Goal

Understanding Computation

Goal

Understanding Computation

- Fundamental capabilities of computers.
- Fundamental limitations of computers.

Goal

Understanding Computation

- Fundamental capabilities of computers.
- Fundamental limitations of computers.

There are different ways to approach the question:

- automata
- computability
- complexity

But why? I am a practitioner!

- Design a new programming language
  - Grammars
- Pattern matching
  - Finite automata
- Real-time/fast computation
  - Complexity theory

But why? I am a practitioner!

- Design a new programming language
  - Grammars
- Pattern matching
  - Finite automata
- Real-time/fast computation
  - Complexity theory

Technology is quickly outdated, but theory remains the same.

What makes some problems computationally hard and others easy?

Which problems are hard and which ones are easy

easy Sort a list of integers in ascending order

hard Schedule the classes of the university satisfying reasonable constraints

What makes some problems computationally hard and others easy?

- Fast answer
  - We don't really know!
- Real answer
  - We have a lot of insight! We classify problems
    - P
    - NP
    - NP-complete
    - ...

Again, why do I care?

Many applications rely on complexity theory. For example, most cryptosystems are based on the assumption that a problem is hard to solve, but easy to confirm a solution.

Again, why do I care?

Many applications rely on complexity theory. For example, most cryptosystems are based on the assumption that a problem is hard to solve, but easy to confirm a solution.
Integer Factorization:

- Factorize: 62615533
- Multiply : $7907 \cdot 7919$

What if my problem is actually hard?

- Understanding the root of difficulty
  - Simplify the problem
- If the exact solution is hard, but not really needed
  - Find an approximate solution
- Is it always hard, or just in worst case?
  - It may be easy almost always, so it is practical.
- Other models of computation
  - For example randomized algorithms

Computability Theory

Is it even possible to solve it?

- Some problems are not solvable by computers
  - Kurt Gödel, Alan Turing, Alonzo Church
- Complexity Theory
  - Easy vs Hard
- Computability Theory
  - Solvable vs Unsolvable

Theory of Automata

Mathematical models of computation

Applications of automata:

- Finite Automaton
  - text processing, compilers, hardware design
- Context-Free Grammars
  - programming languages, artificial intelligence

Automata

Strings and Languages

Definition (Alphabet)

An alphabet is a non-empty finite set.

Definition (String)

A string over an alphabet is a finite sequence of symbols from the alphabet.

Example

- $\Sigma_1 = \{0, 1\}$
  - 0101101
  - 01111110101
  - 01112110101
- $\Sigma_2 = \{a, b, c, d, f, t, o, m, n, u, x, z\}$
  - *automaton*
  - *automata*
  - *aabaabcaduzfo*
  - *merhaba*

Strings and Languages

Let Σ be an alphabet.

Definition (Length)

Given a string w over Σ, the **length** of w, denoted by $|w|$, is the number of symbols it contains.

Definition (Empty String)

The string of length zero, denoted by ε, is called the **empty string**.

Definition (Reverse)

Given a string $w = w_1 w_2 \dots w_n$ over Σ, the **reverse** of w, denoted by w^R, is the string $w_n w_{n-1} \dots w_2 w_1$.

Definition (Substring)

Given two strings $w = w_1 w_2 \dots w_n$ and $z = z_1 z_2 \dots z_m$ over Σ, z is a **substring** of w iff $w = w_1 \dots w_i z_1 z_2 \dots z_m w_{i+m+1} \dots w_n$.

Strings and Languages

Let $\Sigma = \{0, 1, 2, 3\}$ be an alphabet.

Example

- 0101101

length 7
reverse 1011010

  - Substrings
    - 110
    - 1011
    - 01010

- 12213

length 5
reverse 31221

  - Substrings
    - 1
    - 213
    - 23

Operations on Strings

Definition (Concatenation)

Given two strings $w = w_1 w_2 \dots w_n$ and $z = z_1 z_2 \dots z_m$ over Σ, the concatenation of w and z, denoted by wz, is the string

$$wz = w_1 \dots w_n z_1 z_2 \dots z_m.$$

Concatenation of w with itself k times, is denoted by w^k.

Example

- $\Sigma_1 = \{0, 1, 2\}$, $w = 010$, $z = 111$
  - $wz = 010111$
  - $z^2 = 111111$
  - $w^3 z^2 w = 010010010111111010$

Definition (Prefix)

Given two strings w and z over Σ, we say that w is a prefix of z if there exists a string x such that $wx = z$. Proper prefix if $w \neq z$.

Lexicographic ordering

Definition (lex order)

The lexicographic order of strings is the same as the familiar dictionary order.

Definition (shortlex order)

The lexicographic order of strings is the same as the familiar dictionary order.

- shorter strings precede longer strings
- strings of equal length are sorted with lex order.

Example

- $\Sigma_1 = \{0, 1\}$
  - $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

Language

Definition (Language)

A language is a set of strings.

Definition (Prefix-free Language)

A language is prefix-free if no member is a proper prefix of another member.

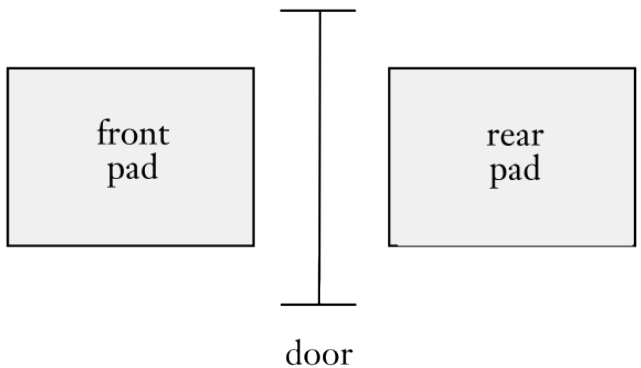
Finite State Machine

a.k.a finite automaton

Finite Automata

- Good model for computers with limited amount of memory
- Simple but useful
- In the core of electromechanical devices

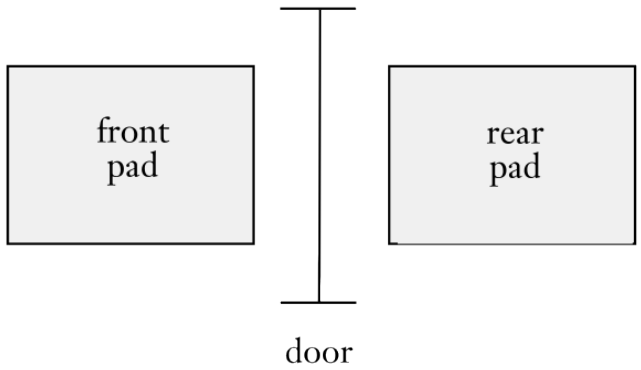
Example



- Two states: OPEN and CLOSED
- Four Input Conditions: FRONT, REAR, BOTH, NEITHER

Finite Automata

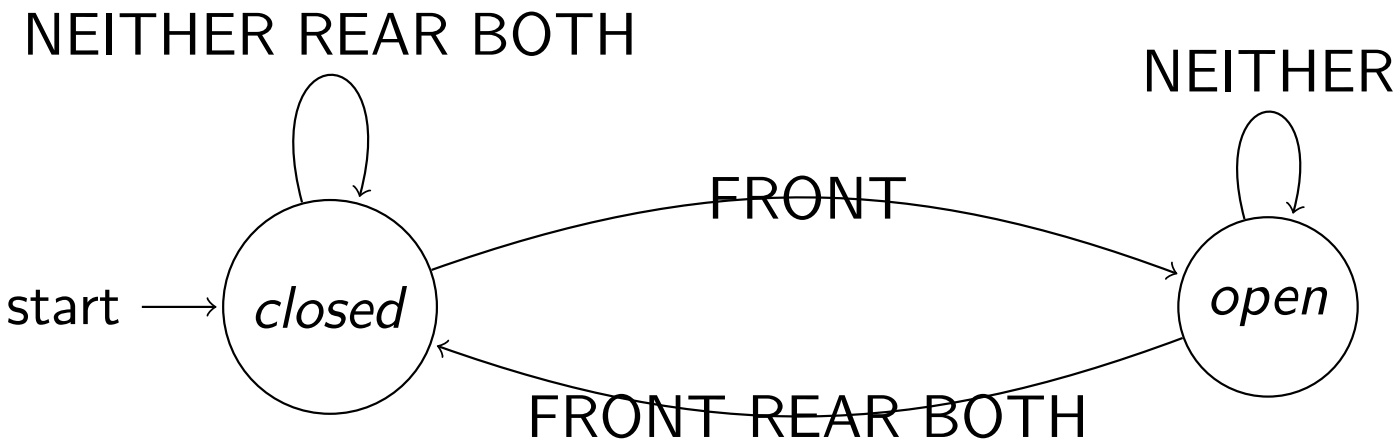
Example



- When CLOSED:
  - NEITHER or REAR or BOTH: CLOSED
  - FRONT: OPEN
- When OPEN:
  - FRONT or REAR or BOTH: OPEN
  - NEITHER: CLOSED

Finite Automata

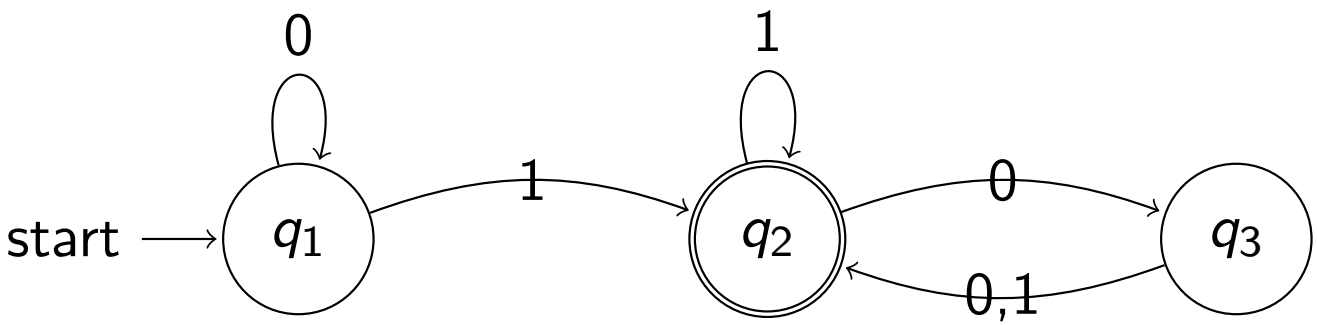
Example



- When CLOSED:
  - NEITHER or REAR or BOTH: CLOSED
  - FRONT: OPEN
- When OPEN:
  - FRONT or REAR or BOTH: OPEN
  - NEITHER: CLOSED

Finite Automata

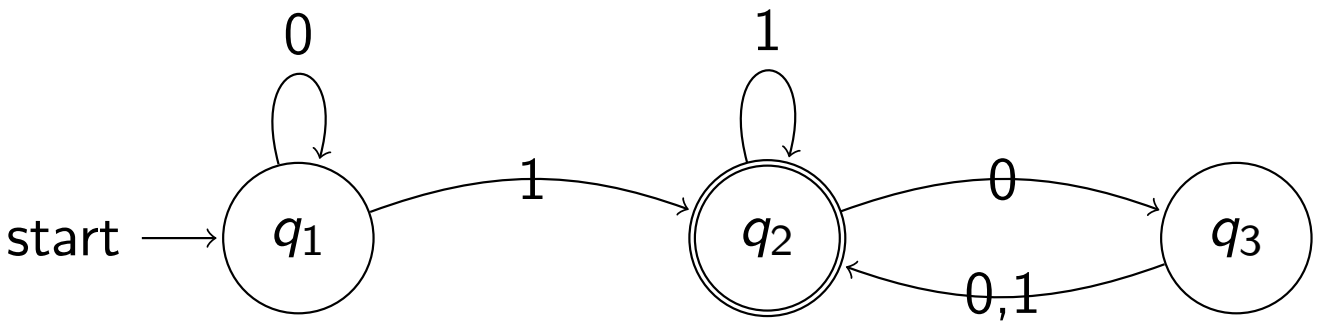
Example



- Three states $q_1.q_2.q_3$.
- q_1 is the **start state**
- q_2 is the accept state
- The arrows are called **transitions**
- The output is either **accept** or **reject**.

Finite Automata

Example



- Start with the start sate
- Receive symbols from input string
  - Traverse the link labeled with the received symbol
  - Transition to another state
- When the input string is exhausted
  - accept: if the final state is an accept state
  - reject: if the final state is not an accept state

Finite Automata

Definition

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

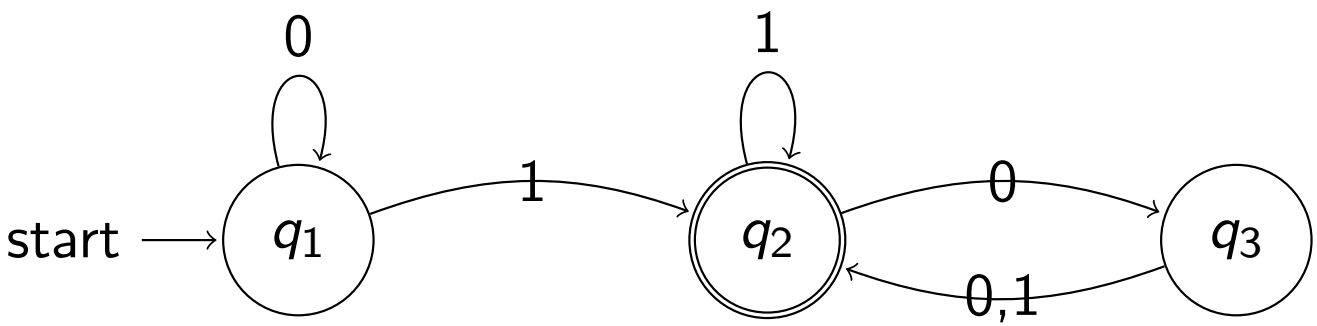
- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \to Q$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**

Finite Automata

Definition

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

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \to Q$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**



Finite Automata

Definition

Given a Finite State Machine M, let A be the set of all strings that M accepts. Then we say that:

- A is the **language of machine** M, denoted by $L(M) = A$.
- M **recognizes** A.

Notes:

- An FSM may accept several strings, but always recognizes only one language.
- If an FSM does not accept any string, it recognizes the empty language $\emptyset$.

Finite Automata

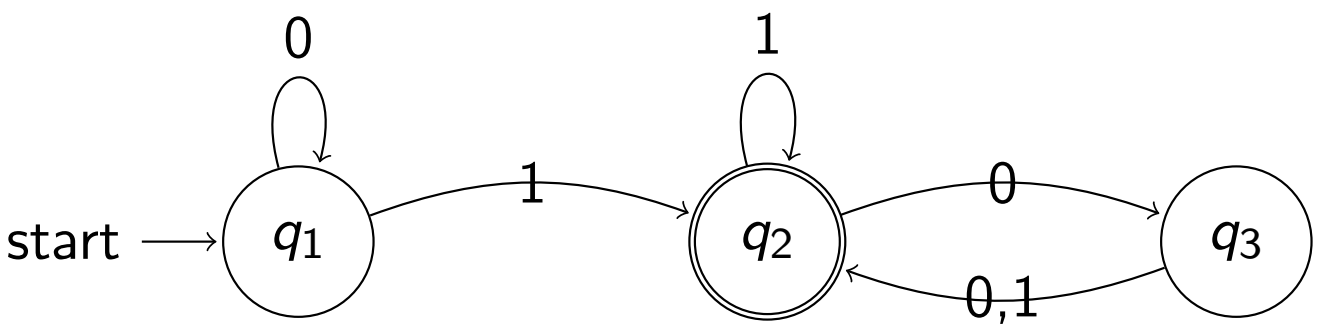
Definition

Given a Finite State Machine M, let A be the set of all strings that M accepts. Then we say that:

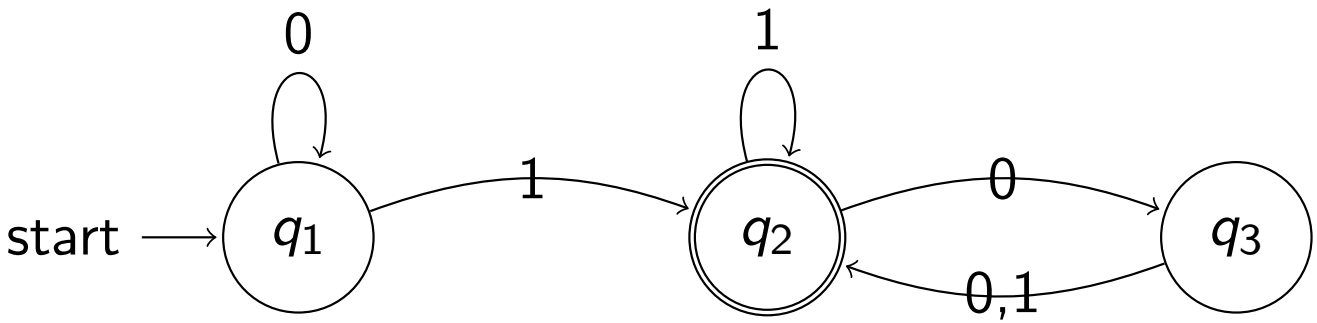
- A is the **language of machine** M, denoted by $L(M) = A$.
- M **recognizes** A.

Notes:

- An FSM may accept several strings, but always recognizes only one language.
- If an FSM does not accept any string, it recognizes the empty language $\emptyset$.



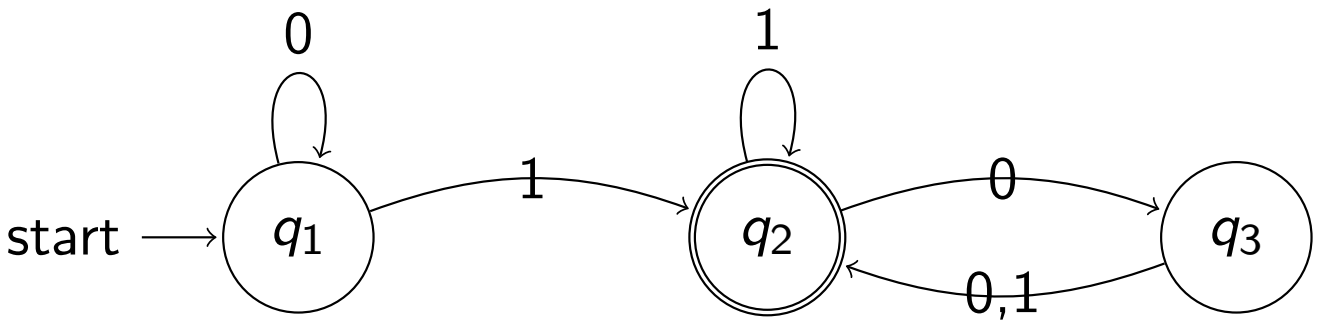
Finite Automata



Example

What is the language recognized by this machine?

Finite Automata



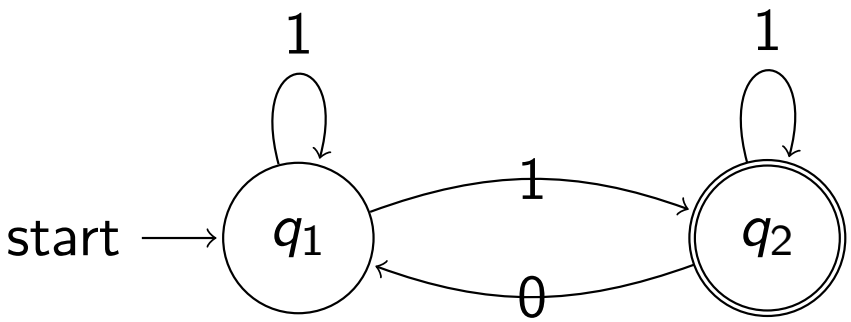
Example

What is the language recognized by this machine?

$$A = \{w : w \text{ contains at least one 1}$$

$$\text{and an even number of 0 following the last 1}\}$$

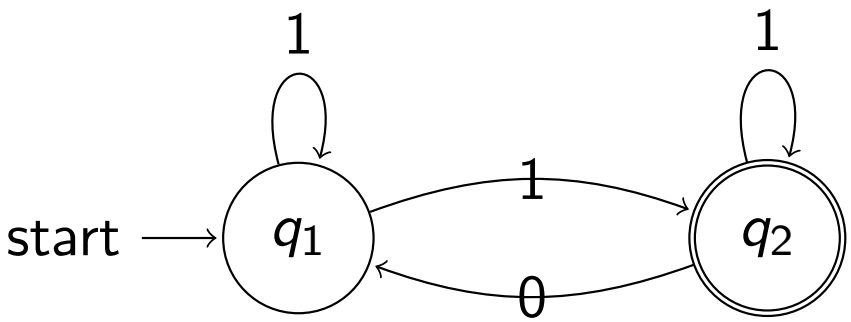
Finite Automata



Example

What is the language recognized by this machine?

Finite Automata

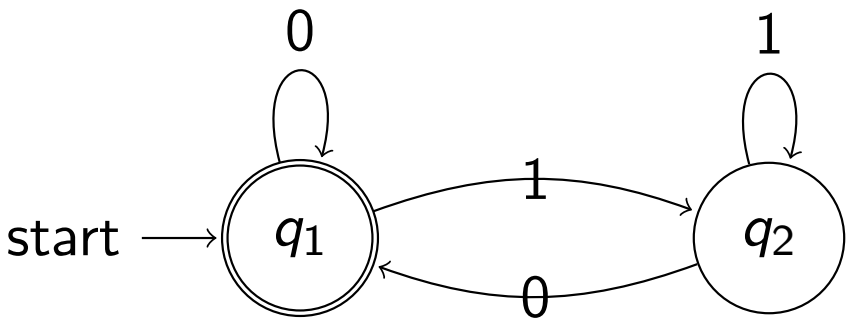


Example

What is the language recognized by this machine?

$$A = \{w : w \text{ ends with } 1\}$$

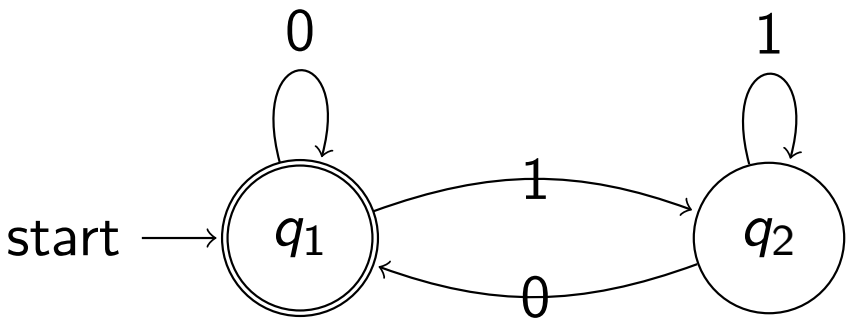
Finite Automata



Example

What is the language recognized by this machine?

