now able to return to the proof of Lemma 1.60:

① Let M be the DFA for language A

② Convert M to GNFA G (add new start state, new final state and required additional edges)

③ Use procedure Convert (G) which receives as input a GNFA (G) and returns an equivalent RE

Convert (G) {

1. Let k be the number of states of G
2. If $k=2$ then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a RE R
   Return R
3. If $k>2$ select any state $q_{rip} \in Q$ $\neq$ from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$ where

   $$Q' = Q - \{q_{rip}\}$$

   and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$ let

   $$\delta(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$

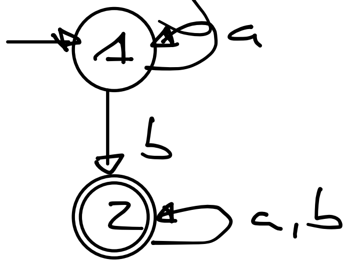
   where:

   $$R_1 = \delta(q_i, q_{rip})$$
   $$R_2 = \delta(q_{rip}, q_{rip})$$
   $$R_3 = \delta(q_{rip}, q_j)$$
   $$R_4 = \delta(q_i, q_j)$$

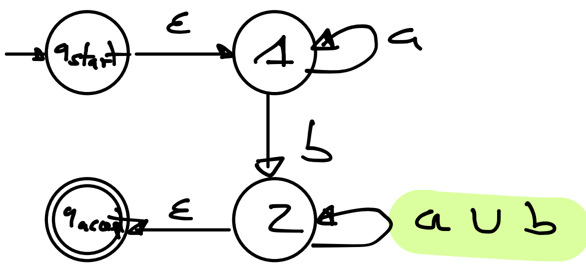
4. Return Convert(G') (recursion)

Example 1.67

Convert the following two-state DFA into a RE



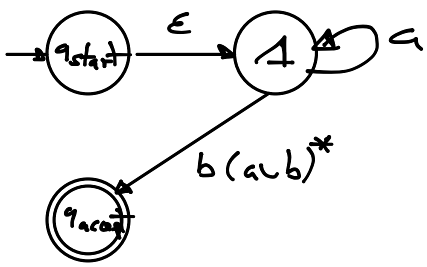
① Convert to a GNFA:



Notes:

- We are omitting the $\emptyset$ edges but they still exist

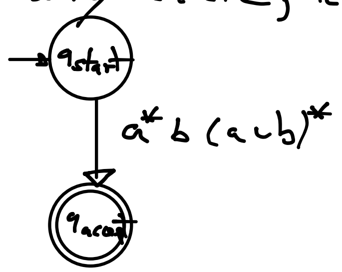
② We can remove any state, let's remove state 2:



Notes:

- $q_i = 1$, $q_{rip} = 2$, $q_j = q_{accept}$
- $R_1 = \delta(q_i, q_{rip}) = \delta(1,2) = b$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(2,2) = a \cup b$
- $R_3 = \delta(q_{rip}, q_j) = \delta(2, q_{accept}) = \epsilon$
- $R_4 = \delta(q_i, q_j) = \delta(1, q_{accept}) = \emptyset$
- $(R_1)(R_2)^*(R_3) \cup (R_4) = (b)(a \cup b)^* \epsilon \cup \emptyset$
  $= b(a \cup b)^*$

③ We can remove any state, let's remove state 1:

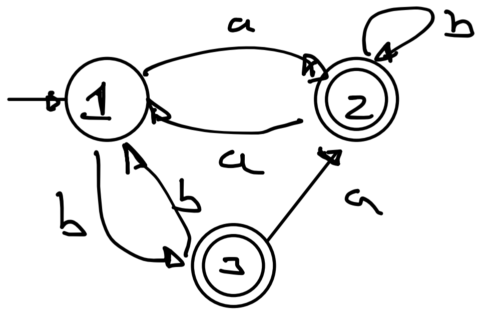


Notes:

