cse581
Computer Science Fundamentals: Theory

Professor Anita Wasilewska

TCB - LECTURE 4

FINITE AUTOMATA and REGULAR LANGUAGES

TCB - Finite Automata and Languages

General Problem

Given a language L over Σ and a word $w \in \Sigma^*$,

HOW to RECOGNIZE whether

$$w \in L \quad \text{or} \quad w \notin L$$

SOLUTION

Automata as LANGUAGE RECOGNITION device

TCB - Finite Automata and Languages

We consider three classes of AUTOMATA

1. Deterministic Finite Automata DFA
2. Nondeterministic Finite Automata NDFA
3. **Push Down Automata** PDA

We examine **relationships** between them and between two Classes of **Languages**

L1. Regular Languages

L1.**Context - Free Languages**

to be **RECOGNIZED** by them, respectively

TCB - Finite Automata and Languages

We first prezent **Theorems** describing **EQUIVALENCE** of Deterministic Finite Automata DFA and Nondeterministic Finite Automata NDFA which **defines** the notion of Finite Automata as any of them.

Next, we present **Theorems** describing relationship between Finite Automata and Regular Languages and discuss **methods** of proving that a language **is** or **is not** Regular

TCB - Finite Automata and Languages

Similar **Theorems** describing relationship between Push Down Automata and Context -Free Languages and and particular **methods** of proving that a given language **is** or **is not** Context -Free will be covered in Lectures 5 , 6

B2 Chapter 2

PART 1: Finite Automata and Regular Languages

Deterministic and Non Deterministic Automata, Automata Equivalency THEOREMS, Closure THEOREMS, Finite Automata and Regular Languages MAIN THEOREM

PART 2: Regular Languages and non-Regular Languages

PART 3: PUMPING LEMMA for Regular Languages

Finite Automata and Regular Languages

PART 1: Finite Automata and Regular Languages

Deterministic and Non Deterministic Automata, Automata Equivalency THEOREMS, Closure THEOREMS, Finite Automata and Regular Languages MAIN THEOREM