Finite Automata



Example

What is the language recognized by this machine?

$$A = \{w : w \text{ ends with } 0 \text{ or } w = \varepsilon\}$$

Regular Language

Definition

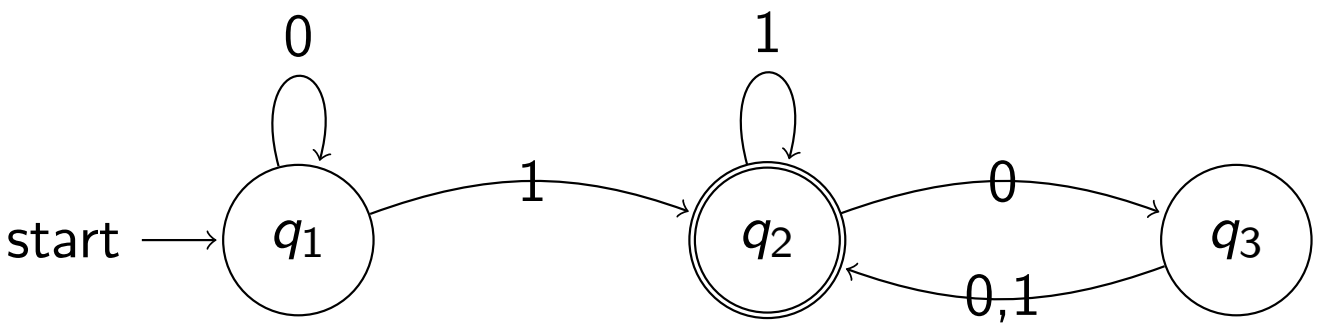
A language is called a **regular language** if some finite automaton recognizes it.

Finite Automata

Definition

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

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \to Q$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**



How to Design a Finite Automaton

We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?

How to Design a Finite Automaton

We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?
  - I need to remember if the number of 1's is even or odd.

How to Design a Finite Automaton

We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?
  - I need to remember if the number of 1's is even or odd.
- What are the possible states?

How to Design a Finite Automaton

We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?
  - I need to remember if the number of 1's is even or odd.
- What are the possible states?
  - Number of 1's till now is odd or not.

How to Design a Finite Automaton

We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?
  - I need to remember if the number of 1's is even or odd.
- What are the possible states?
  - Number of 1's till now is odd or not.
- What are the transitions?

How to Design a Finite Automaton

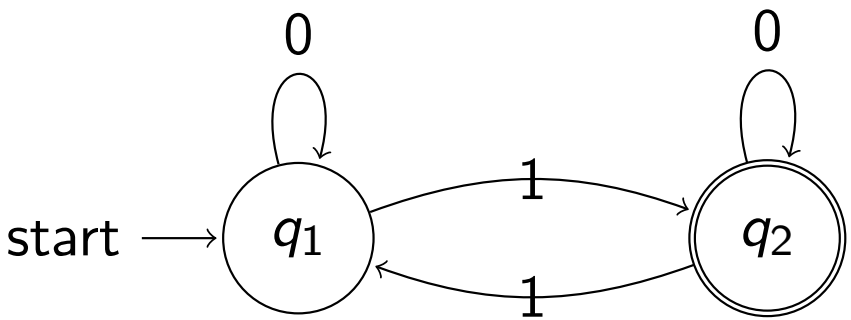
We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?
  - I need to remember if the number of 1's is even or odd.
- What are the possible states?
  - Number of 1's till now is odd or not.
- What are the transitions?
  - If I read 0: Nothing changes.
  - If I read 1: Transition to the other state.

How to Design a Finite Automaton

We are given the language consisting of all strings with an odd number of 1's over the alphabet $\Sigma = \{0, 1\}$.

- Do I need to remember the whole string?
  - I need to remember if the number of 1's is even or odd.
- What are the possible states?
  - Number of 1's till now is odd or not.
- What are the transitions?
  - If I read 0: Nothing changes.
  - If I read 1: Transition to the other state.



How to Design a Finite Automaton

- What do I need to remember?
  - Remembering the whole string is not possible (limited memory).
  - Specify what is needed to decide if the string is in the language.
- Represent this information as a finite set of possibilities and assign a state for each possibility,
- Assign transitions thinking how to go from one possibility to another.
- Set the start state to the possibility when no symbol is received.
- Set the accept states considering the possibilities where the string until now is accepted.

How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?

How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?
  - No symbol of the pattern yet
  - I have read 0
  - I have read 00
  - I have read the whole pattern 001

How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?
  - No symbol of the pattern yet
  - I have read 0
  - I have read 00
  - I have read the whole pattern 001
- What are the transitions?

How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?
  - No symbol of the pattern yet
  - I have read 0
  - I have read 00
  - I have read the whole pattern 001
- What are the transitions?

	q	q_0	q_{00}	q_{001}
0	q_0	q_{00}	q_{00}	q_{001}
1	q	q	q_{001}	q_{001}

How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?
  - No symbol of the pattern yet
  - I have read 0
  - I have read 00
  - I have read the whole pattern 001
- What are the transitions?

	q	q_0	q_{00}	q_{001}
0	q_0	q_{00}	q_{00}	q_{001}
1	q	q	q_{001}	q_{001}

- Start state: q

How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?
  - No symbol of the pattern yet
  - I have read 0
  - I have read 00
  - I have read the whole pattern 001
- What are the transitions?

	q	q_0	q_{00}	q_{001}
0	q_0	q_{00}	q_{00}	q_{001}
1	q	q	q_{001}	q_{001}

- Start state: q
- Accept states: $\{q_{001}\}$

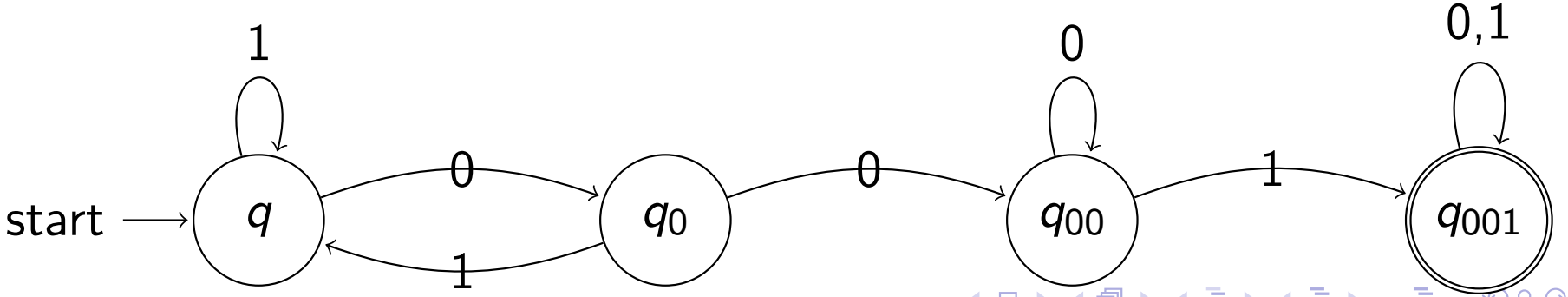
How to Design a Finite Automaton

Recognize the language consisting of all strings, over the alphabet $\Sigma = \{0, 1\}$, that contain 001 as substring.

- What are the possibilities I have?
  - No symbol of the pattern yet
  - I have read 0
  - I have read 00
  - I have read the whole pattern 001
- What are the transitions?

	q	q_0	q_{00}	q_{001}
0	q_0	q_{00}	q_{00}	q_{001}
1	q	q	q_{001}	q_{001}

- Start state: q
- Accept states: $\{q_{001}\}$



Regular Operations

(New) Regular Languages

Definition

A language is called regular iff it is recognized by a finite automaton.

Question

Given some regular languages, how can we produce new ones?

(New) Regular Languages

Definition

A language is called regular iff it is recognized by a finite automaton.

Question

Given some regular languages, how can we produce new ones?

We want to be able to operate on regular languages in way that we obtain new regular languages. Such operations are called regular operations.

Question

What are they?

Regular Operations

Let A and B be languages over some alphabet Σ.

Definition (Union)

$$A \cup B = \{s : s \in A \text{ or } s \in B\}$$

Definition (Concatenation)

$$A \circ B = \{st : s \in A \text{ and } t \in B\}$$

Definition (Star)

$$A^* = \{x_1 x_2 \cdots x_k : k \in \{0, 1, 2, 3, \ldots\} \text{ and } x_i \in A\}$$

Regular Operations

Let $A = \{foo, bar\}$ and $B = \{hello, world\}$ be languages over the English alphabet.

Definition (Union)

$$A \cup B = \{foo, bar, hello, world\}$$

Definition (Concatenation)

$$A \circ B = \{foohello, fooworld, barhello, barworld\}$$

Definition (Star)

$$A^* = \{\epsilon, foo, bar, foofoo, foobar, barfoo, barbar, foofoofoo, \ldots\}$$

Regular Operations

Question

Is $A = \{0\}$ *a regular language over* $\Sigma = \{0, 1\}$*? What about* $A = \{1\}$

Question

What is A^**?*

Regular Operations

Question

Is $A = \{0\}$ *a regular language over* $\Sigma = \{0, 1\}$*? What about* $A = \{1\}$

Question

What is A^**?*

Answer

$$A^* = \{\epsilon, 0, 00, 000, 0000, \ldots\}$$

Question

What is $(A \cup B)^*$*?*

Regular Operations

Question

Is $A = \{0\}$ *a regular language over* $\Sigma = \{0, 1\}$*? What about* $A = \{1\}$

Question

What is A^**?*

Answer

$$A^* = \{\epsilon, 0, 00, 000, 0000, \ldots\}$$

Question

What is $(A \cup B)^*$*?*

Answer

All binary sequences.

Closure

Definition

A collection is **closed** under an operation if applying this operation to any member of the collection we obtain again a member of the collection.

Example

The set $\mathbb{N}$

- is closed under multiplication or addition,
- but it is NOT closed under division.

Closure Theorems

Theorem

The class of regular languages is closed under union.

Theorem

The class of regular languages is closed under concatenation.

Theorem

The class of regular languages is closed under star.

Non-deterministic Finite Automata

Non-determinism

Definition (Deterministic)

When the machine is in a given state and reads an input symbol, there is exactly one next state (after transition). The next state is fully determined.

Definition (Non-deterministic)

When the machine is in a given state and reads an input symbol, there may be several choices for the next state (after transition). The next state is NOT fully determined.

very Deterministic Finite Automaton (DFA) is a Non-deterministic Finite Automaton (NFA).

Non-determinism

Definition (Deterministic)

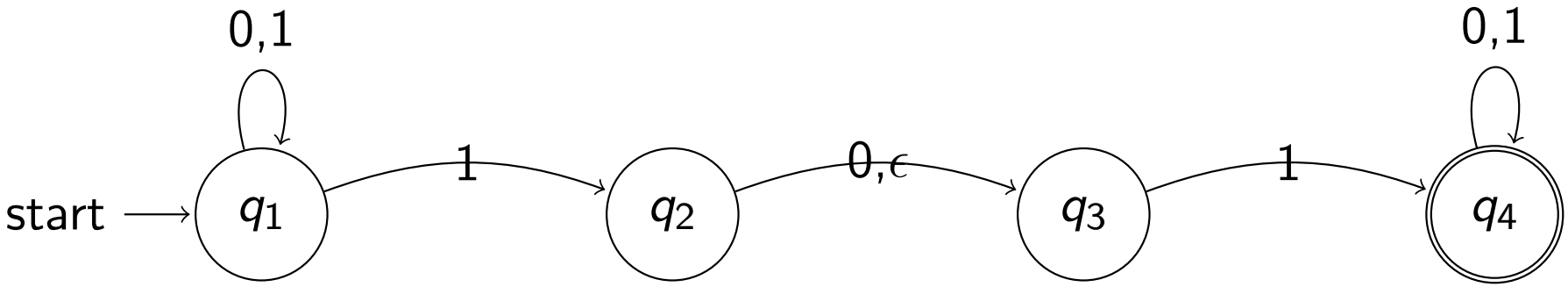
When the machine is in a given state and reads an input symbol, there is exactly one next state (after transition). The next state is fully determined.

Definition (Non-deterministic)

When the machine is in a given state and reads an input symbol, there may be several choices for the next state (after transition). The next state is NOT fully determined.

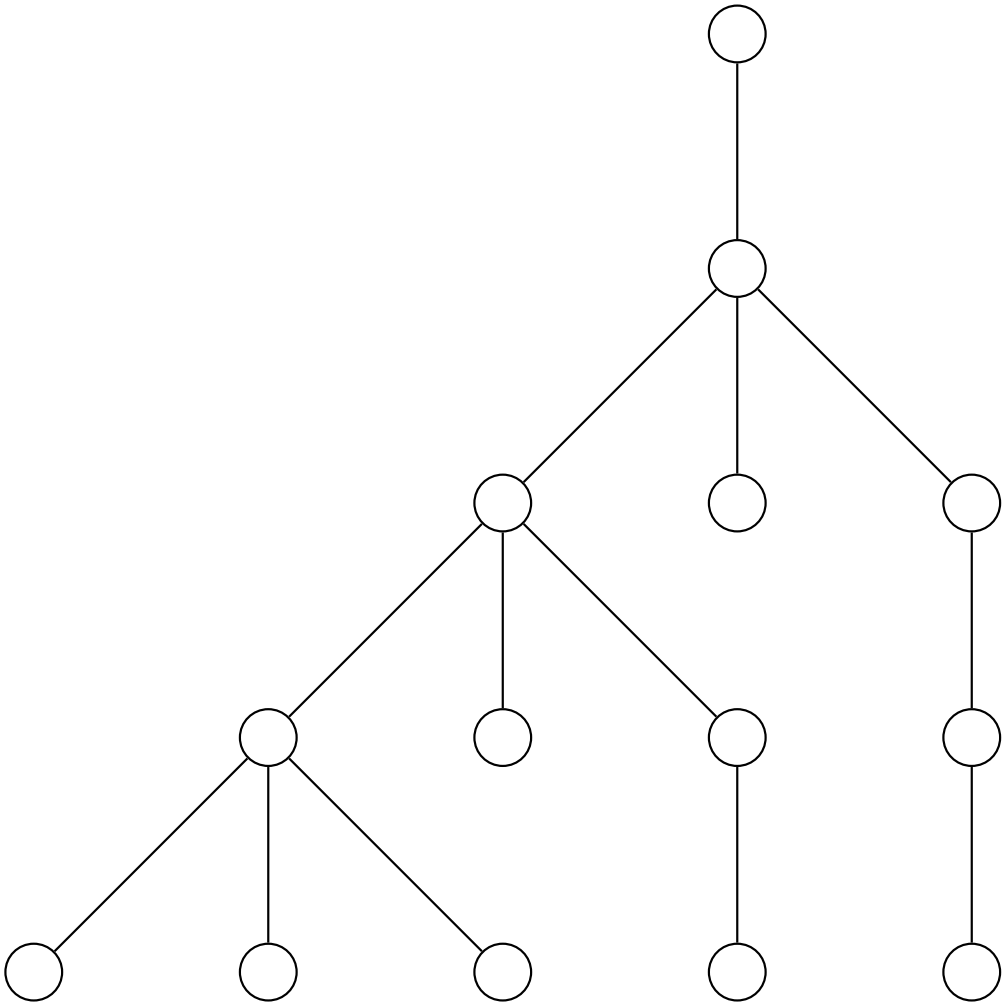
very Deterministic Finite Automaton (DFA) is a Non-deterministic Finite Automaton (NFA).

NFA



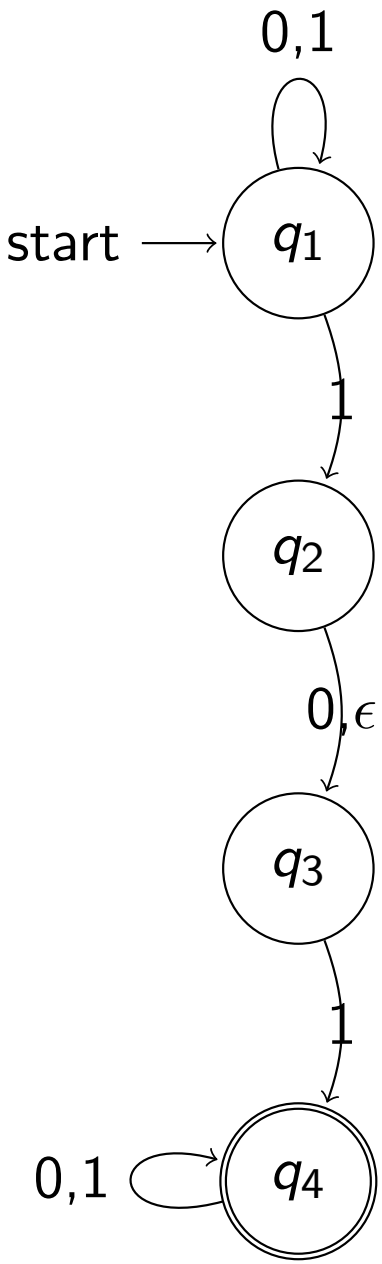
- How do we compute?
  - multiple paths!
  - multiple copies of the machine, each proceeding independently.
- If the next input symbol does not appear in any arrow, the machine dies.
- If a path accepts then input, then the NFA accepts the input.

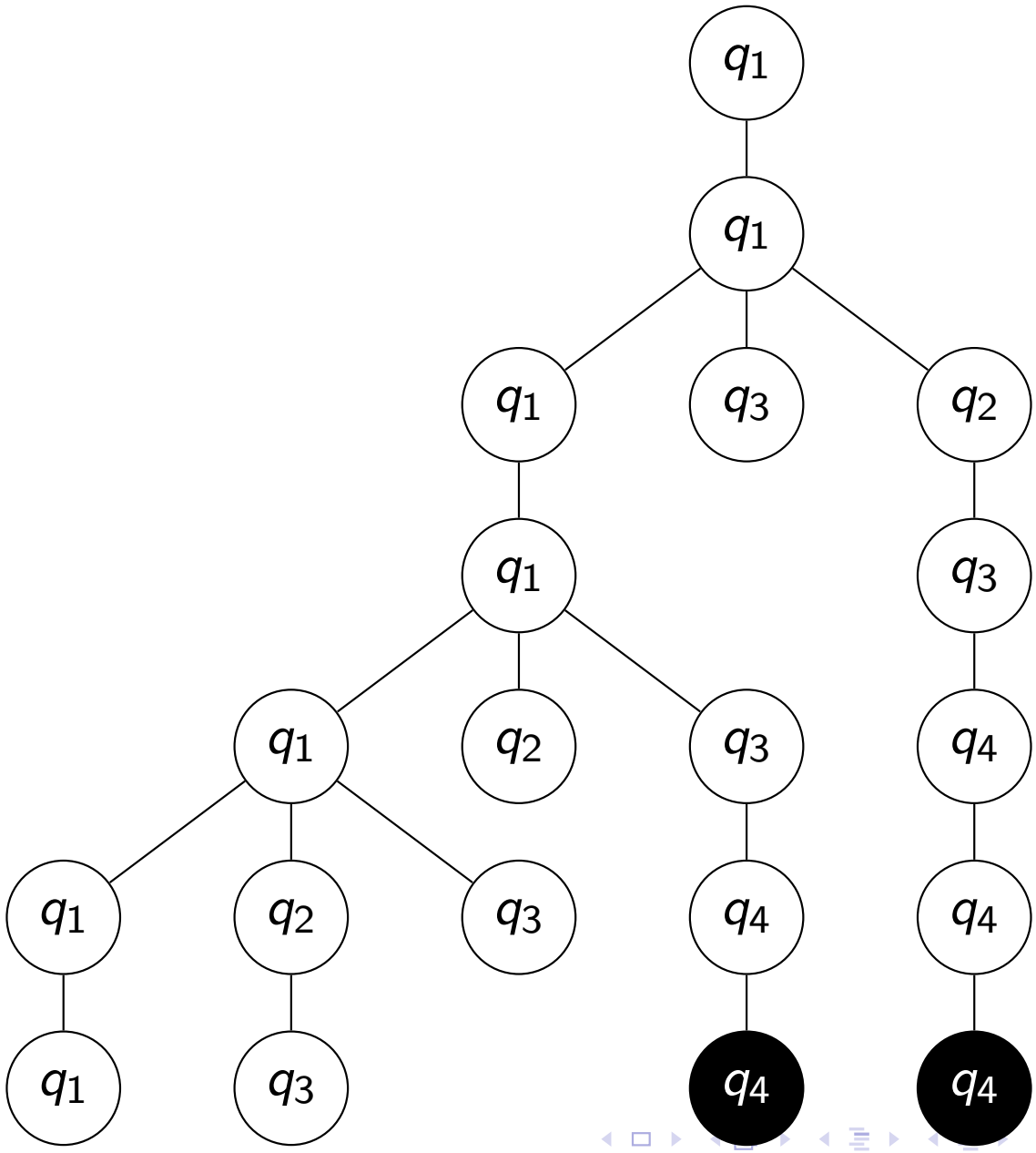
Non-deterministic Finite Automata - Computation



NFA

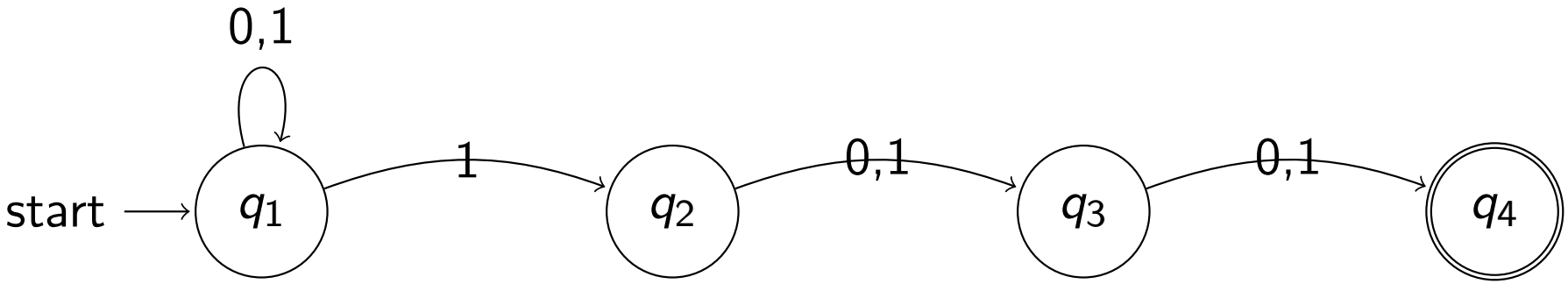
Input: 010110





Example

Let A be the language of all strings over $\{0, 1\}$ having 1 in the third position from the end.