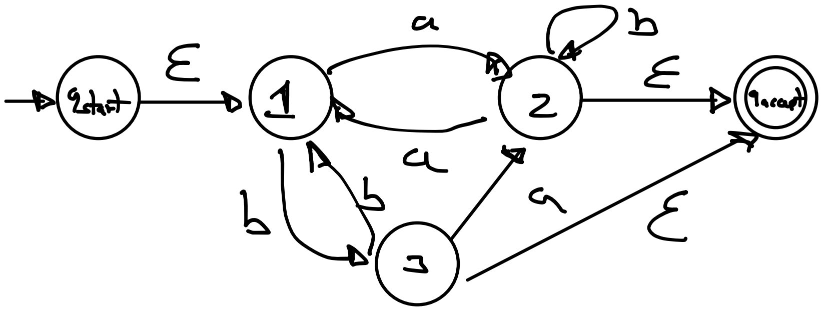
- $q_i = q_{start}$ $q_{rip} = 1$, $q_j = q_{accept}$
- $R_1 = \delta(q_i, q_{rip}) = \delta(q_{start}, 1) = \epsilon$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = a$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1, q_{accept}) = b(a \cup b)^*$
- $R_4 = \delta(q_i, q_j) = \delta(q_{start}, q_{accept}) = \emptyset$
- $(R_1)(R_2)^*(R_3) \cup (R_4) = (\epsilon)(a)^*(b(a \cup b)^*) \cup \emptyset$
  $= a^* b (a \cup b)^*$

Example 1.68

Convert the following three-state DFA into a RE



① Convert to a GNFA:



Notes:

- We are omitting the $\emptyset$ edges but they still exist

② We can remove any state, let's remove state $\underline{1}$:

(2.1)
- $q_i = q_{start}$, $q_{rip} = 1$, $q_j = 2$
- $R_1 = \delta(q_i, q_{rip}) = \delta(q_{start}, 1) = \varepsilon$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = \emptyset$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1, 2) = a$
- $R_4 = \delta(q_i, q_j) = \delta(q_{start}, 2) = \emptyset$

$(R_1)(R_2)^*(R_3) \cup (R_4) = (\varepsilon)(\emptyset)^*(a) \cup (\emptyset) = a$

(2.2)
- $q_i = q_{start}$, $q_{rip} = 1$, $q_j = 3$
- $R_1 = \delta(q_i, q_{rip}) = \delta(q_{start}, 1) = \varepsilon$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = \emptyset$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1, 3) = b$
- $R_4 = \delta(q_i, q_j) = \delta(q_{start}, 3) = \emptyset$

$(R_1)(R_2)^*(R_3) \cup (R_4) = (\varepsilon)(\emptyset)^*(b) \cup (\emptyset) = b$

(2.3)
- $q_i = 2, q_{rip} = 1, q_j = 3$
- $R_1 = \delta(q_i, q_{rip}) = \delta(2,1) = a$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = \phi$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1,3) = b$
- $R_4 = \delta(q_i, q_j) = \delta(2,3) = \phi$

$(R_1)(R_2)^*(R_3) \cup (R_4) =$
$(a)(\phi)^*(b) \cup (\phi) =$
ab

(2.4)
- $q_i = 3, q_{rip} = 1, q_j = 2$
- $R_1 = \delta(q_i, q_{rip}) = \delta(3,1) = b$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = \phi$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1,2) = a$
- $R_4 = \delta(q_i, q_j) = \delta(3,2) = a$