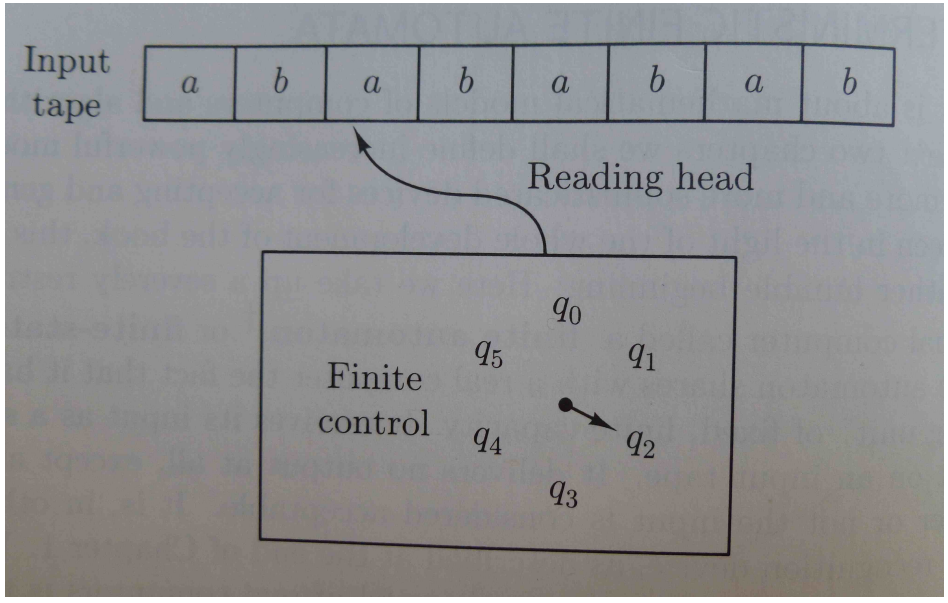
Finite Automata and Regular Languages

PART 1: Deterministic Finite Automata DFA

Deterministic Finite Automata DFA

Simple Computational Model

Here is a picture



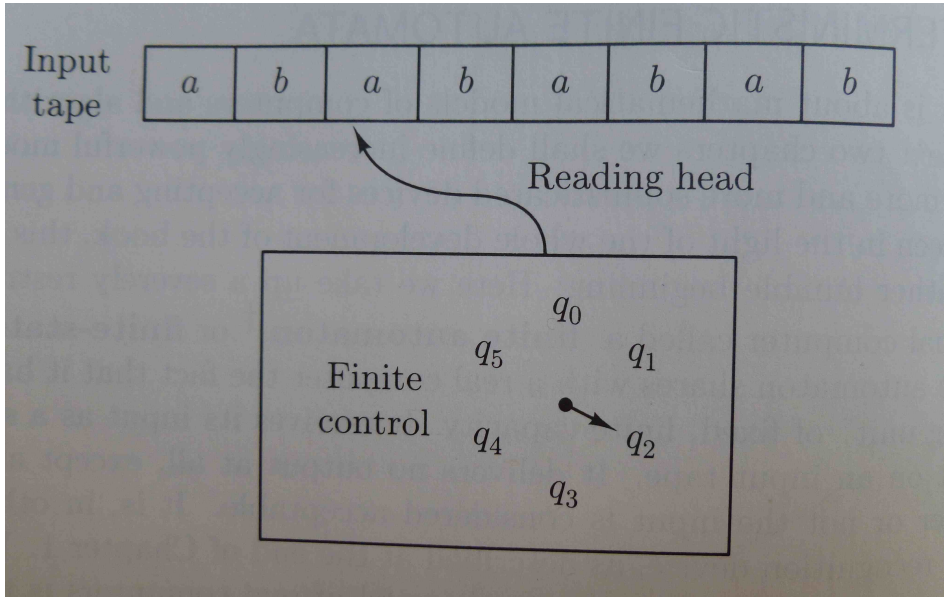
Here are the **components** of the model

C1: Input string on an input tape written at the beginning of the tape

The input tape is divided into squares, with **one symbol** inscribed in each tape square

DFA - A Simple Computational Model

Here is a picture



C2: "Black Box" - called **Finite Control**

It can be in any specific time in **one** of the finite number of **states** $\{q_1, \ldots, q_n\}$

C3: A movable **Reading Head** can sense what symbol is written in any position on the input tape and **moves** only **one square** to the right

DFA - A Simple Computational Model

Here are the **assumptions** for the model

A1: There is no output at all;

A2: DFA **indicates** whether the input is acceptable or not acceptable

A3: DFA is a language **recognition** device

DFA - A Simple Computational Model

Operation of DFA

O1 Initially the **reading head** is placed at left most square at the beginning of the tape and

O2 **finite control** is set on the **initial state**

O3 After reading on the input symbol the **reading head** moves one square to the right and enters a new state

O4 The process is **repeated**

O5 The process **ends** when the reading head reaches the end of the tape

DFA - A Simple Computational Model

The general rules of the operation of DFA are

R1 At regular intervals DFA **reads** only one symbol at the time from the input tape and **enters** a new state

R2: The **move** of DFA depends only on the **current** state and the **symbol** just read

DFA - A Simple Computational Model

Operation of DFA

O6 When the process **stops** the DFA indicates its approval or disapproval of the string by means of the **final state**

O7 If the process **stops** while being in the **final state**, the string is accepted

O8 If the process **stops** while not being in the **final state**, the string is not accepted

Language Accepted by DFA

Informal Definition

Language accepted by a Deterministic Finite Automata is equal to the set of strings accepted by it

DFA - Mathematical Model

To build a mathematical model for DFA we need to include and define the following components

FINITE set of STATES

ALPHABET Σ

INITIAL state

FINAL state

Description of the MOVE of the reading **head** is as follows

R1 At regular intervals DFA **reads** only one symbol at the time from the input tape and **enters** a new state

R2: The MOVE of DFA depends **only** on the current state and the symbol just **read**