NFA

If there are many exiting arrows for a symbol, a guess is made and an arrow is followed

- if all guesses are correct, an input string ends in an accepting state if it is accepted by the automaton.
- If an input string is not accepted by the automaton, there is no sequence of guesses ending in an accepting state.

Non-deterministic Finite Automaton

Definition

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

- Q is a finite set of states
- Σ is a finite alphabet
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function.
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Notation:

- $\mathcal{P}$ denotes the power set
- $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

Computation

Definition

Let

- $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and
- $w = w_1 w_2 \ldots w_m$ an input string

We say that N accepts w if there exists a sequence of states $r_0, r_1, \ldots, r_m$ such that:

- $r_0 = q_0$
- $r_{i+1} \in \delta(r_i, w_{i+1})$
- $r_m \in F$.

DFA vs NFA

Question

Which provides more power?

- *which one recognizes more languages*

DFA vs NFA

Question

Which provides more power?

- *which one recognizes more languages*

Answer (Surprising!)

Deterministic and non-deterministic finite automata recognize the same class of languages.

DFA vs NFA

Question

Which provides more power?

- *which one recognizes more languages*

Answer (Surprising!)

Deterministic and non-deterministic finite automata recognize the same class of languages.

But describing an NFA for a given language is sometimes much easier!

Equivalent Automata

Definition

Two automata are called equivalent if they recognize the same language.

Theorem

Every non-deterministic finite automaton has an equivalent deterministic finite automaton.

Equivalent Automata

Definition

Two automata are called equivalent if they recognize the same language.

Theorem

Every non-deterministic finite automaton has an equivalent deterministic finite automaton.

sketch.

- Simulate NFA by DFA
- branches $\rightarrow$ subsets of states
- k states $\rightarrow 2^k$ subsets of states
- Each subset is a path that the DFA must remember.

Equivalent Automata

Theorem

Every non-deterministic finite automaton has an equivalent deterministic finite automaton.

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A.
Assume N contains no ϵ arrows. Then

- $Q' = \mathcal{P}(Q)$
- Σ
- $\delta'(S, s) = \{q \in Q : q \in \delta(r, s) \text{ for some } r \in S\}$
- $q_0' = \{q_0\}$
- $F' = \{S \in Q' : S \text{ contains an accept state of } N\}$

Equivalent Automata

Theorem

Every non-deterministic finite automaton has an equivalent deterministic finite automaton.

Proof.

For $S \subseteq Q$,define

$E(S) = \{q \in Q : q \text{ can be reached from } S \text{ by using 0 or more } \epsilon \text{ arrows}\}$

Then we modify the automaton we constructed as follows:

- $\delta'(S, s) = \{q \in Q : q \in E(\delta(r, s)) \text{ for some } r \in S\}$
- $q_0' = E(\{q_0\})$

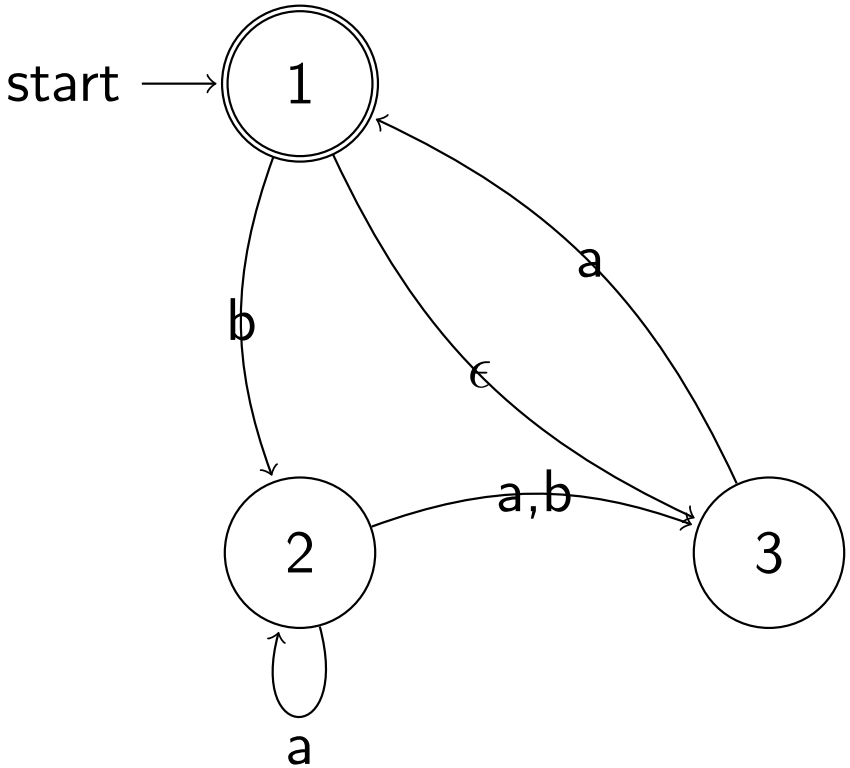
□

Regular Languages

Corollary

A language is regular if and only if some non-deterministic finite automaton recognizes it.

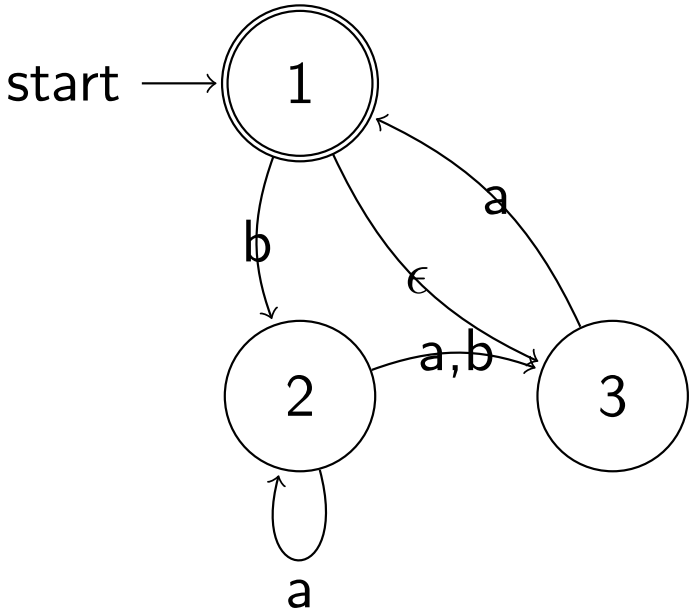
Example NFA $\rightarrow$ DFA

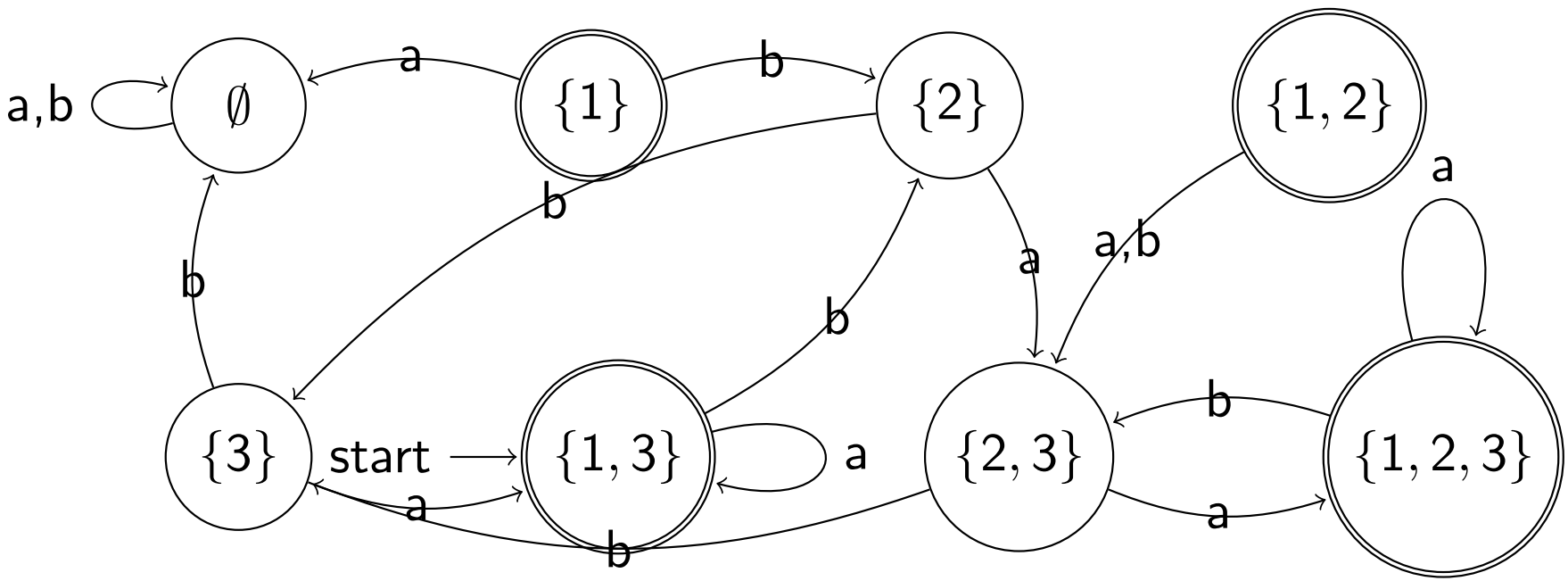


We construct the following automaton:

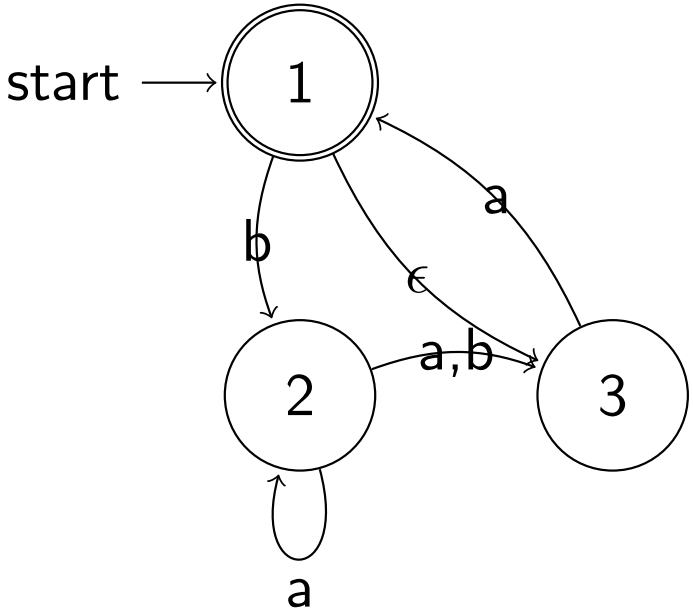
- $Q = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
- $q_0 = E(\{1\}) = \{1,3\}$
- $F = \{\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}$

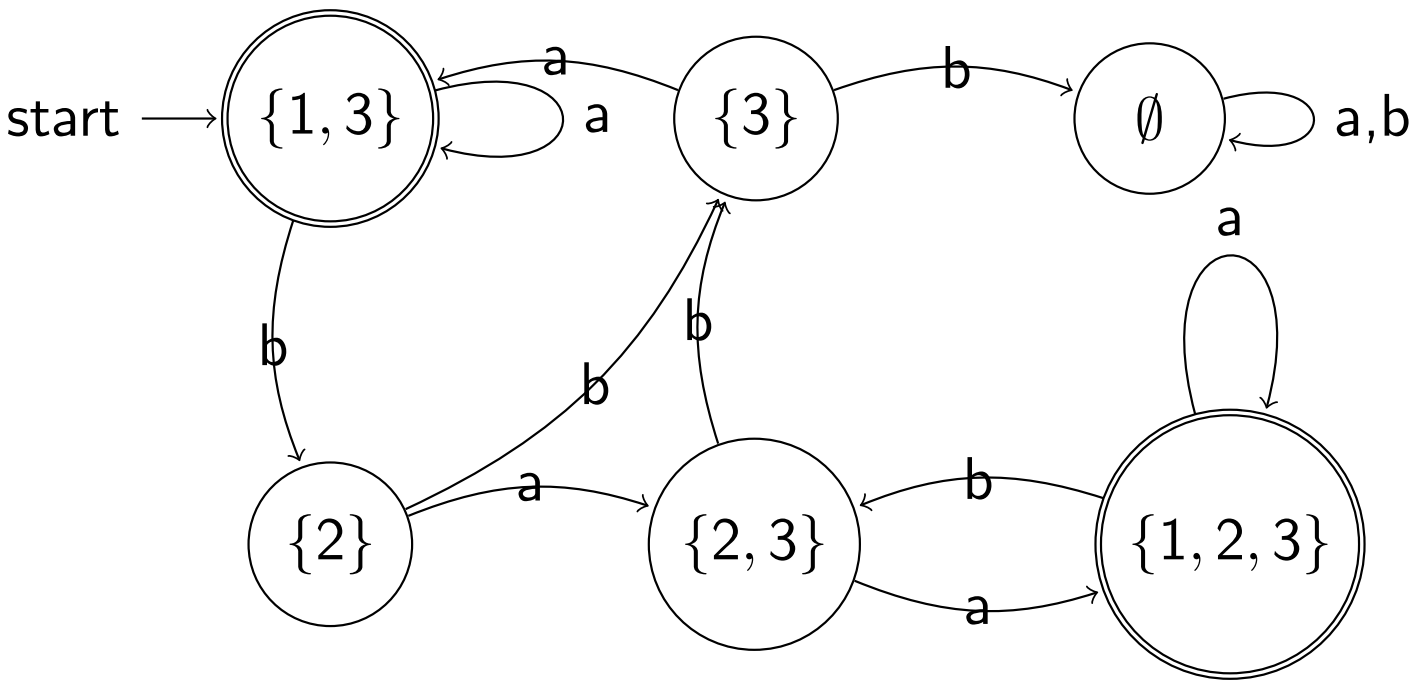
Example NFA $\rightarrow$ DFA





Example NFA $\rightarrow$ DFA





Closure Proofs

Closure under Union

Theorem

The class of regular languages is closed under union.

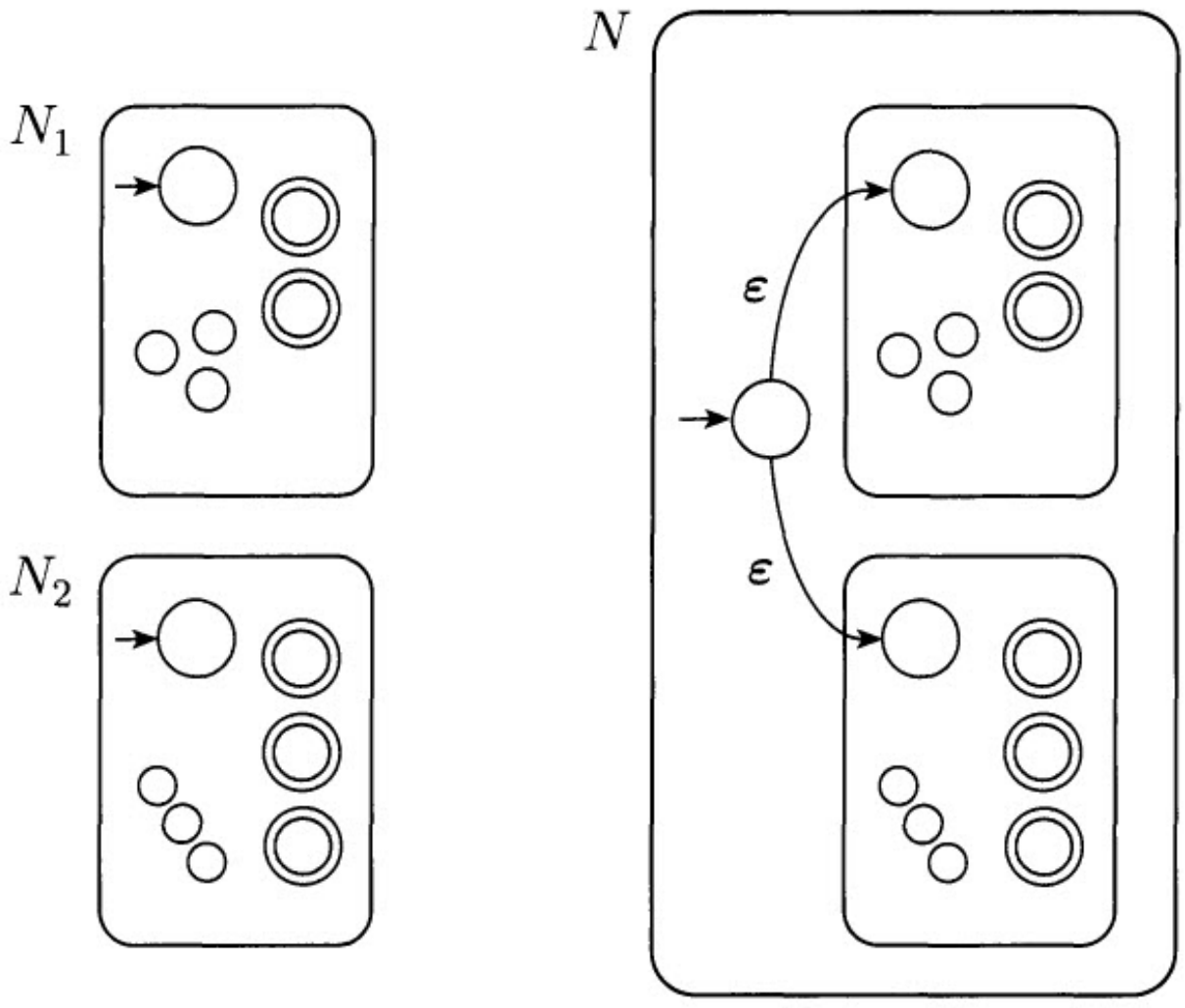
Proof.

- Let $M_1 = (Q1, \Sigma_1, \delta_1, q_{0,1}, F_1)$ recognize A_1
- Let $M_2 = (Q2, \Sigma_2, \delta_2, q_{0,2}, F_2)$ recognize A_2

We will construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 \cup A_2$:

- $Q = \{q_0\} \cup Q_1 \cup Q_2$
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $\delta\left((q_1, q_2), a\right) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_{0,1}, q_{0,2}\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$
- $q_0 = q_0$
- $F = F_1 \cup F2$

Closure under Union



Closure under Concatenation

Theorem

The class of regular languages is closed under concatenation.

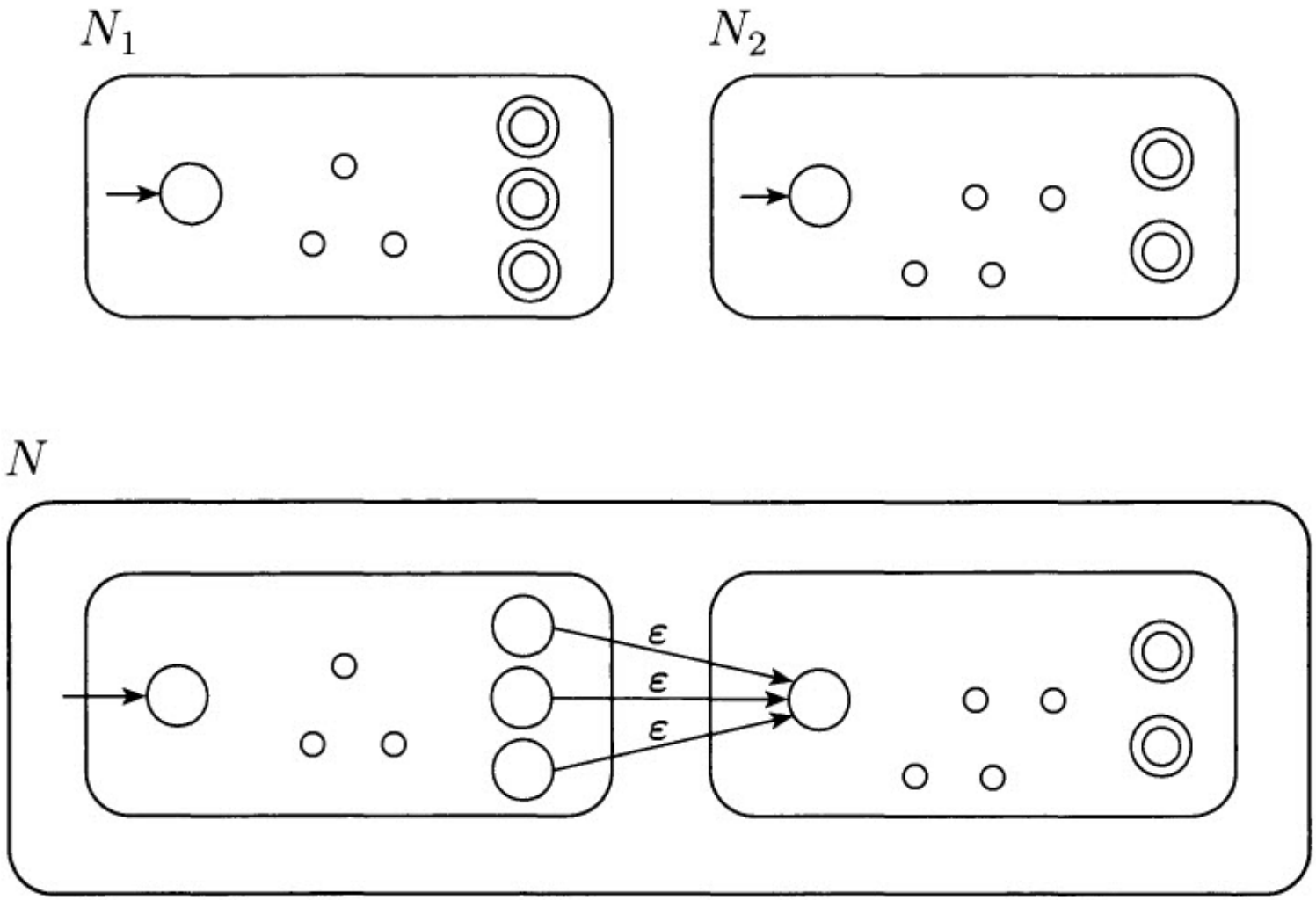
Proof.