$(R_1)(R_2)^*(R_3) \cup (R_4) =$
$(b)(\phi)^*(a) \cup a =$
$ba \cup a$

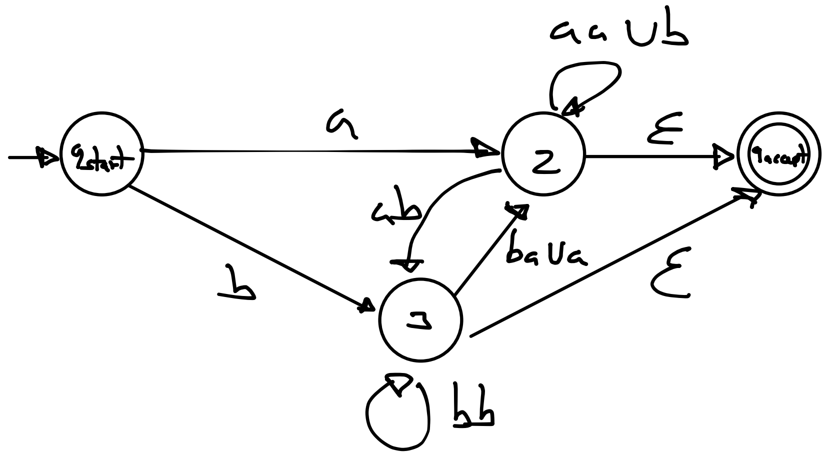
(2.5)
- $q_i = 2, q_{rip} = 1, q_j = 2$
- $R_1 = \delta(q_i, q_{rip}) = \delta(2,1) = a$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = \phi$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1,2) = a$
- $R_4 = \delta(q_i, q_j) = \delta(2,2) = b$

$(R_1)(R_2)^*(R_3) \cup (R_4) =$
$(a)(\phi)^*(a) \cup (b) =$
$aa \cup b$

(2.5)
- $q_i = 3, q_{rip} = 1, q_j = 3$
- $R_1 = \delta(q_i, q_{rip}) = \delta(3,1) = b$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(1,1) = \phi$
- $R_3 = \delta(q_{rip}, q_j) = \delta(1,3) = b$
- $R_4 = \delta(q_i, q_j) = \delta(3,3) = \phi$

$(R_1)(R_2)^*(R_3) \cup (R_4) =$
$(b)(\phi)^*(b) \cup \phi =$
bb

$\therefore$



③ We can remove any state, let's remove state 2:

(3.1)
- $q_i = q_{start}$, $q_{rip} = 2$, $q_j = q_{accept}$
- $R_1 = \delta(q_i, q_{rip}) = \delta(q_{start}, 2) = a$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(2,2) = aa \cup b$
- $R_3 = \delta(q_{rip}, q_j) = \delta(2, q_{accept}) = \varepsilon$
- $R_4 = \delta(q_i, q_j) = \delta(q_{start}, q_{accept}) = \emptyset$

$$(R_1)(R_2)^*(R_3) \cup (R_4) = (a)(aa \cup b)^*(\varepsilon) \cup (\emptyset) = a(aa \cup b)^*$$

(3.2)
- $q_i = q_{start}$, $q_{rip} = 2$, $q_j = 3$
- $R_1 = \delta(q_i, q_{rip}) = \delta(q_{start}, 2) = a$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(2,2) = aa \cup b$
- $R_3 = \delta(q_{rip}, q_j) = \delta(2, 3) = ab$
- $R_4 = \delta(q_i, q_j) = \delta(q_{start}, 3) = b$

$$(R_1)(R_2)^*(R_3) \cup (R_4) = (a)(aa \cup b)^*(ab) \cup (b) = a(aa \cup b)^* ab \cup b$$

(3.3)
- $q_i = 3$, $q_{rip} = 2$, $q_j = q_{accept}$
- $R_1 = \delta(q_i, q_{rip}) = \delta(3, 2) = ba \cup a$
- $R_2 = \delta(q_{rip}, q_{rip}) = \delta(2,2) = aa \cup b$
- $R_3 = \delta(q_{rip}, q_j) = \delta(2, q_{accept}) = \varepsilon$
- $R_4 = \delta(q_i, q_j) = \delta(3, q_{accept}) = \varepsilon$

$$(R_1)(R_2)^*(R_3) \cup (R_4) = (ba \cup a)(aa \cup b)^*(\varepsilon) \cup (\varepsilon) = (ba \cup a)(aa \cup b)^* \cup \varepsilon$$