DFA - Mathematical Model

Definition

A Deterministic Finite Automata is a quintuple

$$M = (K,\ \Sigma,\ \delta,\ s,\ F)$$

where

K is a finite set of **states**

Σ as an **alphabet**

$s \in K$ is the **initial state**

$F \subseteq K$ is the set of **final states**

δ is a function

$$\delta :\ K \times \Sigma \ \longrightarrow \ K$$

called the **transition function**

We usually use different symbols for $K,\ \Sigma$, i.e. we have that $K \cap \Sigma = \emptyset$

DFA Definition

Definition revisited

A Deterministic Finite Automata is a quintuple

$$M = (K,\ \Sigma,\ \delta,\ s,\ F)$$

where

K is a finite set of **states**

$K \neq \emptyset$ because $s \in K$

Σ as an **alphabet**

Σ can be $\emptyset$ - case to consider

$s \in K$ is the **initial state**

$F \subseteq K$ is the set of **final states**

F can be $\emptyset$ - case to consider

δ is a function

$$\delta :\ \ K \times \Sigma \ \ \longrightarrow \ \ K$$

called the **transition function**

Transition Function

Given DFA

$$M = (K,\ \Sigma,\ \delta,\ s,\ F)$$

where

$$\delta :\ \ K \times \Sigma \ \longrightarrow \ K$$

Let

$$\delta(q, \sigma) = q' \quad \text{for} \quad q,\ q' \in K, \quad \sigma \in \Sigma$$

means: the automaton M in the state q **reads** $\sigma \in \Sigma$ and **moves** to a state $q' \in K$, which is uniquely determined by state q and σ just **read**

Configuration

In order to define a notion of computation of M on an input string $w \in \Sigma^*$ we introduce first a notion of a **configuration**

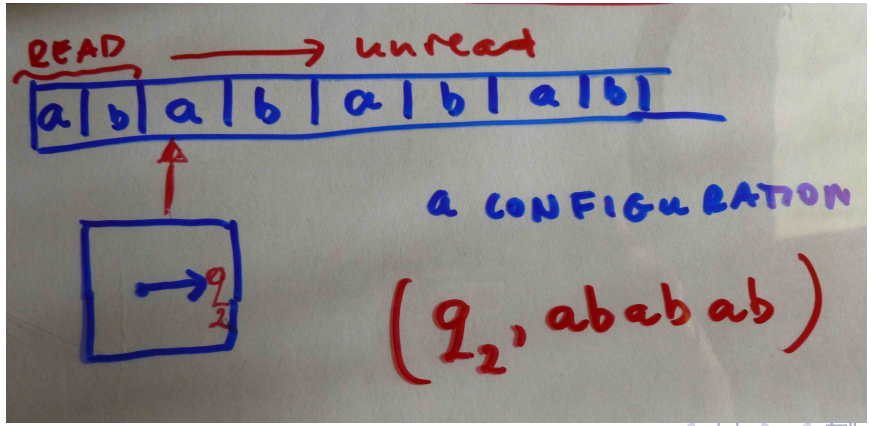
Definition

A configuration is any tuple

$$(q, w) \in K \times \Sigma^*$$

where $q \in K$ represents a **current** state of M and $w \in \Sigma^*$ is **unread part** of the input

Picture



Transition Relation

Definition

The set of all possible configurations of $M = (K,\ \Sigma,\ \delta,\ s,\ F)$ is just

$$K \times \Sigma^* = \{(q, w) : \quad q \in K, \quad w \in \Sigma^*\}$$

We **define** move of an automaton M i in terms of a **transition relation**

$$\vdash_M$$

The **transition relation** acts between two **configurations** and hence $\vdash_M$ is a certain binary relation defined on $K \times \Sigma^*$, i.e.

$$\vdash_M \ \subseteq \ (K \times \Sigma^*)^2$$

Formal definition follows

Transition Relation

Definition (Transition relation)

Given $M = (K,\ \Sigma,\ \delta,\ s,\ F)$

For any $q,\ q' \in K,\quad \sigma \in \Sigma,\quad w \in \Sigma^*$

$$(q, \sigma w) \vdash_M (q', w)$$

if and only if

$$\delta(q, \sigma) = q'$$

Idea of Computation

We use the transition relation to define a move of M along a given input, i.e. a given $w \in \Sigma^*$