- Let $M_1 = (Q1, \Sigma_1, \delta_1, q_{0,1}, F_1)$ recognize A_1
- Let $M_2 = (Q2, \Sigma_2, \delta_2, q_{0,2}, F_2)$ recognize A_2

We will construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 \circ A_2$:

- $Q = Q_1 \cup Q_2$
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $\delta((q_1, q_2), a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_{0,2}\} & q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$
- $q_0 = q_{0,1}$
- $F = F2$

Closure under Union



Closure under Star

Theorem

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

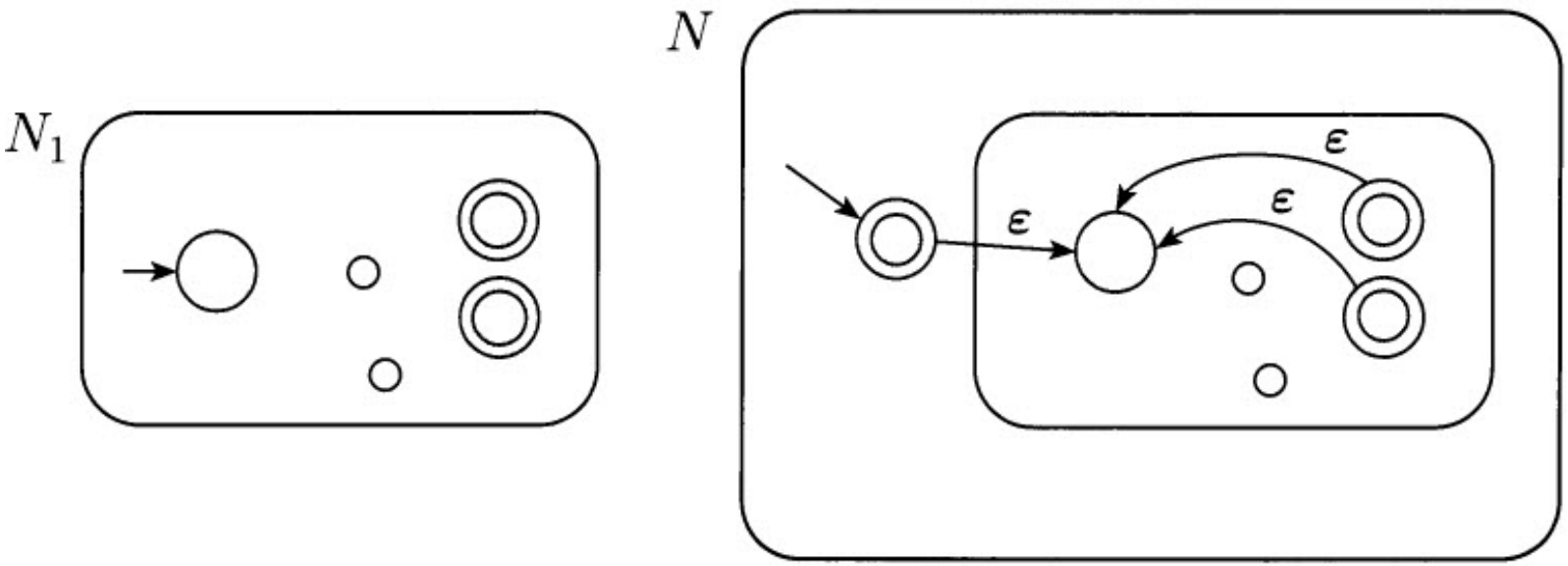
Proof.

- Let $M_1 = (Q1, \Sigma_1, \delta_1, q_{0,1}, F_1)$ recognize A

We will construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing A^*:

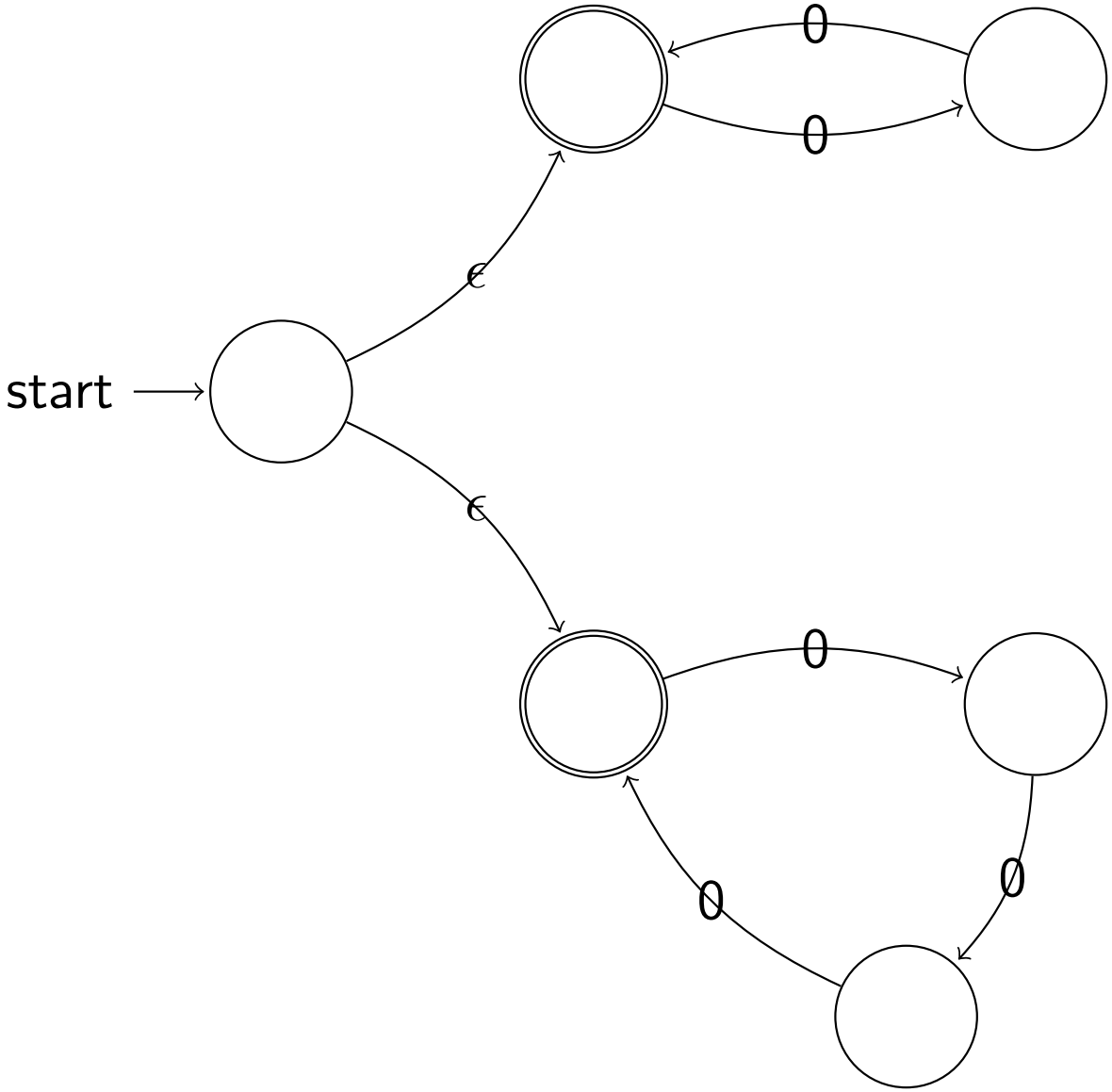
- $Q = \{q_0\} \cup Q_1$
- $\Sigma = \Sigma_1$
- $\delta((q_1, q_2), a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_{0,1}\} & q \in F_1 \text{ and } a = \epsilon \\ \{q_{0,1}\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$
- $q_0 = 0$
- $F = F2 \cup \{q_0\}$

Closure under Union

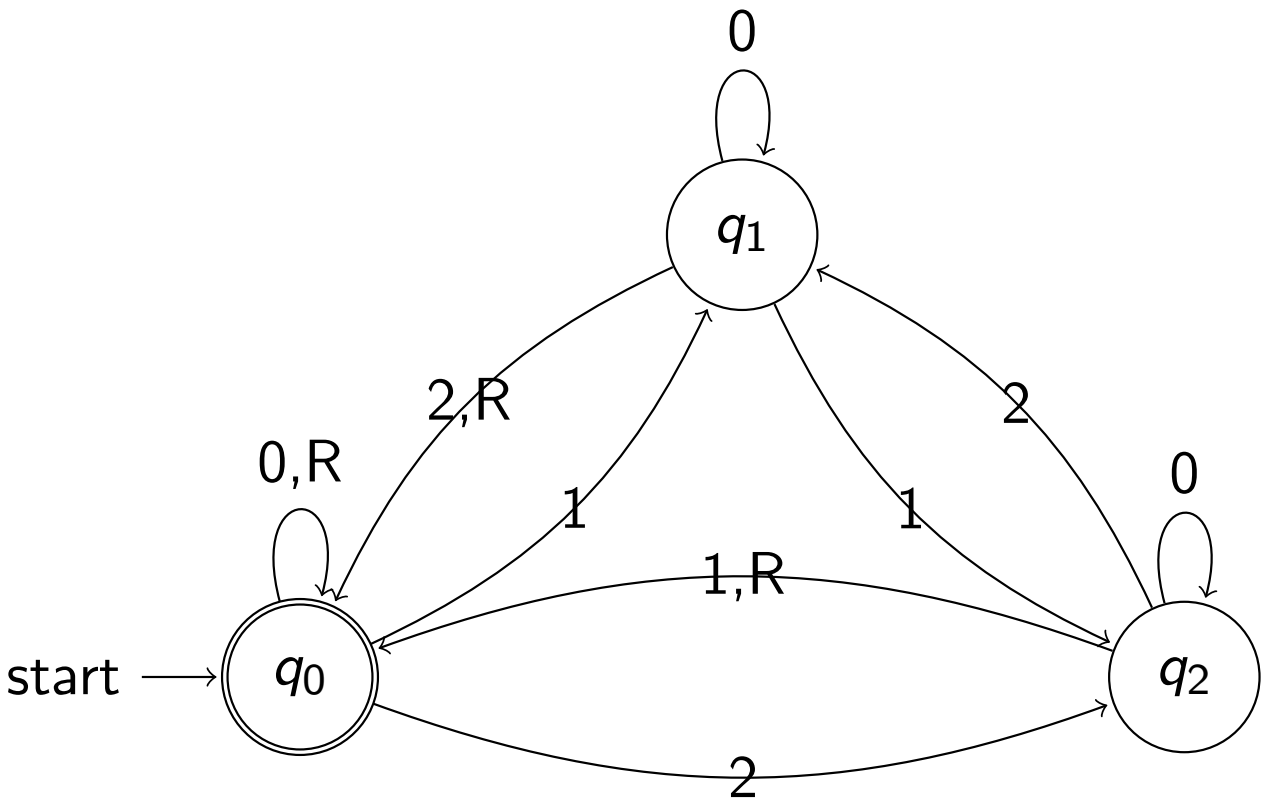


Exercise

What is the language?



What does it compute?



Regular Expressions

Expressions using the three regular operations:

- union
- concatenation
- star

Operator precedence:

$$\text{star} \;>\; \text{concatenation} \;>\; \text{union}$$

Regular Expressions

Example

The expression $(0 \cup 1)$ means $(\{0\} \cup \{1\}) = \{0, 1\}$

Example

The expression 0^* means $\{0\}^*$, which the language consisting of all strings containing any number of 0s.

Example

The expression Σ^*, where $\Sigma = \{0, 1\}$ is the language consisting of all binary strings.

Example

The expression Σ^*1, where $\Sigma = \{0, 1\}$ is the language consisting of all binary strings ending with 1.

Regular Expressions

Definition

Given an alphabet Σ, R is a regular expression if R is:

- a for some $a \in \Sigma$
- ϵ
- $\emptyset$
- $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
- (R_1^*) where R_1 is a regular expression

Regular Expressions

Definition

Given an alphabet Σ, R is a regular expression if R is:

- a for some $a \in \Sigma$
- ϵ
- $\emptyset$

These are regular languages

- $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
- (R_1^*) where R_1 is a regular expression

Notation

A^* $\{x_1 x_2 \cdots x_k : k \in \{0, 1, 2, 3, \ldots\} \text{ and } x_i \in A\}$

A^+ $\{x_1 x_2 \cdots x_k : k \in \{1, 2, 3, \ldots\} \text{ and } x_i \in A\}$

A^k $\{x_1 x_2 \cdots x_k : x_i \in A \; \forall i \in [k]\}$

Note that

$$A^* = A^+ \cup \{\epsilon\}$$

- $L(R)$ is the language described by the regular expression R

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- 0^*10^*

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single} 1\}$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single} 1\}$
- $\Sigma^*1\Sigma^*$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^*$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^*$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^*$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol }\}$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol } \}$
- $R \cup \emptyset$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol } \}$
- $R \cup \emptyset \rightarrow R$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol }\}$
- $R \cup \emptyset \rightarrow R$
- $R \circ \epsilon$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol }\}$
- $R \cup \emptyset \rightarrow R$
- $R \circ \epsilon \rightarrow R$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol }\}$
- $R \cup \emptyset \rightarrow R$
- $R \circ \epsilon \rightarrow R$
- $R \circ \emptyset$

Example

Fix $\Sigma = \{0, 1\}$.
What are the languages described by the regular expressions:

- $0^*10^* \rightarrow \{w : w \text{ contains a single1}\}$
- $\Sigma^*1\Sigma^* \rightarrow \{w : w \text{ contains at least one } 1\}$
- $\Sigma^*001\Sigma^* \rightarrow \{w : w \text{ contains the string } 001 \text{ as a substring}\}$
- $1^*\left(01^+\right)^* \rightarrow \{w : \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* \rightarrow \{w : \text{ the length of } w \text{ is a multiple of } 3\}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
  $\rightarrow \{w : w \text{ starts and ends with the same symbol } \}$
- $R \cup \emptyset \rightarrow R$
- $R \circ \epsilon \rightarrow R$
- $R \circ \emptyset \rightarrow \emptyset$

Regular Languages and Expressions

Theorem

A language is regular if and only if some regular expression describes it.

Regular Languages and Expressions

Theorem

A language is regular if and only if some regular expression describes it.

Corollary

Regular expressions and finite automata are equivalent in their description power.

Regular Languages and Expressions

Theorem

A language is regular if and only if some regular expression describes it.

Corollary

Regular expressions and finite automata are equivalent in their description power.

Proof idea:

- RE $\rightarrow$ FA
- FA $\rightarrow$ RE

Convert RE to FA

To convert a regular expression R to an NFA N, we need to consider the 6 cases in the definition of a regular expression.

- a for some $a \in \Sigma$
- ϵ
- $\emptyset$
- $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
- (R_1^*) where R_1 is a regular expression

We will provide NFAs recognizing $L(R)$.

Convert RE to FA

- $R = a$ for some $a \in \Sigma$.

Convert RE to FA

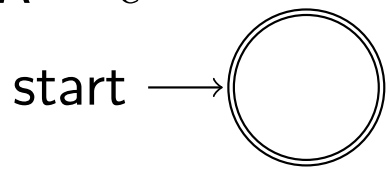
- $R = a$ for some $a \in \Sigma$.

- $R = \epsilon$

Convert RE to FA

- $R = a$ for some $a \in \Sigma$.

- $R = \epsilon$

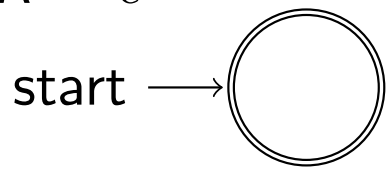


- $R = \emptyset$

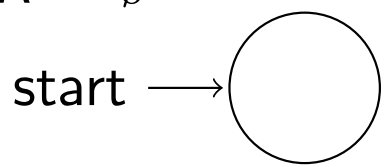
Convert RE to FA

- $R = a$ for some $a \in \Sigma$.

- $R = \epsilon$



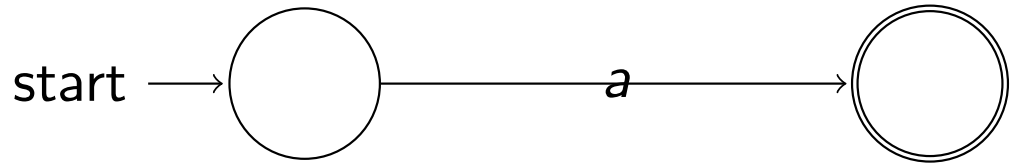
- $R = \emptyset$



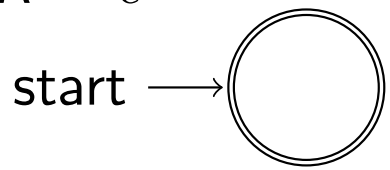
- $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
- (R_1^*) where R_1 is a regular expression

Convert RE to FA

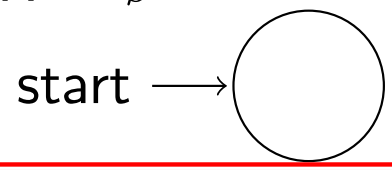
- $R = a$ for some $a \in \Sigma$.



- $R = \epsilon$



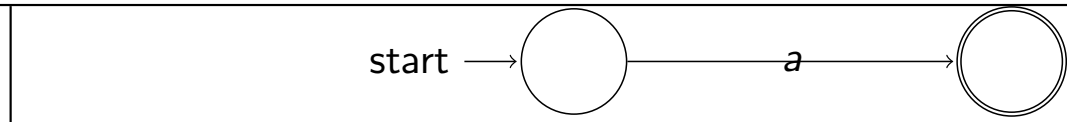
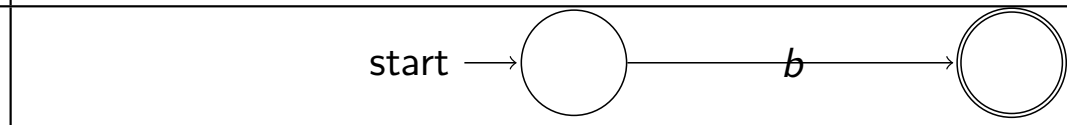
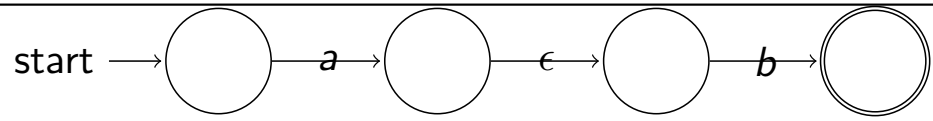
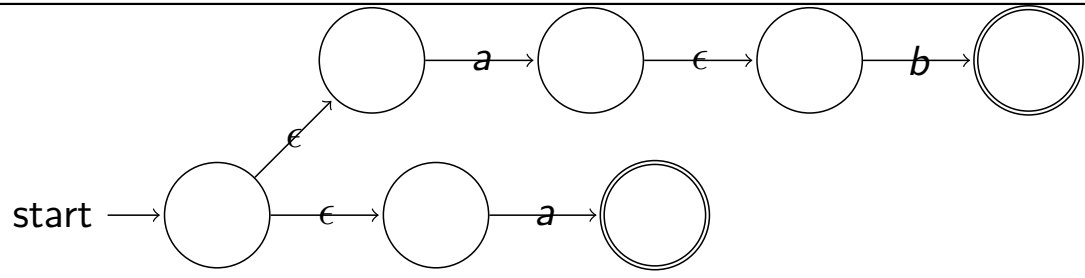
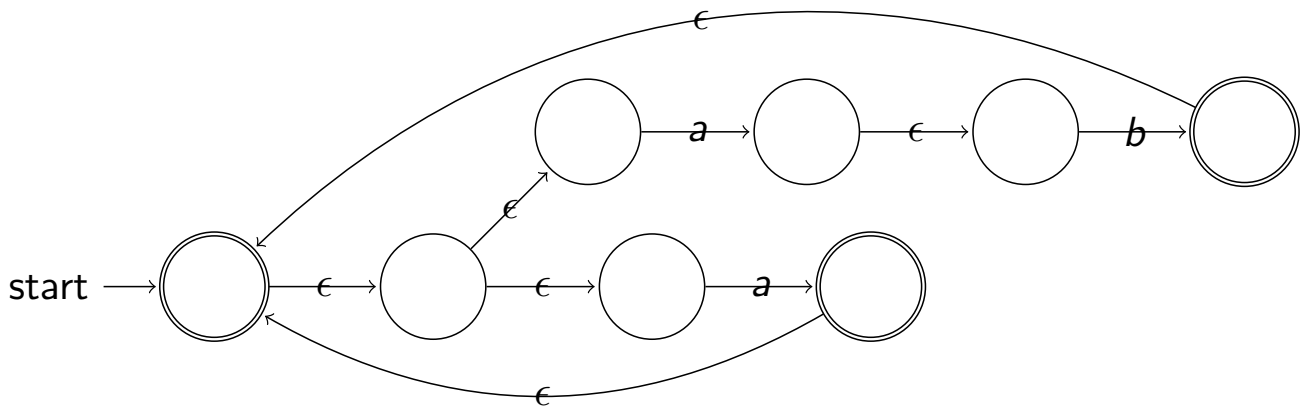
- $R = \emptyset$



- $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
- (R_1^*) where R_1 is a regular expression