3.4 – $q_i = 3$, $q_{rip} = 2$, $q_j = 3$

– $R_1 = \delta(q_i, q_{rip}) = \delta(3,2) = ba \cup a$

– $R_2 = \delta(q_{rip}, q_{rip}) = \delta(2,2) = aa \cup b$

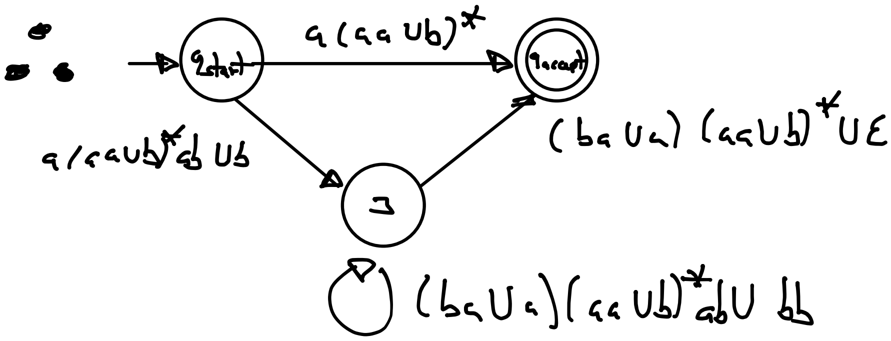
– $R_3 = \delta(q_{rip}, q_j) = \delta(2,3) = ab$

– $R_4 = \delta(q_i, q_j) = \delta(3,3) = bb$

$(R_1)(R_2)^*(R_3) \cup (R_4) =$

$= (ba \cup a)(aa \cup b)^*(ab) \cup (bb)$

$= (ba \cup a)(aa \cup b)^* ab \cup bb$



4 We can remove any state, let's remove state 3:

4.1 – $q_i = q_{start}$, $q_{rip} = 3$, $q_j = q_{accept}$

– $R_1 = \delta(q_i, q_{rip}) = \delta(q_{start}, 3) = a(aa \cup b)^* ab \cup b$

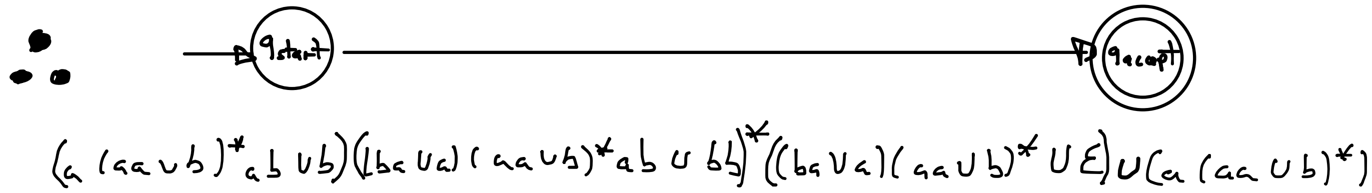
– $R_2 = \delta(q_{rip}, q_{rip}) = \delta(3,3) = (ba \cup a)(aa \cup b)^* ab \cup bb$

– $R_3 = \delta(q_{rip}, q_j) = \delta(3, q_{accept}) = (ba \cup a)(aa \cup b)^* \cup \mathcal{E}$

– $R_4 = \delta(q_i, q_j) = \delta(q_{start}, q_{accept}) = a(aa \cup b)^*$

– $(R_1)(R_2)^*(R_3) \cup (R_4) =$

$(a(aa \cup b)^* ab \cup b)((ba \cup a)(aa \cup b)^* ab \cup bb)^*((ba \cup a)(aa \cup b)^* \cup \mathcal{E}) \cup (a(aa \cup b)^*)$



1.4 NonRegular Languages

Q:	Are there any limitations to finite automata?

↳ Yes!!! Certain languages cannot be recognized

Consider the language $B = \{0^n 1^n \mid n \geq 0\}$:

① Attempt to find a DFA to accept B

② The machine needs to remember how many 0's have been seen.

③ Because the number of 0's isn't limited the machine will need to keep track of an unlimited number of possibilities (unlimited number of states)

④ ∴ B is nonregular

Q:	So, how can we prove that a language is not regular?

↳ Through a teorem called the pumping lemma

Theorem 1.70

Pumping Lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p then s may be divided into three pieces $s = \underbrace{x}_{\text{can be } \varepsilon} y \underbrace{z}_{\text{can be } \varepsilon}$ satisfying the following conditions:

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

Notes:

- $y^0 = \varepsilon \Rightarrow xy^0z = xz \in A$
- $x = \varepsilon$ or $z = \varepsilon \Rightarrow xyz = y$, $|y| > 0$

(In Portuguese: Lema do Bombeamento)

Q: Ok, this is all very pretty but what does it mean in practice?

Informally: All sufficiently long strings in a RL may be pumped, i.e. have a middle section of the string repeated an arbitrary number of times, to produce a new string that is also part of the language.

Proof Idea:

① Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A

② Let the pumping length p to be the number of states of M. This is an assumption.

③ We show that any string s in A of length at least p may be broken into three pieces xyz satisfying the three conditions.

④ What if no strings in A are of length at least p? Obviously, the three conditions hold for all strings of length at least p if there aren't any such strings (i.e. the conditions remain true but no such strings (of length at least p) exist)

⑤ If s in A has length at least p, consider the sequence of states that M goes through when computing with input s. It starts with q_1 the start state, then goes to, say q_3, then say q_{20}, then q_9, and so on, until it reaches the end of s in state q_{13}. With s in A, we know that M accepts s, so q_{13} is a final state

⑥ Let $n = |s|$, the sequence of states $q_1, q_3, q_{20}, q_9, \dots, q_{13}$ has length $\underline{n+1}$. Because $\underline{n}$ is at least $\underline{p}$, we know that $n+1 > p$ (recall that in step ② we defined $\underline{p}$ to be the number of states of M).

⑦ Notice what this means: The number of states in the sequence is $\underline{n+1}$ which is larger than $\underline{p}$ (the number of states of M). This means that one state must have been repeated when processing $\underline{s}$, i.e.:

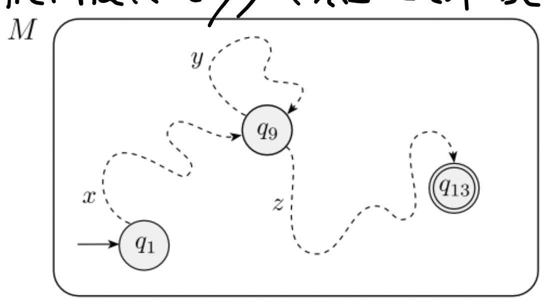
$$
\begin{array}{ccccccccccccccc}
s = & & s_1 & & s_2 & & s_3 & & s_4 & & s_5 & & s_6 & & \cdots & & s_n & \\
 & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 & q_1 & & q_3 & & q_{20} & & q_9 & & q_{17} & & q_9 & & q_6 & & q_{35} & & q_{13}
\end{array}
$$

⑧ We can now divide $\underline{s}$:

$$
s = \overbrace{s_1\, s_2\, s_3}^{x}\ \overbrace{s_4\, s_5}^{y}\ \overbrace{s_6 \ \cdots \ s_n}^{z}
$$

$$
\begin{array}{ccccccccccccccc}
s = & & s_1 & & s_2 & & s_3 & & s_4 & & s_5 & & s_6 & & \cdots & & s_n & \\
 & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 & q_1 & & q_3 & & q_{20} & & q_9 & & q_{17} & & q_9 & & q_6 & & q_{35} & & q_{13}
\end{array}
$$

Alternatively, this can be perceived as:



- $\underline{x}$ takes M from q_1 to q_9
- $\underline{y}$ takes M from q_9 to q_9
- $\underline{z}$ takes M from q_9 to q_{13}

⑨ Let's see why this division of $\underline{s}$ satisfies the three conditions.

Condition 1: Suppose that we run $\underline{M}$ on $\underline{xyyz}$. We know that $\underline{x}$ takes $\underline{M}$ from $\underline{q_1}$ to $\underline{q_9}$, and then the first $\underline{y}$ takes it from q_9 back to q_9, as does the second $\underline{y}$, and then $\underline{z}$ takes it to q_{13}. With q_{13} being a final state the string is recognized.