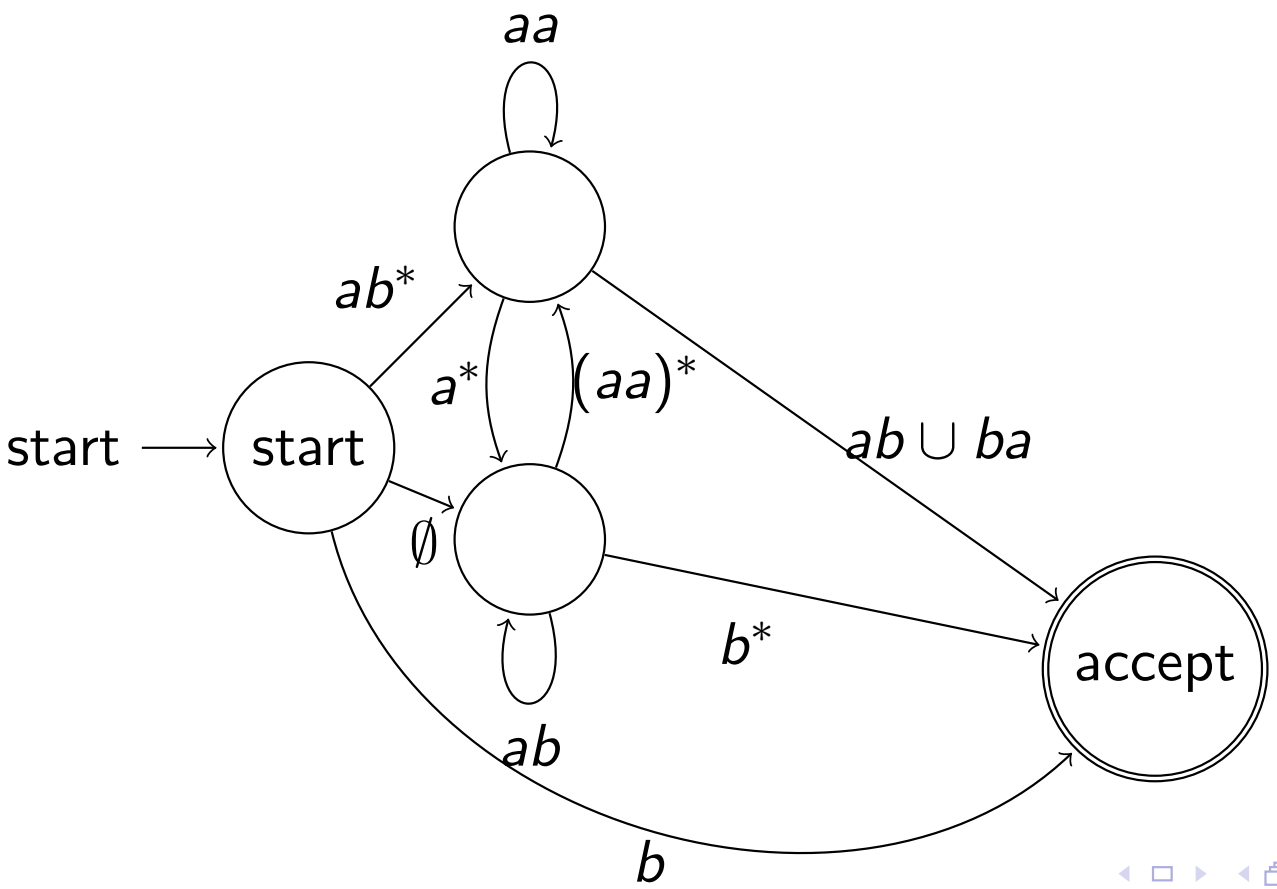
Class of regular languages is closed under regular operations

$R = (ab \cup a)^*$

a	
b	
ab	
$ab \cup a$	
$(ab \cup a)^*$	

Generalized NFA

- One start and one accept state (different from each other).
- One arrow from every state to every other state but
  - No incoming arrow to start state
  - No outgoing arrows from accept state
- Transitions may have any regular expression
- Can read blocks of input



Generalized NFA

Definition

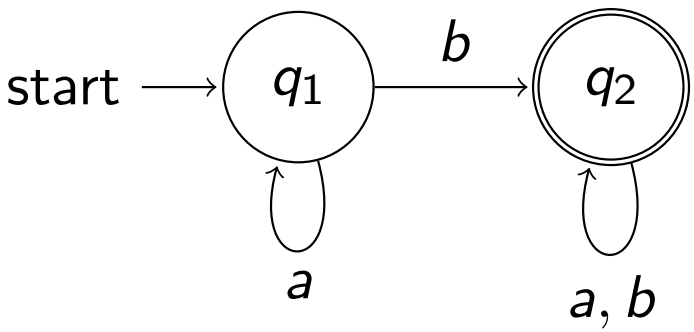
A generalized non-deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ such that

- Q is a finite set of states
- Σ is the input alphabet
- $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{accept}}\}) \to \mathcal{R}$ is the transition function
- q_{start} is the start state
- q_{accept} is the accept state

Transform a DFA to GNFA

- Add a new state with an ϵ arrow to the old start state
- Add a new accept state with ϵ arrow from the old accept states
- If an arrow had multiple labels, replace them with their union
- Add arrows labeled $\emptyset$ between states that had no arrows

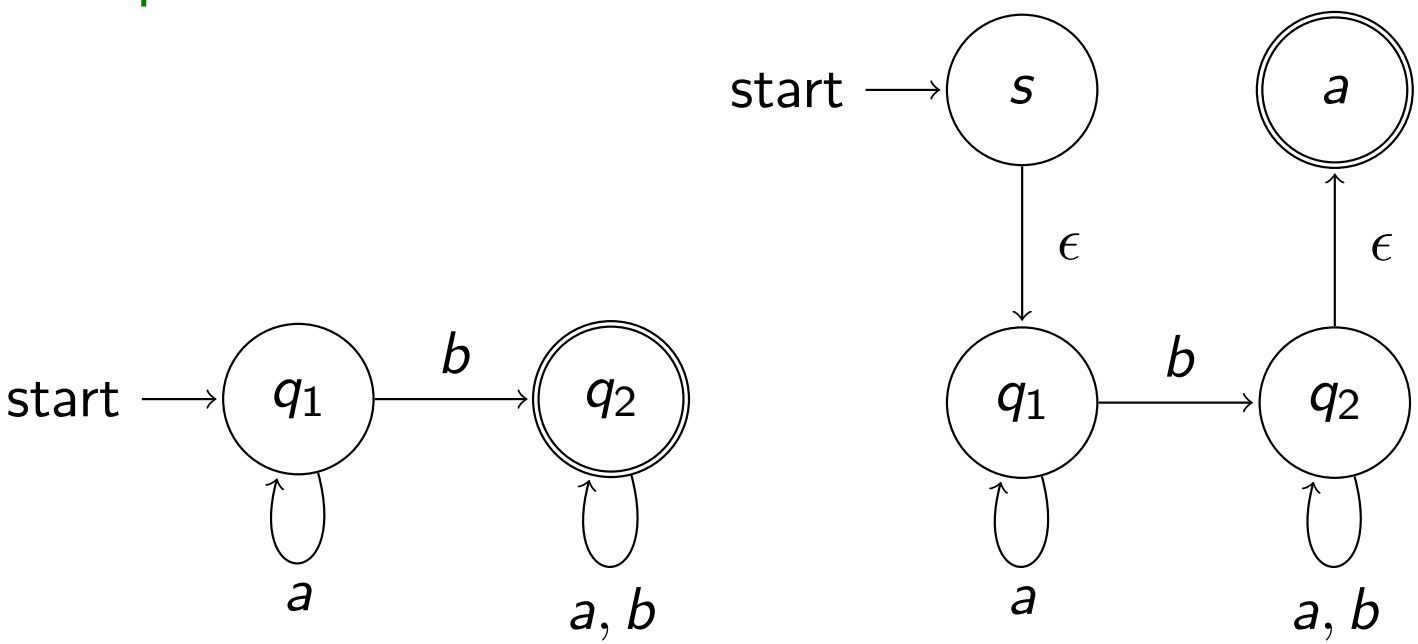
Example



Transform a DFA to GNFA

- Add a new state with an ϵ arrow to the old start state
- Add a new accept state with ϵ arrow from the old accept states
- If an arrow had multiple labels, replace them with their union
- Add arrows labeled $\emptyset$ between states that had no arrows

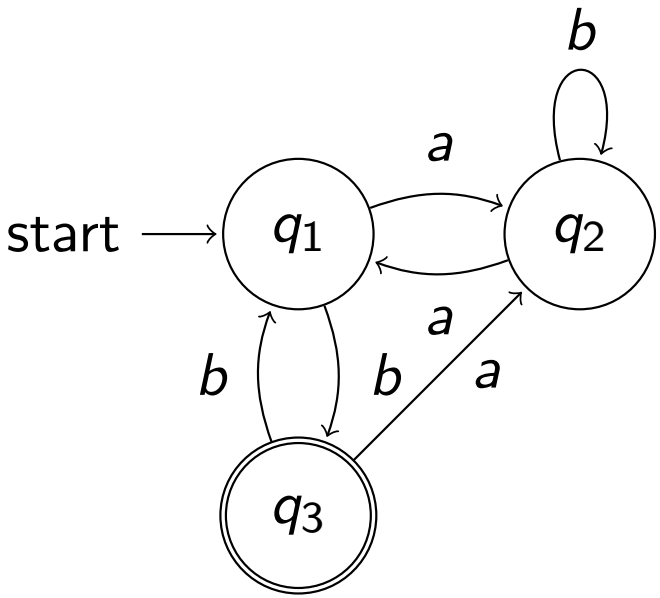
Example



Transform a DFA to GNFA

- Add a new state with an ϵ arrow to the old start state
- Add a new accept state with ϵ arrow from the old accept states
- If an arrow had multiple labels, replace them with their union
- Add arrows labeled $\emptyset$ between states that had no arrows

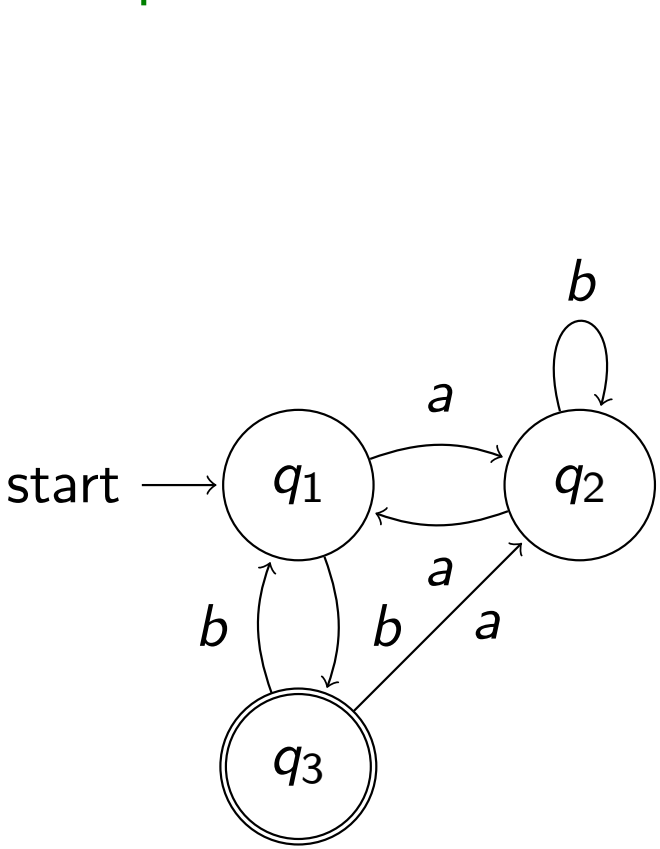
Example

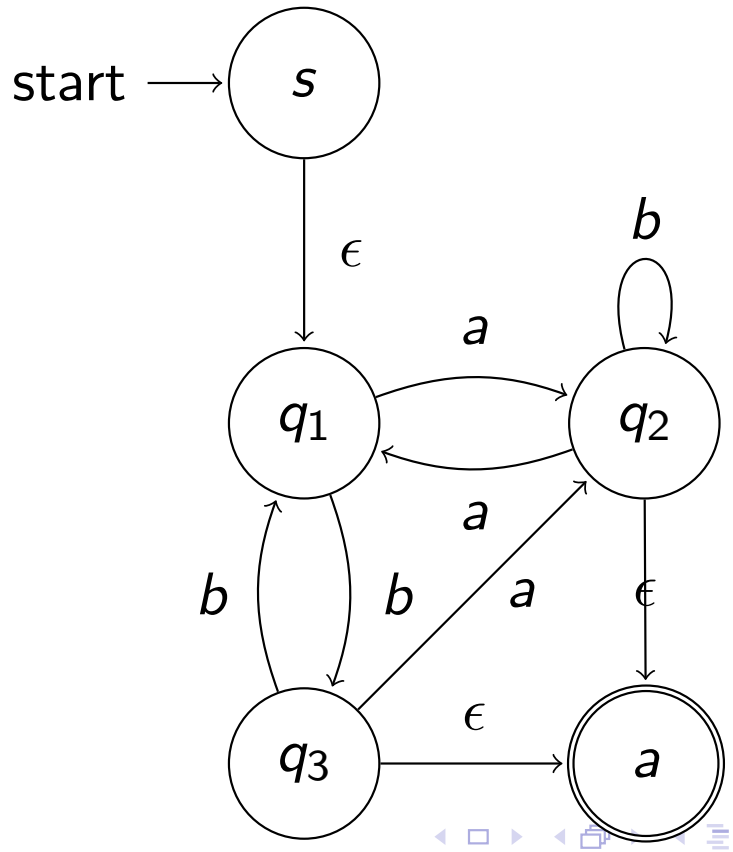


Transform a DFA to GNFA

- Add a new state with an ϵ arrow to the old start state
- Add a new accept state with ϵ arrow from the old accept states
- If an arrow had multiple labels, replace them with their union
- Add arrows labeled $\emptyset$ between states that had no arrows

Example





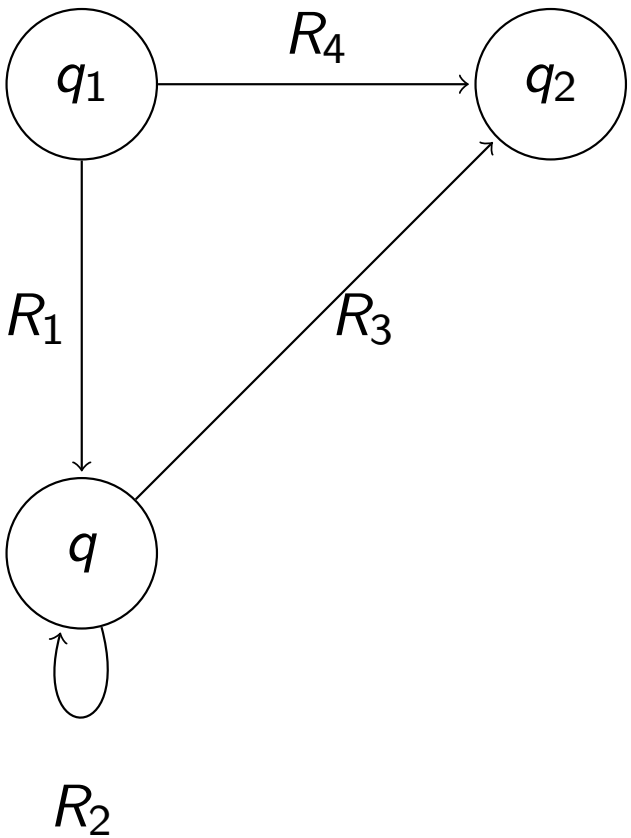
Transform GNFA to RE

Let $k = |Q|$. Since the start and accept states are different, we have $k \geq 2$.

$k = 2$ The label of the arrow is the equivalent RE.

$k < 2$ Iteratively delete arrows, updating the labels.

Example



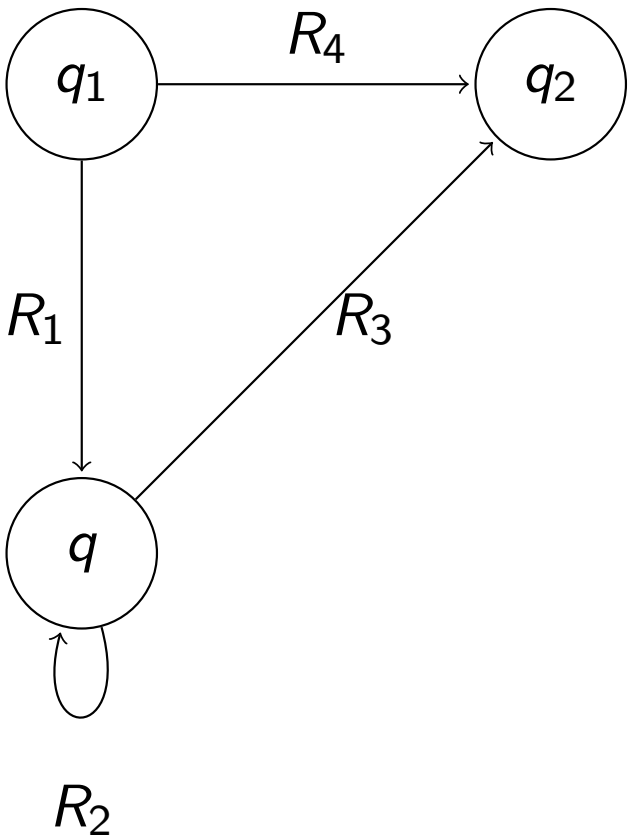
Transform GNFA to RE

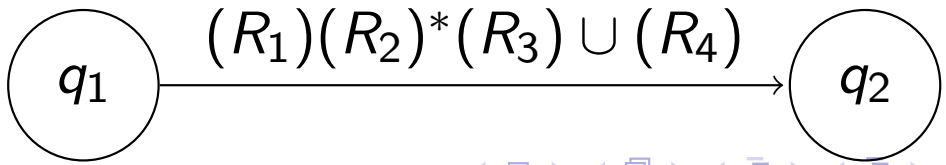
Let $k = |Q|$. Since the start and accept states are different, we have $k \geq 2$.

$k = 2$ The label of the arrow is the equivalent RE.

$k < 2$ Iteratively delete arrows, updating the labels.

Example





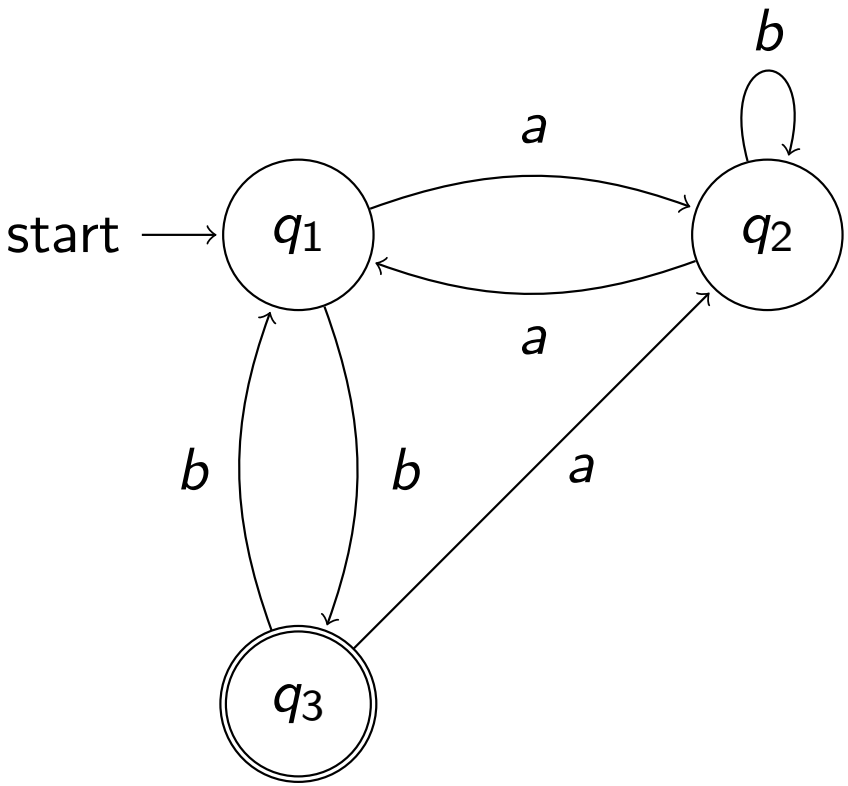
Transform DFA to RE

We showed that

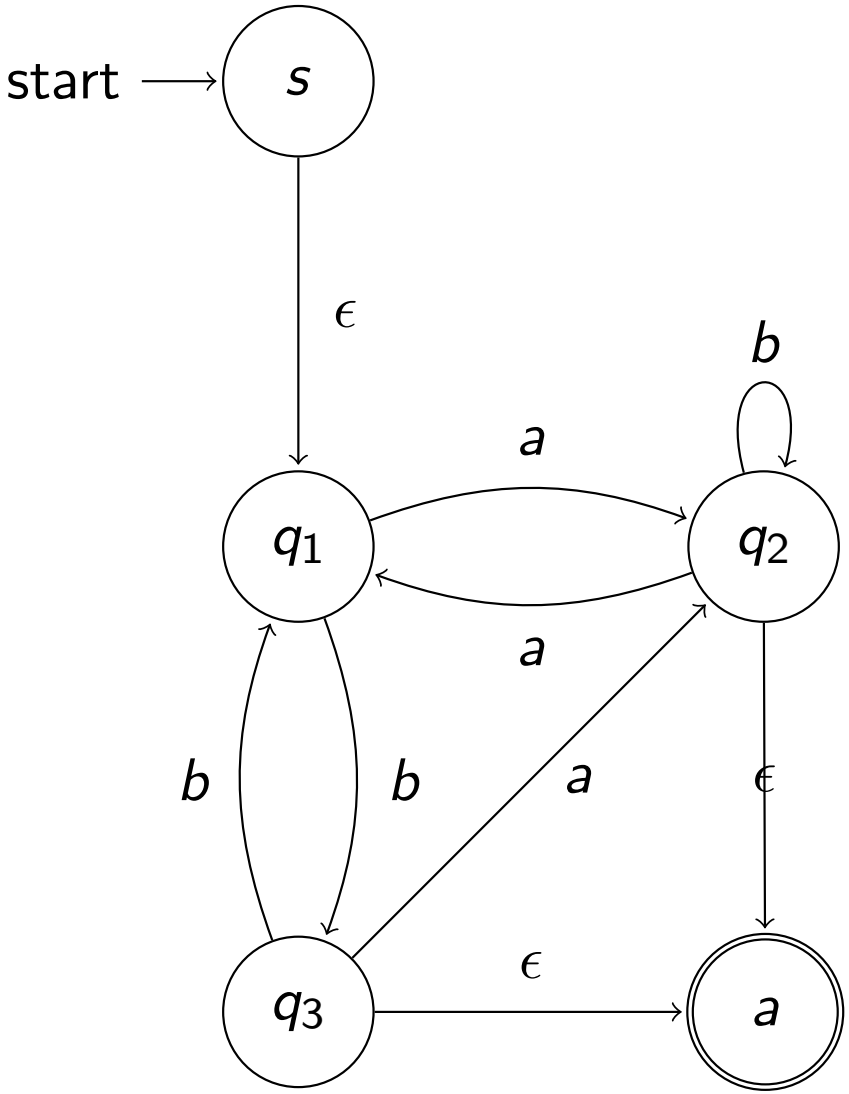
- We can transform a DFA to a GNFA
- We can transform a GNFA to an RE

This concludes the proof of the theorem.