The same is valid for xy^iz, $i>0$. For $i=0$, $xy^iz = xz$ which is accepted for similar reasons

Condition 2: We can see that $|y|>0$, as it was the part of $\underline{s}$ that occurred between two $\neq$ occurrences of state q_9

Condition 3: Because $\overset{\#\text{states in the sequence}}{n+1} > \overset{\#\text{number of states of } M}{p}$ we know that there must be a repetition of a state in the sequence. Let q_9 be the first repetition in the sequence. Because $\underline{n}$ is at least $\underline{p}$ (i.e. $n \geq p$) then the $\#statesInTheSequence = n+1 = p+1$. This means that the first $p+1$ states of the sequence must contain a repetition, i.e.:

$$s = \overbrace{s_1\ s_2\ s_3}^{x}\ \overbrace{s_4\ s_5}^{y}\ \overbrace{s_6\ \ldots\ s_n}^{z}$$

$$\underbrace{q_1\ \ q_3\ \ q_{20}\ \ q_9\ \ q_{17}\ \ q_9\ \ q_6}_{p+1 \text{ states}}\ \ \ldots\ \ q_{35}\ \ q_{13}$$

This implies that $p+1$ states are needed to recognize strings $\underline{x}$ and $\underline{y}$. Which implies that $|xy| \leq p$ (one additional state is needed to accept xy).

Proof:

① Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and p be the number of states of M

② Let $s = s_1 s_2 \dots s_n$ be a string in A of length n, where $n \geq p$

③ Let $r = r_1, \dots, r_{n+1}$ be the sequence of states that M enters while processing s, i.e. $r_{i+1} = \delta(r_i, s_i)$ $\forall i \in [1, n]$

Because n is at least p, then $n+1 > p$. This means that the sequence of states (which has length $n+1$) is larger than the number of states p in M. Accordingly, this can only mean that states are being repeated.

④ Among the first $p+1$ elements in the sequence, two must be the same state. We call the first of these r_j and the second r_l. Because r_l occurs among the first $p+1$ places in a sequence starting at r_1, we have $l \leq p+1$

⑤ Let $x = s_1 \dots s_{j-1}$, $y = s_j \dots s_{l-1}$ and $z = s_l \dots s_n$. This means ① that $\boxed{j \neq l}$ so $|y| > 0$ and ② $l \leq p+1$ so $|xy| \leq p$.

⑥ x takes M from r_1 to r_j
y takes M from r_j to r_j
z takes M from r_j to r_{n+1} (final state)

$\therefore$ M accepts xy^iz $(i \geq 0)$

∎

Q: So, how can we use the pumping lemma to prove that a language is not regular?

1. Assume that B is regular (contradiction)

2. Use pumping lemma to garantee the existence of a pumping length p such that all strings of length $\geq p$ can be pumped

3. Find a string s in B that has length p or greater but that cannot be pumped.

4. Demonstrate that s cannot be pumped by considering all ways of dividing s into x, y and z (considering condition 3 if necessary) and, for each such division, finding a value i where $xy^iz \notin B$.

5. The existence of s contradicts the pumping lemma if B were regular. Hence B cannot be regular.

Notes:

Finding s may require creative thinking

Example 1.73

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Use the pumping lemma to prove that B is not regular.

[Resolution:]

1. Assume B is regular (contradiction)
2. Let p be the pumping length
3. Choose $s = 0^p 1^p$
4. $s \in B$ and $|s| \geq p$, then the P.L. guarantees that s can be broken into xyz such that $xy^iz \in B$ $\forall i \geq 0$
5. Consider the following cases where this is impossible:

5.1 String y consists of only 0's:

Then if we pump y (i.e. xy^iz) the final string will have more 0's than 1's and $xy^iz \notin B$ (contradiction)

5.2 String y consists of only 1's:

Then if we pump y (i.e. xy^iz) the final string will have more 1's than 1's and $xy^iz \notin B$ (contradiction)

5.3 String y consists of 0's and 1's:

Then if we pump y (i.e. xy^iz) the final string may have the same number of 0's and 1's but they will be out of order $\Rightarrow xy^iz \notin B$ (contradiction)

Example 1.74

Let $C = \{w \mid w$ has an equal number of 0s and 1s$\}$. Use the pumping lemma to prove that C is not regular.