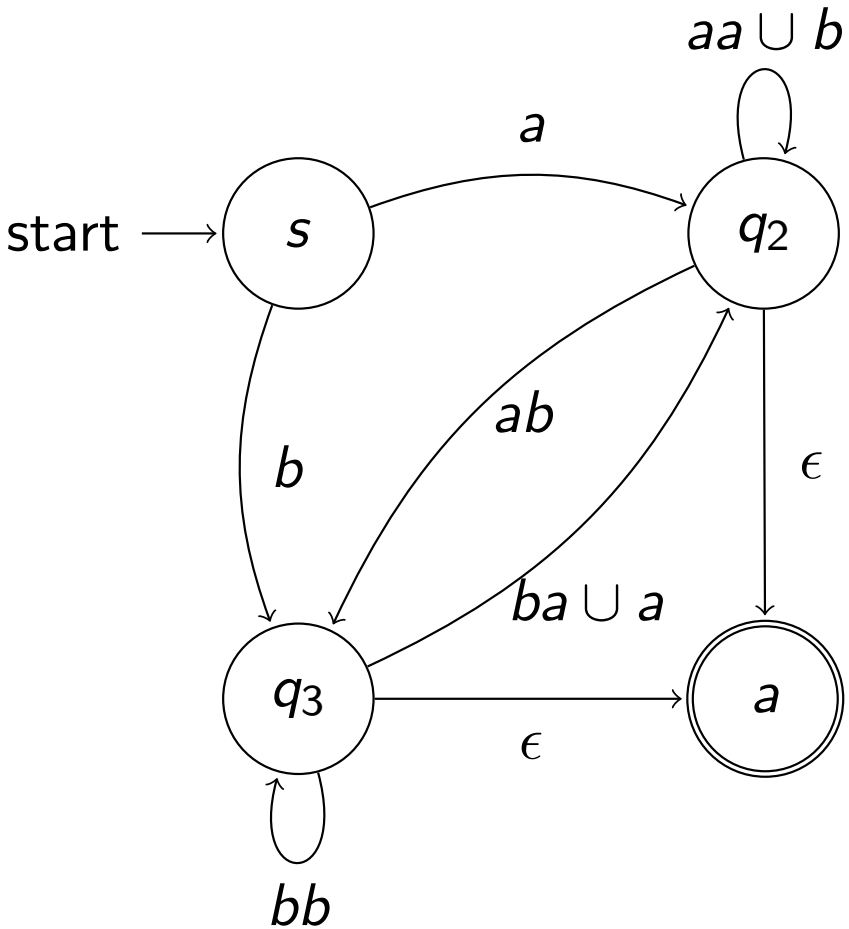
DFA



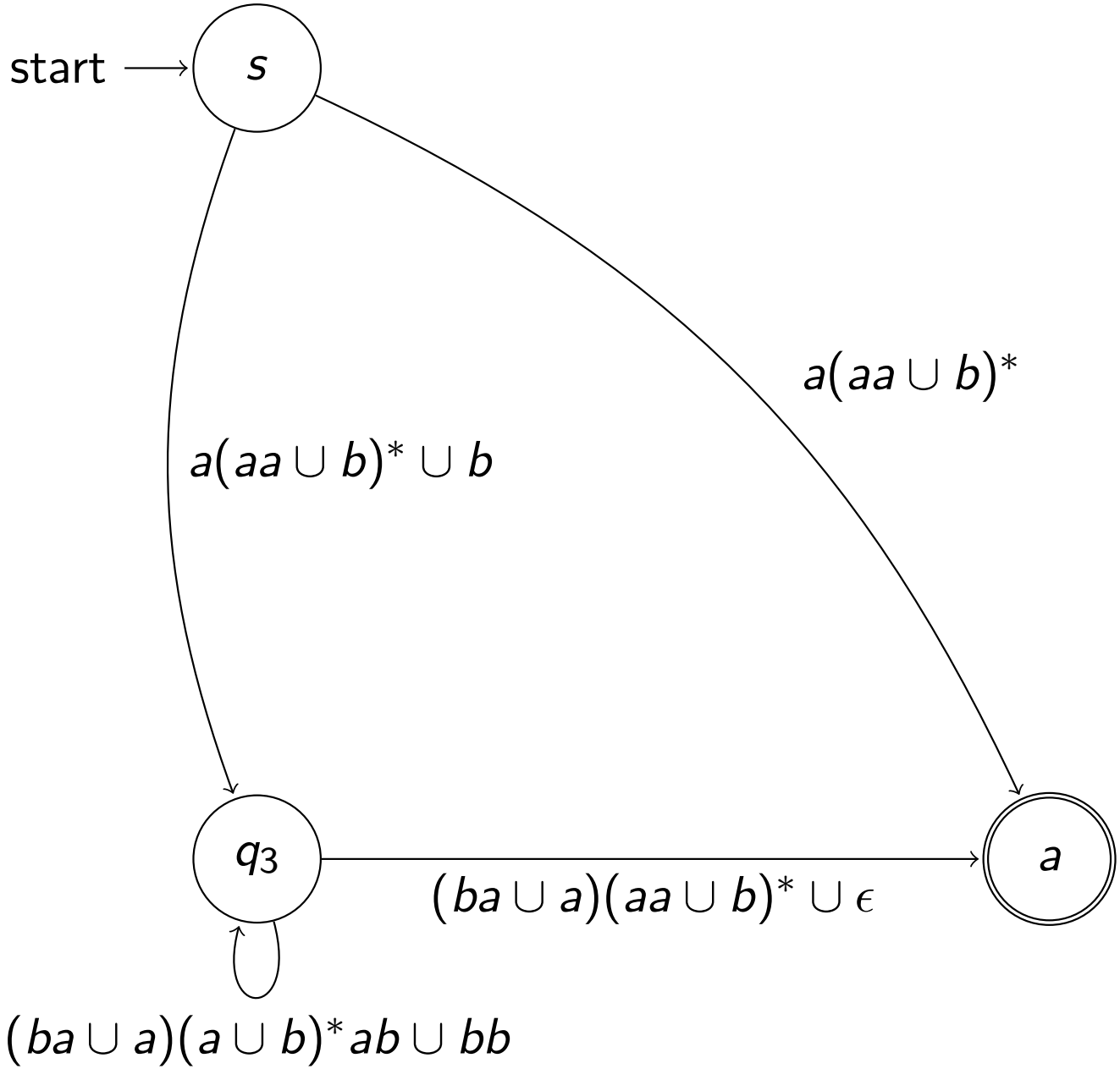
GNFA



Eliminate



Eliminate


start
s
$a(aa \cup b)^*$
$a(aa \cup b)^* \cup b$
q_3
a
$(ba \cup a)(aa \cup b)^* \cup \epsilon$
$(ba \cup a)(a \cup b)^* ab \cup bb$

The RE

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

$$((ba \cup a)(aa \cup b)^* \cup \epsilon) \cup a(aa \cup b)^*$$

Non-Regular Languages

Non-Regular Languages

Example

Consider the language

$$A = \{0^n 1^n : n \geq 0\}$$

- We need to remember the number of 0s.
- The number of 0s is not bounded.

This means an unbounded number of possibilities, but finite state machines cannot keep track of that.

Non-Regular Languages

Example

Consider the language

$$A = \{0^n 1^n : n \geq 0\}$$

- We need to remember the number of 0s.
- The number of 0s is not bounded.

This means an unbounded number of possibilities, but finite state machines cannot keep track of that.
That's a good way of thinking of the problem and understanding it. But it is not a proof of non-regularity.

Non-Regular Languages

Example

Consider the languages

$$B = \{w : w \text{ has an equal number of 0 and 1}\}$$

$$C = \{w : w \text{ has an equal number of 01 and 10}\}$$

They look like non-regular given the previous discussion.

- B is non-regular
- C is regular

Non-Regular Languages

Example

Consider the languages

$$B = \{w : w \text{ has an equal number of 0 and 1}\}$$

$$C = \{w : w \text{ has an equal number of 01 and 10}\}$$

They look like non-regular given the previous discussion.

- B is non-regular
- C is regular

OK, we need proofs. Intuition is sometimes not enough!

Pumping Lemma

Theorem (Pumping Lemma)

If A is a regular language, then there is a number p, the pumping length, such that if s is any word in A of length at least p, then s can be divided in three pieces, i.e., $s = xyz$ satisfying:

- *for each $i \geq 0$: $xy^iz \in A$*
- $|y| > 0$
- $|xy| \leq p$

If a language doesn't satisfy the pumping lemma, then it's non-regular.

Example

Let $A = \{0^n 1^n : n > 0\}$.

Assume that A is regular, then the pumping lemma holds. Let's call p the pumping length. Then we can split $s = xyz$ such that for all $i > 0$ we have $xy^i z \in A$. There are three possibilities concerning y:

- y consists only of 0s.
- y consists only of 1s.
- y consists only of 0s and 1s.

Example

Let $A = \{0^n 1^n : n > 0\}$.

Assume that A is regular, then the pumping lemma holds. Let's call p the pumping length. Then we can split $s = xyz$ such that for all $i > 0$ we have $xy^i z \in A$. There are three possibilities concerning y:

- y consists only of 0s.
- y consists only of 1s.
- y consists only of 0s and 1s.

Balance is broken in $xy^i z$

Example

Let $A = \{0^n 1^n : n > 0\}$.

Assume that A is regular, then the pumping lemma holds. Let's call p the pumping length. Then we can split $s = xyz$ such that for all $i > 0$ we have $xy^i z \in A$. There are three possibilities concerning y:

- y consists only of 0s.
- y consists only of 1s.

  Balance is broken in $xy^i z$
- y consists only of 0s and 1s.

  Order is broken in $xy^i z$

Context-Free Grammars

Context-Free Grammars

The expressive power of Finite Automata and Regular Expressions is the same.
But we need more. The study of CFG started in the context of

- human languages
- programming languages (parser)

Substitution Rule

General form

Symbol $\rightarrow$ String

- The left hand side is exactly a symbol (called variable)
- The right hand side may contain variables or terminals (symbols that are not variables)

Substitution Rule

General form

Symbol $\rightarrow$ String

- The left hand side is exactly a symbol (called variable)
- The right hand side may contain variables or terminals (symbols that are not variables)

Example

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow \#$

Context-Free Grammars

Definition

A context-free grammar is a 4-tuple (V, Σ, R, S) such where:

- V is a finite set called the **variables**
- Σ is a finite set, disjoint from V, called the **terminals**
- R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals
- $S \in V$ is the **start variable**

Language of a Grammar

Describe a language by generating each string of that language in the following manner.

- Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
- Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
- Repeat step 2 until no variables remain.

Derivation

Let u, v, and w be strings of variables and terminals and $A \to w$ a rule of the grammar. We say that

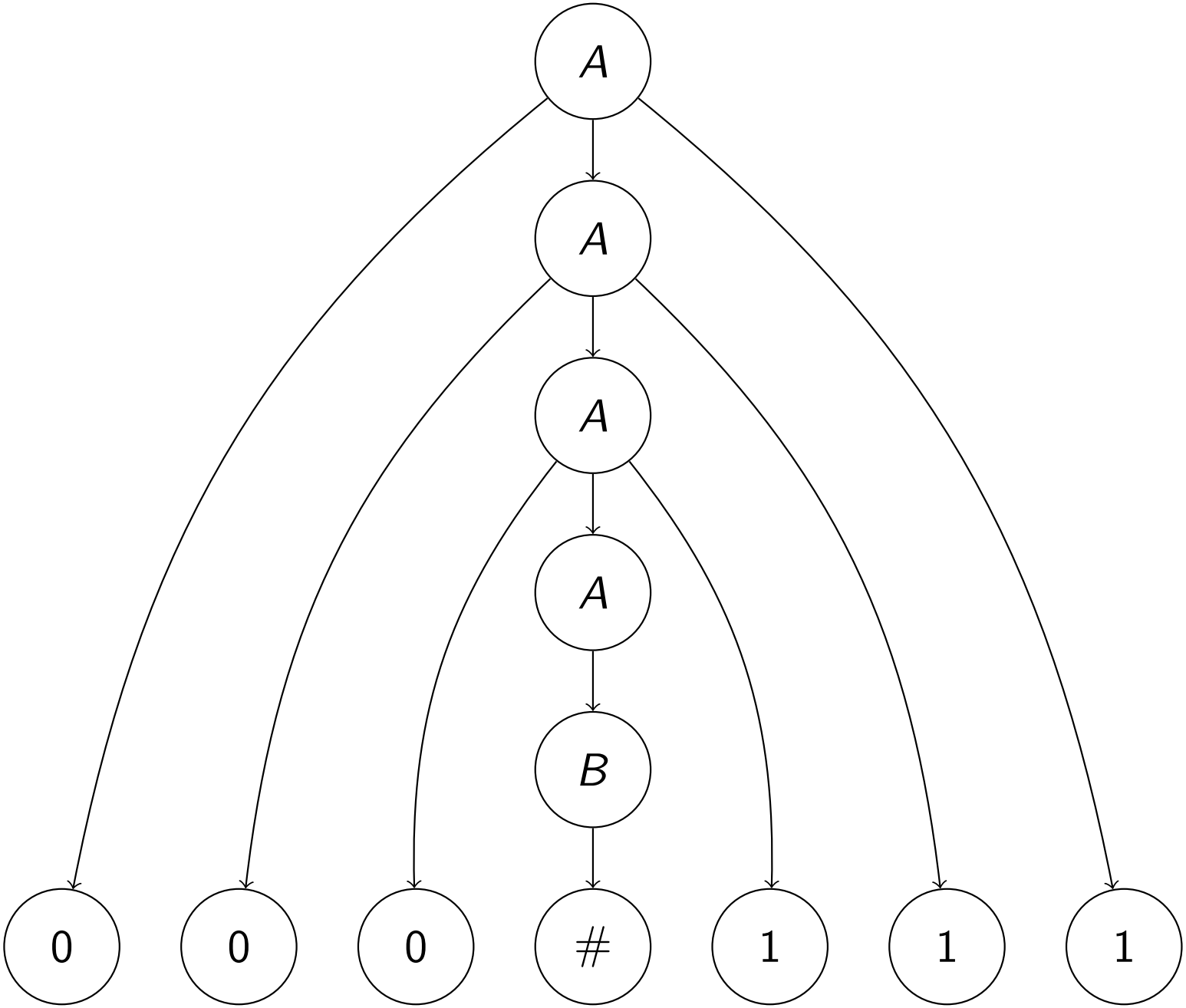
- uAv yields uwv ($uAv \to uwv$)
- u derives v ($u \to^* v$) if for some $k > 0$ there exist $u_1, u_2, \ldots, u_k$ such that $u \to u_1 \to u_2 \to \ldots \to u_k \to v$

Example (Derivation)

$G = \{A \to 0A1, A \to B, B \to \#\}$ generates the string $000\#111$.

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111.$$

Parse Tree



Language of a Grammar

Definition

All the strings generated by a grammar, constitute the **language of the grammar.**

Example

$G = \{A \to 0A1, A \to B, B \to \#\}$ generates the language

$$L(G) = \{0^n \# 1^n : n \geq 0\}$$

Example

Let G be the grammar

$$(\{S\}, \{a, b\}, \{S \to aSb|SS|\epsilon\}, S)$$

G generates for example:

- $abab$
- $aaabbb$
- $aababb$

Example

Let G be the grammar

$$(\{S\}, \{a, b\}, \{S \to aSb|SS|\epsilon\}, S)$$

G generates for example:

- $abab$
- $aaabbb$
- $aababb$

Question

What is the language $L(G)$?

Example

Let G be the grammar

$$(\{S\}, \{a, b\}, \{S \to aSb|SS|\epsilon\}, S)$$

G generates for example:

- $abab$
- $aaabbb$
- $aababb$

Question

What is the language $L(G)$?

- The language of properly nested parentheses.

Context-Free Grammars

Definition (Grammar)

A grammar consists of:

- a set of **substitution rules** (productions)
- a set of symbols called **variables**
- a **start variable**
- a set of **terminals**

Context-Free Grammars

Definition (Grammar)

A grammar consists of:

- a set of **substitution rules** (productions)
- a set of symbols called **variables**
- a **start variable**
- a set of **terminals**

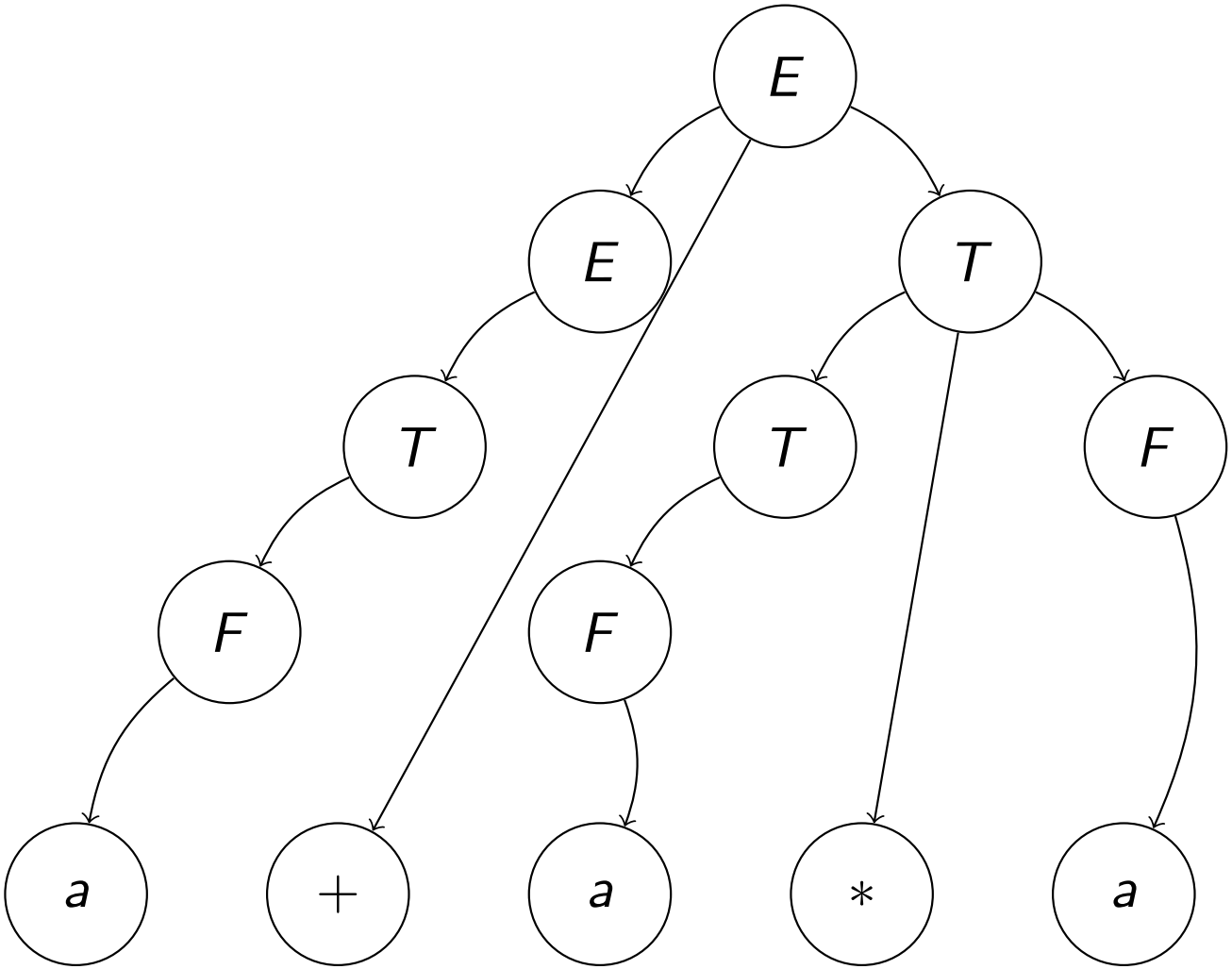
Example

- substitution rules
  - $A \to 0A1$, $A \to B$, $B \to \#$
- variables: A, B
- start variable: A
- terminals: $0, 1, \#$

Arithmetic Parser

- Variables: E, T, F
- Alphabet: $a, +, *, (,)$
- Rules:
  - $E \rightarrow E + T | T$
  - $T \rightarrow T * F | F$
  - $F \rightarrow (E) | a$

Arithmetic Parser



Context-Free Languages

Definition (Context-Free Languages)

The class of languages associated to context-free grammars.

- They include regular languages

Context-Free Languages

Definition (Context-Free Languages)

The class of languages associated to context-free grammars.

- They include regular languages
- but they also include more than that

How to Design Context-Free Grammars

Design

- Requires creativity

Design

- Requires creativity
- more than finite automata need

Design

- Requires creativity
- more than finite automata need because we are used to program machines rather than describing languages with grammars.

Let's see some tricks that can be used in their own or in combination.

Union

CFLs are the union of simpler CFLs.

- Break the language in smaller pieces
- Construct individual grammars $G1, G2, \ldots, G_k$
- Combine them by adding the rule $S \to S_1|S_2|\cdots|S_k$, where S_i is start symbol of G_i

Example

$\{0^n1^n : n \geq 0\} \cup \{1^n0^n : n \geq 0\}$

Union

CFLs are the union of simpler CFLs.

- Break the language in smaller pieces
- Construct individual grammars $G1, G2, \ldots, G_k$
- Combine them by adding the rule $S \to S_1|S_2|\cdots|S_k$, where S_i is start symbol of G_i

Example

$\{0^n1^n : n \geq 0\} \cup \{1^n0^n : n \geq 0\}$

- $G_1 = \{S_1 \to 0S_11|\epsilon\}$
- $G_2 = \{S_2 \to 1S_20|\epsilon\}$
- $G_2 = \{S_1 \to 0S_11|\epsilon, S_2 \to 1S_20|\epsilon, S \to S_1|S_2\}$

Regular

A CFL may be regular.

- Construct a DFA
- One varibale R_i for each state q_i for the DFA
- If $\delta(q_i, a) = q_j$ is a transition, add the rule $R_i \rightarrow aR_j$
- If q_i is an accepting state of the DFA, add the rule $R_i \rightarrow \epsilon$
- If q_0 is the starting state of the DFA, set R_0 to be the start variable of the grammar.

Regular

A CFL may contain string that have two parts related to each other, but we need to remember an unbounded amount of information to verify that one part corresponds to the other.

- Use a rule like uRv

Example

$$\{0^n 1^n : n \geq 0\}$$

Ambiguity

Ambiguity

Definition

If a grammar generates the same string in more than one ways, then we say that the string is derived ambiguously in the grammar.

Definition

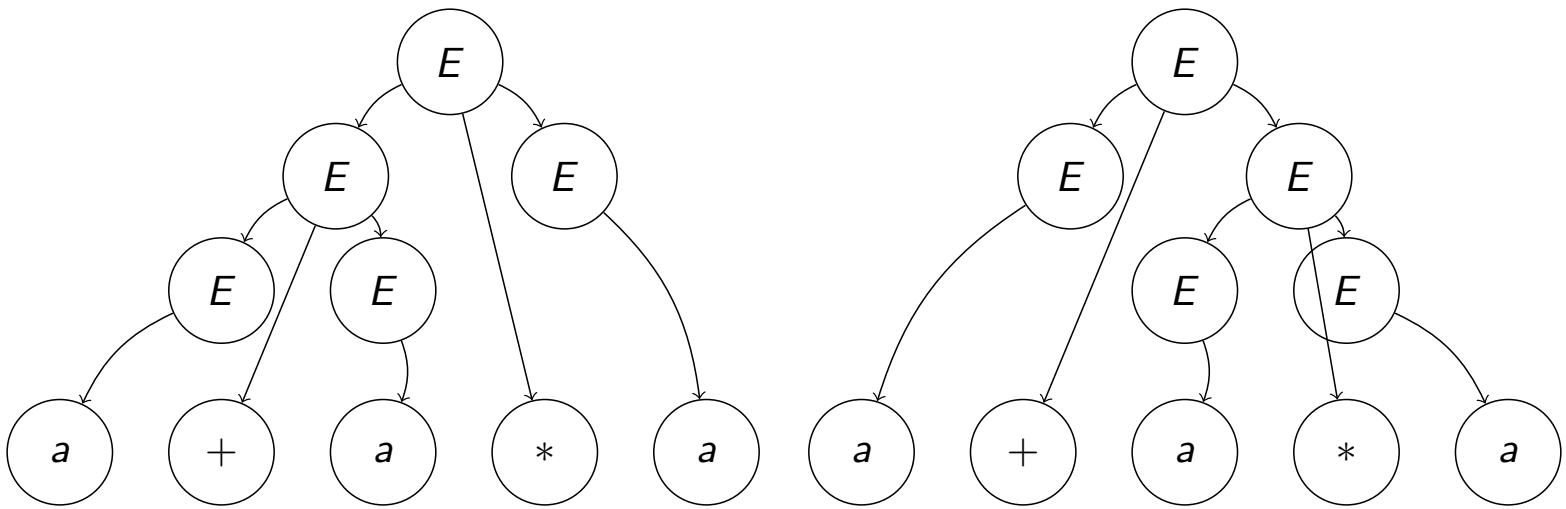
If a grammar generates some string ambiguously, then the grammar is called ambiguous.

Example

$$E \rightarrow E + E | E * E | (E) | a$$

It generates $a + a * a$ ambiguously.

Arithmetic Parser



Parse trees

- The parse tree captures the meaning
- The two expressions have the same word, but different meannings

Parse trees

- The parse tree captures the meaning
- The two expressions have the same word, but different meannings

Question

Does order matter?

Parse trees

- The parse tree captures the meaning
- The two expressions have the same word, but different meannings

Question

Does order matter?

Definition (leftmost derivation)

A derivation of a string w in a grammar G is a leftmost derivation if at every step the leftmost remaining variable is the one replaced.

Ambiguity

Definition

A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations.
Grammar G is ambiguous if it generates some string ambiguously.

- Some CFL can be genrated both by ambiguous and non-ambiguous grammars.
- If some CFL cannot be generated by a non-ambiguous grammar, it is called **inherently** ambiguous.

Chomsky Normal Form

Chomsky Normal Form

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

- $A \to BC$
- $A \to a$
- $S \to \epsilon$

where a is any terminal and A, B, and C are any variablesexcept that B and C may not be the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Transform to Chomsky Normal Form

- START: The start symbol doesn't appear in the RHS of rules
  - Introduce a new start symbol S_0
  - Introduce a new rule $S_0 \rightarrow S$

Transform to Chomsky Normal Form

- START: The start symbol doesn't appear in the RHS of rules
  - Introduce a new start symbol S_0
  - Introduce a new rule $S_0 \to S$
- Term: Eliminate rules with non-solitary terminals
  - For each rule $A \to X_1 \cdots X_i a X_{i+1} \cdots X_n$, introduce a non-terminal symbol N_a and a new rule $N_a \to a$
  - Change every rule $A \to X_1 \cdots X_i a X_{i+1} \cdots X_n$ to $A \to X_1 \cdots X_i N_a X_{i+1} \cdots X_n$

Transform to Chomsky Normal Form

- START: The start symbol doesn't appear in the RHS of rules
  - Introduce a new start symbol S_0
  - Introduce a new rule $S_0 \to S$
- Term: Eliminate rules with non-solitary terminals
  - For each rule $A \to X_1 \cdots X_i a X_{i+1} \cdots X_n$, introduce a non-terminal symbol N_a and a new rule $N_a \to a$
  - Change every rule $A \to X_1 \cdots X_i a X_{i+1} \cdots X_n$ to $A \to X_1 \cdots X_i N_a X_{i+1} \cdots X_n$
- Bin: Eliminate RHS with more than 2 non-terminal symbols
  - For each rule $A \to X_1 X_2 \cdots X_n$, introduce new symbols $A_1 A_2 \cdots A_{n-2}$
  - Replace rule $A \to X_1 X_2 \cdots X_n$, with
    - $A \to X_1 A_1$
    - $A_1 \to X_2 A_2$
    - $\ldots$
    - $A_{n-2} \to X_{n-1} X_n$

Transform to Chomsky Normal Form

- DEL: Eliminate ϵ-rules
  - For each rule $A \to X_1 X_2 \cdots X_n$ if $X_1, X_2, \cdots, X_n$, add all versions with some nullable X_i removed
  - Remove all ϵ-rules except if the LHS is the start symbol

Transform to Chomsky Normal Form

- DEL: Eliminate ϵ-rules
  - For each rule $A \to X_1 X_2 \cdots X_n$ if $X_1, X_2, \cdots, X_n$, add all versions with some nullable X_i removed
  - Remove all ϵ-rules except if the LHS is the start symbol
- Unit: Eliminate unit rules
  - For each unit rule $A \to B$
    - For each $B \to X_1 X_2 \cdots X_n$ add $A \to X_1 X_2 \cdots X_n$
    - Remove $A \to B$

Definition (Nullable)

The set of all non-terminals that derive ϵ is the set of nullables.
A is nullable if

- $A \to \epsilon$
- $A \to X_1 X_2 \cdots X_n$ if $X_1, X_2, \cdots, X_n$ are nullables

Definition (Unit rule)

$A \to B$

Example Chomsky Normal Form

English Language Example

Push Down Automata

Push Down Automata

- A new computational model.
- Similar to NFA but they have an extra: **Stack**
  - extra memory
  - allows PDA to recognize non-regular languages
- PDA are equivalent to CFG

- PDA can write symbols at the top of the stack and read them back later
- PDA can read and remove the top symbol of the stack at any time.
- Pushing and popping in a LIFO structure
- Stack provides storage to unlimited ammount of information.

Example

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

- A is recognized by a PDA.
- How does the PDA recognize A?
  - Push each leading 0
  - If you read 1, pop a 0 from the stack.
  - If reading the input is done when the stack is empty, accept the string. Reject otherwise.

Non-deterministic PDA

- There are deterministic PDA and non-deterministic PDA.
- They are **NOT** equivalent in expressive power
- NPDA are equivalent to CFG.

PushDown Automata

Definition

A **push down automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is a finite set of **states**
- Σ is a finite **alphabet**
- Γ is a finite **stack alphabet**
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**

Transition Function

The move is determined by

- current state
- next input symbol
- top symbol of the stack

In other words, read a symbol from the stack and a symbol from input and move.
We may have several possible moves, i.e.m non-determinism.

Computation

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts an input $w = w_1 w_2 \dots w_m \in (\Sigma_\epsilon^m)$ if there exist a sequence of states $r_0, r_1, \dots r_m \in Q$ and strings $s_0, s_1, \dots s_m \in \Gamma^*$ satisfying the following conditions:

- $r_0 = q_0$ and $s_0 = \epsilon$
- For $i \in [m-1]$ we have that $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$
- $r_m \in F$

In other words, read a symbol from the stack and a symbol from input and move.
We may have several possible moves, i.e.m non-determinism.

Example

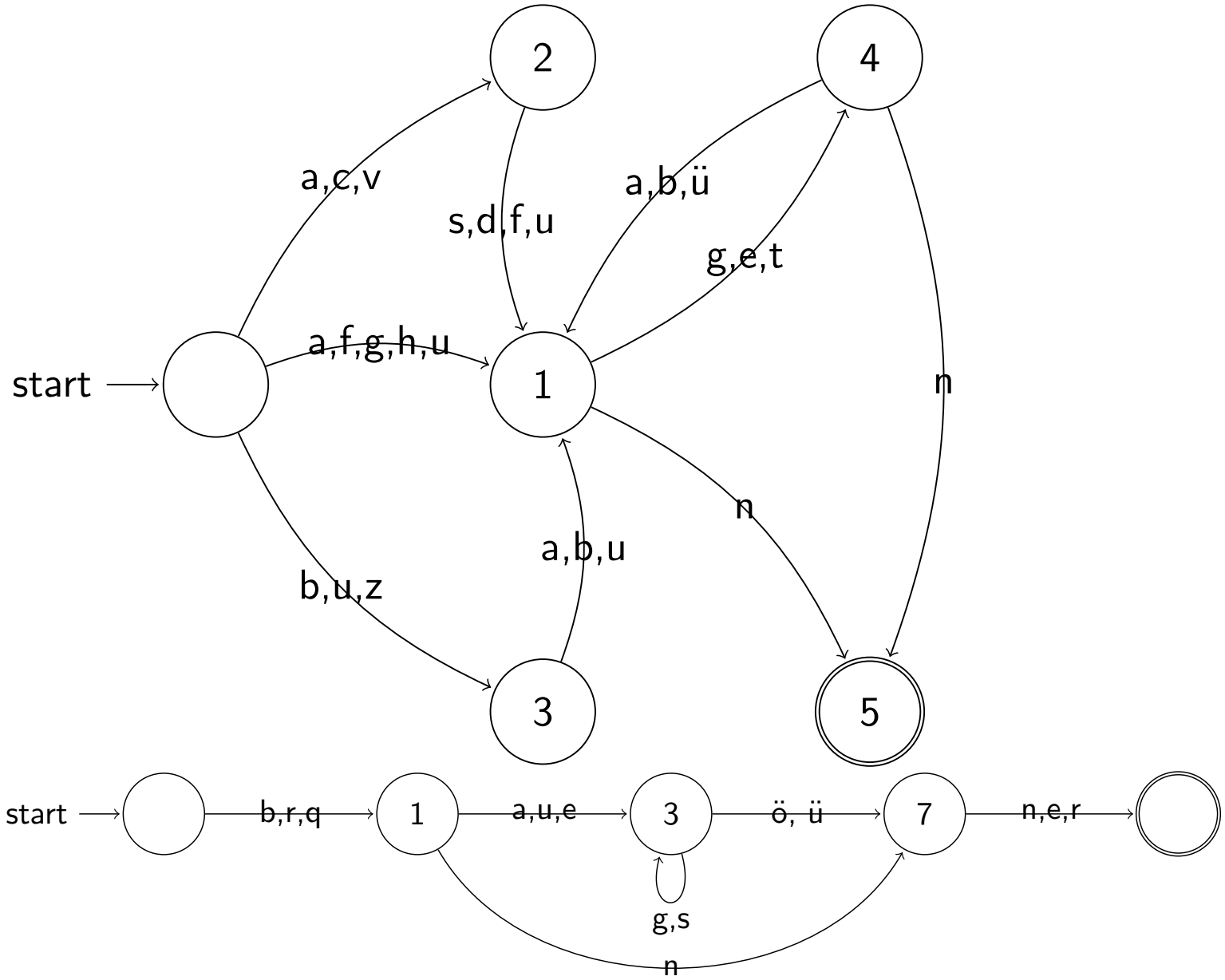
Consider the language $A = \{0^n 1^n : n \geq 0\}$.

Example

Consider the language $A = \{0^n 1^n : n \geq 0\}$.

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, \#\}$
- q_1 is the start state
- $F = \{q_1, q_4\}$

Quiz


2
4
a,c,v
s,d,f,u
a,b,ü
g,e,t
start
a,f,g,h,u
1
n
n
a,b,u
b,u,z
3
5
start
b,r,q
1
a,u,e
3
ö, ü
7
n,e,r
g,s
n

Happy π Day

Push Down Automata

Non-deterministic PDA

- There are deterministic PDA and non-deterministic PDA.
- They are **NOT** equivalent in expressive power
- NPDA are equivalent to CFG.

PushDown Automata

Definition

A **push down automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is a finite set of **states**
- Σ is a finite **alphabet**
- Γ is a finite **stack alphabet**
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**

Example

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$

Example

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$

- Read a's and push them in the stack
- Match their number with b's or c's
  - But which one? b or c?
  - Non-determinism enters the game
    - Try both in parallel
    - Accept if one accepts
  - Without non-determinism it is not possible to recognize the language.

Example

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$

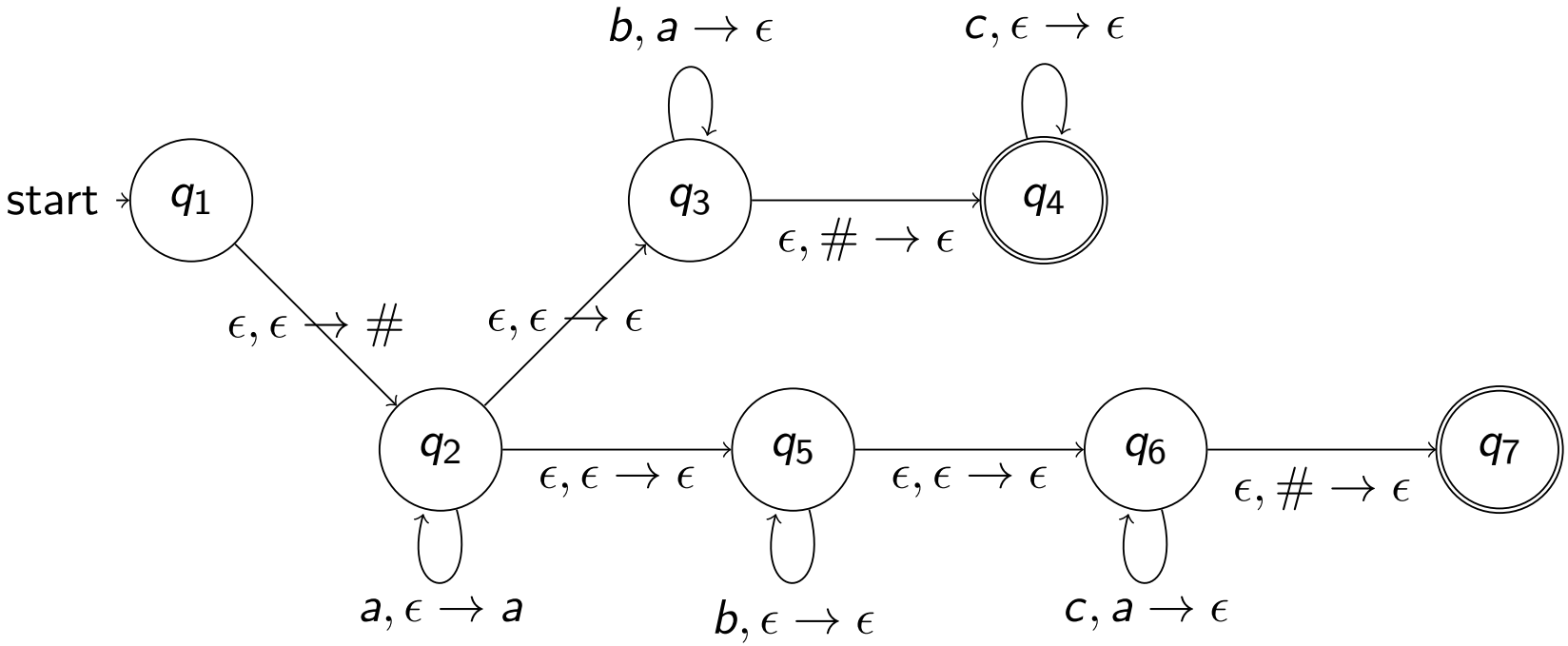
- Read a's and push them in the stack
- Match their number with b's or c's
  - But which one? b or c?
  - Non-determinism enters the game
    - Try both in parallel
    - Accept if one accepts
  - Without non-determinism it is not possible to recognize the language.
- NPDA are equivalent to CFG.

Example

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$

Example

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$



Palindromes

$$\left\{ ww^R : w \in \{0,1\}^* \right\}$$

Palindromes

$$\left\{ ww^R : w \in \{0,1\}^* \right\}$$

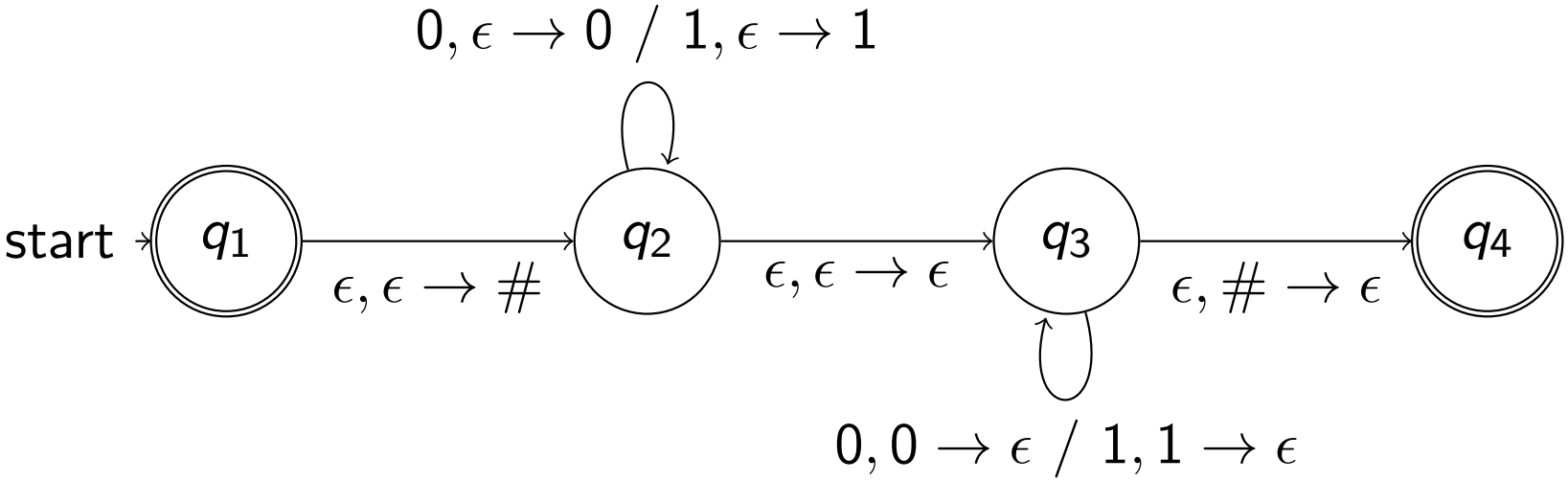
- Read symbols and push them in the stack
- At each step, non-deterministacally, guess the middle of the string
  - Start popping
  - Check if the input symbol and the popped symbol agree
- Accept if
  - They always agreed
  - The stack is emptied the moment the input finished

Palindromes

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$

Palindromes

$$\left\{ a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \right\}$$



Equivalence

Theorem

A language is context-free if and only if some pushdown automaton recognizes it.

Proof idea:

- Let A be a CFL
  - There exists a CFG G generating A
  - Convert G to a PDA P
- Let P be a PDA
  - Create a CFG G generating all string that P accepts
  - Then $L(P)$ is a CFL

CFL $\rightarrow$ PDA

Proof idea:

- PDA accepts w if there is a rerication of w
  - Each step of the derivation yields an intermediate sting of variables and terminals

CFL $\rightarrow$ PDA

Proof idea:

- PDA accepts w if there is a rerication of w
  - Each step of the derivation yields an intermediate sting of variables and terminals
- PDA start by writing the start variable in the stack

CFL $\rightarrow$ PDA

Proof idea:

- PDA accepts w if there is a rerication of w
  - Each step of the derivation yields an intermediate sting of variables and terminals
- PDA start by writing the start variable in the stack
- Performs substitutions one after another, going through a series of intermediate strings.
  - non-determinism needed for selecting the correct substitution
- It may arrive at a string containing only terminals.

CFL $\rightarrow$ PDA

Question

How does the PDA store intermediate strings?

CFL $\rightarrow$ PDA

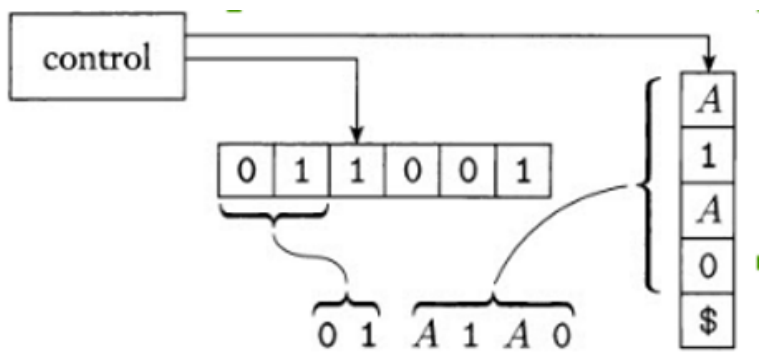
Question

How does the PDA store intermediate strings?

- Stack cannot be used directly
- The symbol at the top may be a terminal

Solution

- Keep only part of the intermediate string in the stack
- Terminals before the first variable are matched immediatelly



CFL $\rightarrow$ PDA

Infromal Description

- Push # and the start variable on the stack
- Repeat forever
  - If the top of the stack is a variable A, select non-deterministacally one of the rules for A and substitute
  - If the top of the stack is a terminal a, read the next input symbol and compare to a. If they don't match reject.
  - If the top of the stack is # enter the accept state. (It will accept if all the input is consumed)

Shorthand Notation

- Let q and r be states of the PDA
- Let $a \in \Sigma_\epsilon$ and $u \in \Gamma^*$

Then, transition $\delta(q, a, s)$ contains (r, u) if we can go from q to r while reading a, popping s and pushing $u = u_1 u_2 \dots u_\ell$.