[Resolution:]

① Assume C is regular (contradiction)

② Let p be the pumping length

③ Choose $s = 0^p 1^p$

④ $s \in B$ and $|s| \geq p$, then the P.L. guarantees that s can be broken into xyz such that $xy^iz \in B$ $\forall i \geq 0$

⑤ If $s = xyz$ with $x = \varepsilon$ and $z = \varepsilon$, then $s = y \Rightarrow y = 0^p 1^p$ which means that $xy^iz = y^i = (0^p 1^p)^i \in C$ (i.e. it seems that s can be pumped)

⑥ We need an alternative:

6.1 Condition 3 of the P.L.: $|xy| \leq p$

6.2 This means that if $s = xyz = \underbrace{00\ldots0}_{p} 1^p$ and $|xy| \leq \boxed{p}$, then string y must consist only of 0's.

6.3 This means that xy^iz will produce string with a greater number of 0's than 1's, i.e. $xy^iz \notin C$ (contradiction)

Example 1.75

Let $F = \{ww \mid w \in \{0,1\}^*\}$. Use the pumping lemma to prove that F is not regular.

[Resolution:]

① Assume F is regular (contradiction)

② Let p be the pumping length

③ Choose $s = 0^p 1 \; 0^p 1$

④ $s \in B$ and $|s| \geq p$, then the P.L. gurantees that s can be broken into xyz such that $xy^iz \in C \; \forall i \geq 0$

⑤ Similarly to Example 1.74 if $x = \varepsilon$ and $z = \varepsilon$ then $s = xyz = y$ which would imply that $xy^iz = y^i = (0^p 1 \; 0^p 1)^i$ so it would seem that s could be pumped.

⑥ We need an alternative:

6.1 Condition 3 of the P.L.: $|xy| \leq p$

6.2 If $s = 0^p 1 \; 0^p 1 = \overbrace{00 \ldots 0}^{p} 1 \; 0^p 1$ and $|xy| \leq p$ then string y must consist only of 0's. This means that xy^iz would produce strings where the first part of the string (the first w of ww) would not match the 2nd w.
i.e. $xy^iz \notin F$ (contradiction)

Example 1.76

Let $D = \{1^{n^2} \mid n \geq 0\}$, i.e. D contains all strings of 1's whose length is a perfect square. Use the pumping lemma to prove that D is not regular.

[Resolution:]

① Assume D is regular (contradiction)

② Let p be the pumping length

③ Choose $s = 1^{p^2}$

④ $s \in B$ and $|s| \geq p$, then the P.L. guarantees that s can be broken into xyz such that $xy^iz \in D \ \forall i \geq 0$

⑤ To solve this one we need to think about the sequence of perfect squares:

$$0, 1, 4, 9, 16, 25, 36, 49, \ldots$$

Note the growing gap between consecutive members of this sequence. Large members of this sequence cannot be near each other.

⑥ Consider the two strings xyz and xy^2z. These strings differ from each other by a single repetition of y. Consequently, their lengths differ by a length y.

⑦ Condition 3 of P.L.: $|xy| \leq p$. If $x = \varepsilon$ this means that at most y has length p, i.e. $|y| \leq p$.

⑧ Since $s = 1^{p^2} = \overbrace{11\ldots1}^{p^2}$, i.e. $|s| = p^2 \Leftrightarrow |xyz| = p^2$

9 Therefore, if we increase the length by y and given that $|y| \leq p$ then $|xy^2z| \leq p^2 + p$

10 But $p^2 + p < p^2 + 2p + 1 = (p+1)^2$

Q:	Why is this fact important?

If we calculate the next perfecte element to p, i.e. $s' = 1^{(p+1)^2} = \overbrace{11\ldots1}^{(p+1)^2}$, i.e. we can see that the length of s' should be $(p+1)^2$. However, when we pumped y in step 9 we obtained that the next element should have length $\leq p^2+p$ (contradiction)