Shorthand Notation

- Let q and r be states of the PDA
- Let $a \in \Sigma_\epsilon$ and $u \in \Gamma^*$

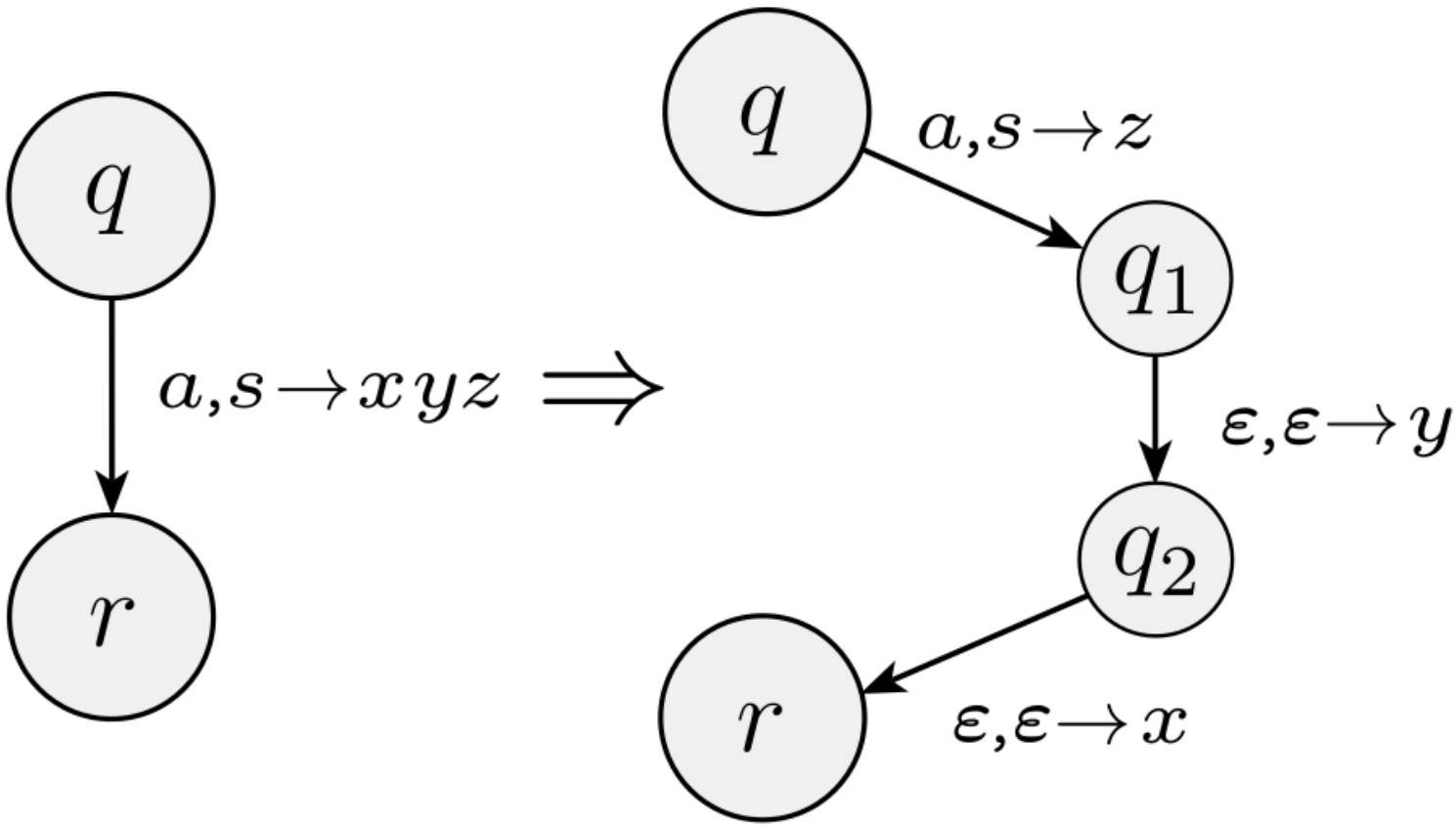
Then, transition $\delta(q, a, s)$ contains (r, u) if we can go from q to r while reading a, popping s and pushing $u = u_1 u_2 \dots u_\ell$.

Implementation

By introducing new states $q_1, q_2, \dots, q_{\ell-1}$

- $\delta(q, a, s)$ contains (q_1, u_ℓ)
- $\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{\ell-1})\}$
- $\delta(q_2, \epsilon, \epsilon) = \{(q_3, u_{\ell-2})\}$
- $\vdots$
- $\delta(q_{\ell-1}, \epsilon, \epsilon) = \{(r, u_1)\}$

$(r, xyz) \in \delta(q, a, s)$



CFL $\rightarrow$ PDA

Formal Description

Let $P = (Q, \Sigma, \Gamma, \delta, q_1, F)$.

- Push # and the start variable on the stack
- Repeat forever
  - If the top of the stack is a variable A, select non-deterministically one of the rules for A and substitute
  - If the top of the stack is a terminal a, read the next input symbol and compare to a. If they don't match reject.
  - If the top of the stack is # enter the accept state. (It will accept if all the input is consumed)

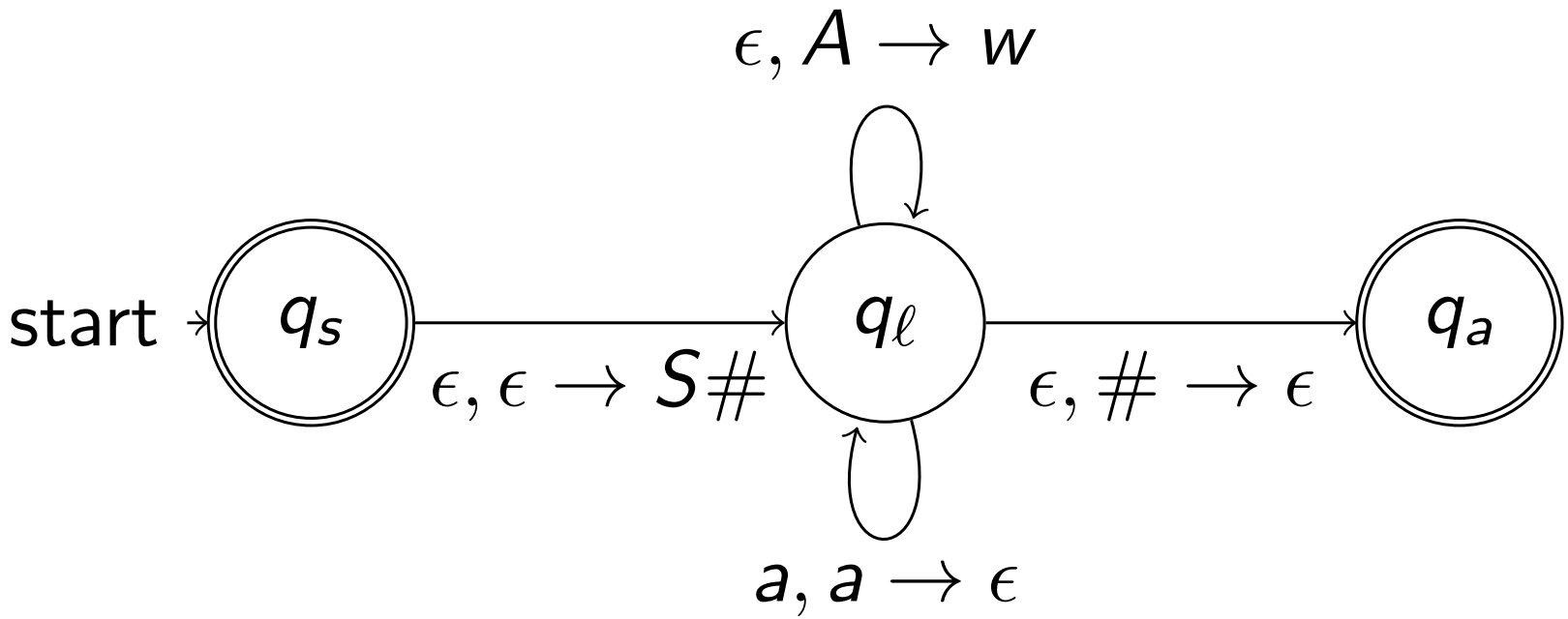
CFL $\rightarrow$ PDA

Given a CFL with start variable S and rules R.

Proof

- States: $Q = \{q_s, q_\ell, q_a\} \cup E$
  - E are the extra states for implementing shorthand notation
- Start state: q_s
- Accept states: $\{q_a\}$
- Transition function
  - Step 1 of the informal description
    - $\delta(q_s, \epsilon, \epsilon) = \{q_\ell, S\#\}$
  - Step 2 of the informal description
    - $\delta(q_\ell, \epsilon, A) = \{q_\ell, w\}$
      (top of the stack contains a variable, $A \rightarrow w \in R$)
    - $\delta(q_\ell, a, a) = \{q_\ell, \epsilon\}$
      (top of the stack contains a terminal)
    - $\delta(q_\ell, \epsilon, \#) = \{q_a, \epsilon\}$
      (the stack is empty)

CFL $\rightarrow$ PDA

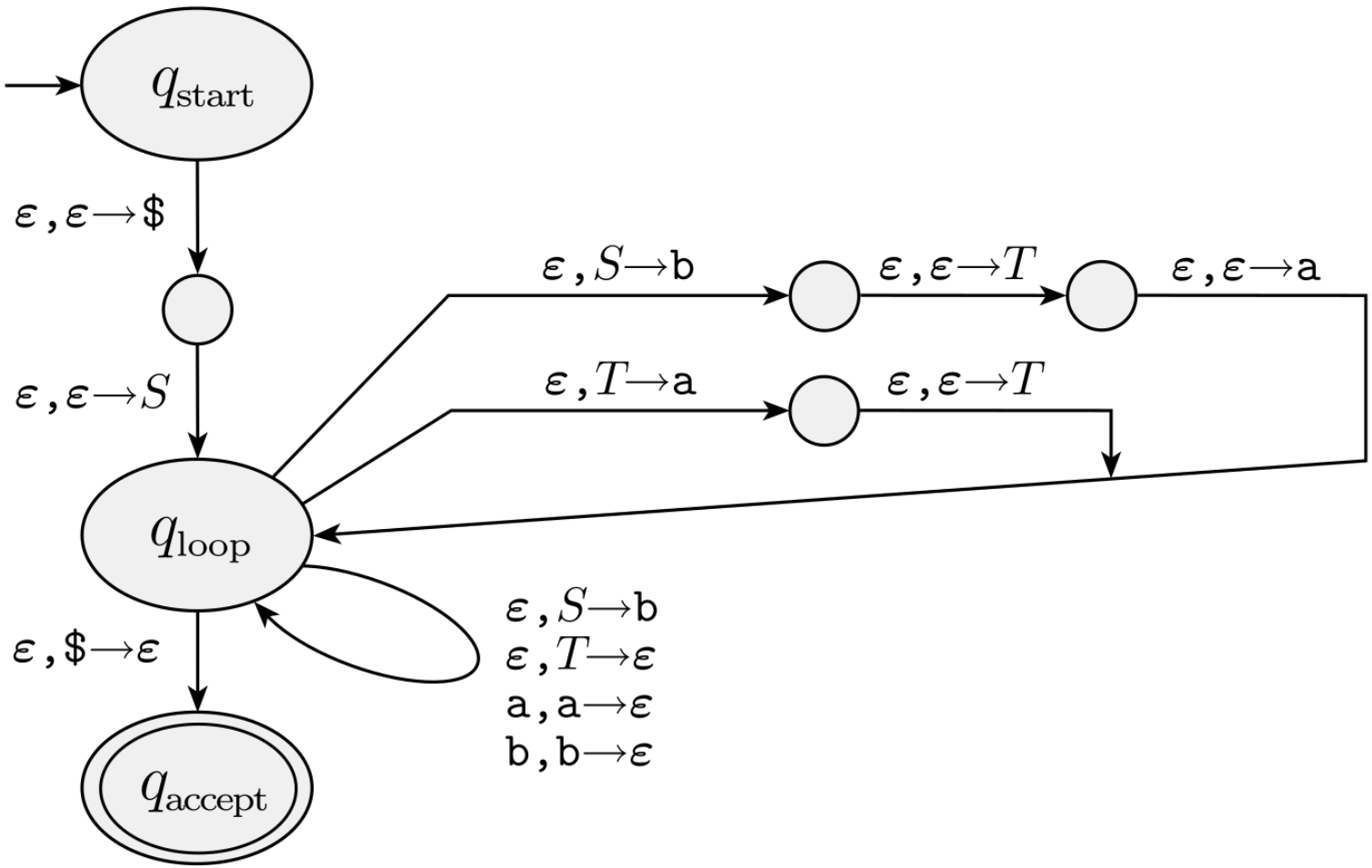


Example

Let $G = \{S \rightarrow aTb|b, T \rightarrow Ta|\epsilon\}$.

Example

Let $G = \{S \to aTb|b, T \to Ta|\epsilon\}$.



PDA $\rightarrow$ CFL

Simplify P

- It has a single accept state q_a
- Empties the stack before accepting
- Each transition either pushes or pops a symbol but not both

PDA $\rightarrow$ CFL

Simplify P

- It has a single accept state q_a
- Empties the stack before accepting
- Each transition either pushes or pops a symbol but not both

Question

How to satisfy the first two conditions?

PDA $\rightarrow$ CFL

Simplify P

- It has a single accept state q_a
- Empties the stack before accepting
- Each transition either pushes or pops a symbol but not both

Question

How to satisfy the first two conditions?

Question

How to satisfy the third condition?

- If a transition simultaneously pushes and pops, replace it with a sequence of two transitions, one pushing and one popping
- If a transition neither pushes nor pops, replace it with a sequence of two transitions, one pushing and one popping an arbitrary symbol

PDA $\rightarrow$ CFL

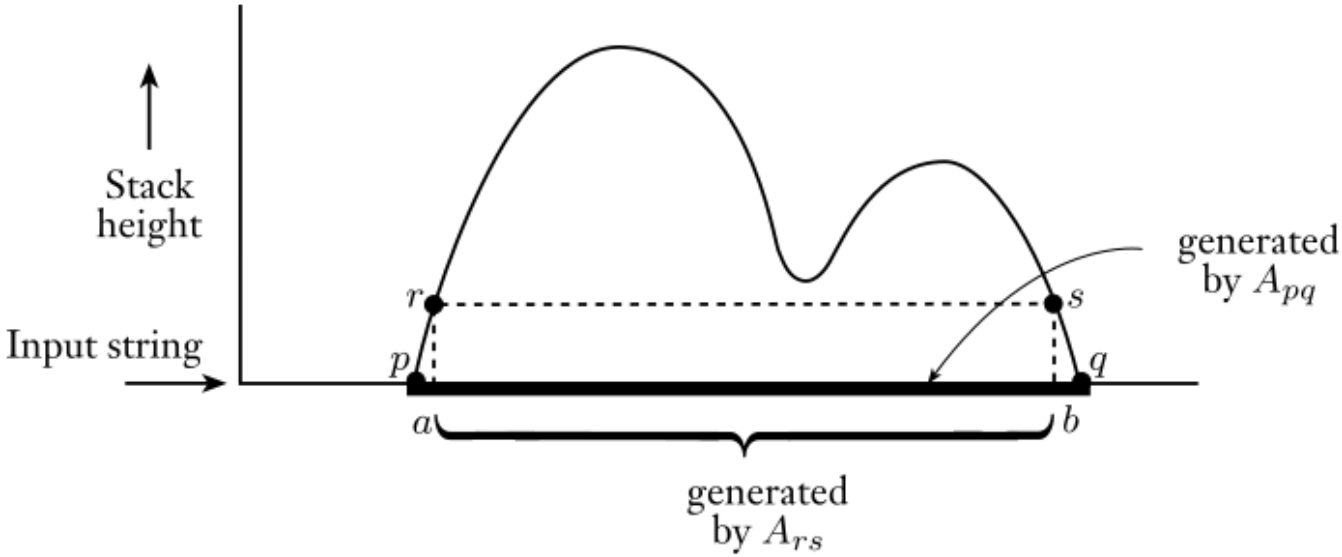
Design a CFG G

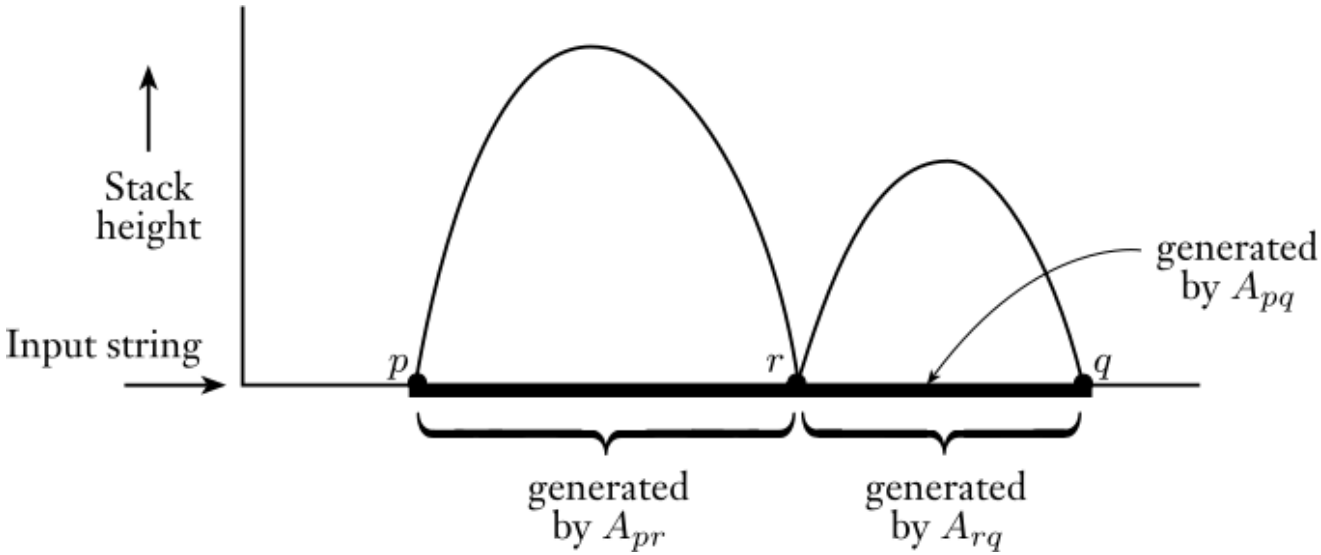
- For each pair of states $p, q \in P$, create a variable A_{pq}
- A_{pq} generates all strings that take P from p to q, starting and ending with same stack.

Computation of P for w

- P's first move on w is a push, last move on w a pop
- During the computation we have two options
  - The last popped symbol is the first pushed symbol, then $A_{pq} = aA_{rs}b$
    - a is the input read in the first move and b in the last
    - r is the state following p and s the state preceding q
  - The first pushed symbol gets popped before the end of w $A_{pq} = A_{pr}A_{rq}$
    - r is the state when the stack empties

PDA $\rightarrow$ CFL





PDA $\rightarrow$ CFL

Construct the grammar

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$

- Variables: $\{A_{pq} : p, q \in Q\}$
- Rules:
  - For each $p, q, r, s \in Q$, $t \in \Gamma$ and $a, b \in \Sigma_\epsilon$, if $\delta(p, a, \epsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ϵ), add the rule $A_{pq} \rightarrow aA_{rs}b$
  - For each $p, q, r \in Q$ add the rule $A_{pq} \rightarrow A_{pr}A_{rq}$
  - For each $p \in Q$ add the rule $A_{pp} \rightarrow \epsilon$

PDA $\rightarrow$ CFL

Construct the grammar

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$

- Variables: $\{A_{pq} : p, q \in Q\}$
- Rules:
  - For each $p, q, r, s \in Q, t \in \Gamma$ and $a, b \in \Sigma_\epsilon$, if $\delta(p, a, \epsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ϵ), add the rule $A_{pq} \rightarrow aA_{rs}b$
  - For each $p, q, r \in Q$ add the rule $A_{pq} \rightarrow A_{pr}A_{rq}$
  - For each $p \in Q$ add the rule $A_{pp} \rightarrow \epsilon$

Claim

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

PDA $\rightarrow$ CFL

Proof by induction on the length of the derivation

- Basis: The derivation has 1 step
  - Uses a rule with no variables in the right-hand side.
  - $A_{pp} \rightarrow \epsilon$ (the only such rules)
  - Input ϵ takes P from p with empty stack to q with empty stack
- Induction step: True for derivations of length up to $k \geq 1$
  - Let $A_{pq} \Rightarrow x$ with $k+1$ steps
  - Derivation's first step either $A_{pq} \rightarrow aA_{rs}b$ or $A_{pq} \rightarrow A_{pr}A_{rq}$.
  - If first step $A_{pq} \rightarrow aA_{rs}b$
    - Let y be the part of x such that $x = ayb$
    - $A_{rs} \Rightarrow y$ in k steps, thus P can go from r w.e.s. to s w.e.s.
    - Since $A_{pq} \rightarrow aA_{rs}b$ is a rule, $(r, t) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$ for some $t \in \Gamma$.
    - P starts w.e.s., after reading a pushes t. Reading y goes to s with t in the stack. Reading b goes to q w.e.s
  - If first step $A_{pq} \rightarrow A_{pr}A_{rq}$
    - Let y and z be the parts of x such that $x = yz$
    - $A_{pr} \Rightarrow y$ and $A_{rq} \Rightarrow z$ in at most k steps, thus P can go from p w.e.s. to r w.e.s. and from r w.e.s. to q w.e.s.

PDA $\rightarrow$ CFL

Claim

If x can bring P from p with empty stack to p with empty stack then A_{pq} generates x.

PDA $\rightarrow$ CFL

Proof by induction on the length of the computation

- Basis: The computation has 0 steps
  - Starts and ends at p. We must show $A_{pp} \rightarrow x$.
  - In 0 steps, P can only read the empty string, thus $x = \epsilon$
  - $A_{pp} \rightarrow \epsilon$ is a rule of G
- Induction step: True for computations of length up to $k \geq 0$
  - P has a computation brings p w.e.s to p w.e.s in $k + 1$ steps
  - The stack may be empty in the middle of the computation
  - If stack is empty only at beginning and end of computation
    - First symbol pushed is the same as last popped, call it t
    - $A_{pq} \rightarrow aA_{rs}b$ is in G
    - If $x = ayb$, then y can bring P from r to s without touching t
    - P can go from p to q w.e.s on input y.
    - From induction hypothesis: $A_{rs} \Rightarrow y$, thus $A_{pq} \Rightarrow x$
  - The stack is empty in state r, where $r \neq p, r \neq q$
    - From p to r and r to q at most k steps.
    - y and z are the inputs for the two parts
    - From induction hypothesis: $A_{pr} \Rightarrow y$ and $A_{rq} \Rightarrow z$.
    - Since $A_{pq} \rightarrow A_{pr}A_{rq}$, we have $A_{pq} \Rightarrow x$

Equivalence

Theorem

A language is context-free if and only if some pushdown automaton recognizes it.

Hierarchy