Such a move is called a **computation**

Example

Given M such that $K = \{s,\ q\}$ and let $\vdash_M$ be a transition relation such that

$$(s, aab) \vdash_M (q, ab) \vdash_M (s, b) \vdash_M (q, e)$$

We call a **sequence** of configurations

$$(s,\ aab),\ \ (q, ab),\ \ (s, b),\ \ (q, e)$$

a **computation** from (s, aab) to (q, e) in automaton M

Idea of Computation

Given a a **computation**

$$(s,\ aab),\ \ (q, ab),\ \ (s, b),\ \ (q, e)$$

We write this **computation** in a more general form as

$$(q_1, aab),\ \ (q_2, ab),\ \ (q_3, b),\ \ (q_4, e)$$

for $q_1,\ q_2,\ q_3,\ q_4$ being a specific **sequence of states** from $K = \{s,\ q\}$, namely $q_1 = s,\ q_2 =,\ q_3 = s,\ q_4 = q$ and say that the **length** of this computation is 4

In general we write any **computation of length** 4 as

$$(q_1, w_1),\ \ (q_2, w_2),\ \ (q_3, w_3),\ \ (q_4, w_4)$$

for any **sequence** $q_1,\ q_2,\ q_3,\ q_4$ of states from K and words $w_i \in \Sigma^*$

Idea of the Computation

Example

Given M and the **computation**

$$(s, aab),\ \ (q, ab),\ \ (s, b),\ \ (q, e)$$

We say that the word w= aab is **accepted** by M if and only if

1. the **computation** starts when M is in the initial state

- true here as s denotes the **initial state**

2. the whole word w has been read, i.e. the last configuration of the computation is (q, e) for certain state in K,

- true as $K = \{s, q\}$

3. the **computation** ends when M is in the **final state**

- true only if we have that $q \in F$

Otherwise the word w is **not accepted** by M

Definition of the Computation

Definition

Given $M = (K,\ \Sigma,\ \delta,\ s,\ F)$

A sequence of **configurations**

$$(q_1, w_1),\ (q_2, w_2),\ \ldots, (q_n, w_n), \qquad n \geq 1$$

is a computation of the **length** n in M from (q, w) to (q', w')

if and only if

$$(q_1, w_1) = (q, w), \quad (q_n, w_n) = (q', w') \quad \text{and}$$

$$(q_i, w_i) \vdash_M (q_{i+1}, w_{i+1}) \quad \text{for} \quad i = 1, 2, \ldots n-1$$

Observe that when $n = 1$ the computation (q_1, w_1) **always** exists . It is a computation of the length 1, called also a trivial computation

We also write sometimes the computations as

$(q_1, w_1) \vdash_M (q_2, w_2) \vdash_M \ \ldots \ \vdash_M (q_n, w_n)$ for $n \geq 1$

Words Accepted by M

In Plain Words:

A word $w \in \Sigma^*$ is **accepted** by $M = (K, \Sigma, \delta, s, F)$

if and only if

there is a computation such that

1. starts with the word w and M in the **initial state** ,
2. ends when M is in a **final state**, and
3. the whole word w has been read

Language Accepted by M

Definition

We define the language **accepted** by M as follows

$$L(M) = \{w \in \Sigma^* : \quad w \text{ is accepted by } M\}$$

i.e. we write

$$L(M) = \{w \in \Sigma^* : \ (s, w) \vdash_M \ldots \vdash_M (q, e) \text{ for some } q \in F\}$$

Language Accepted by M
DEFINITIOIN

The question: how to write the definition of the L(M) in a more concise and elegant way

Answer: we use the notion (TCB- Lecture 1) of reflexive, transitive closure of $\vdash_M$ denoted by $\vdash_M{}^*$ and write

DEFINITION

$$L(M) = \{w \in \Sigma^* : \ (s, w) \vdash_M{}^* (q, e) \quad \text{for some} \quad q \in F\}$$

Example

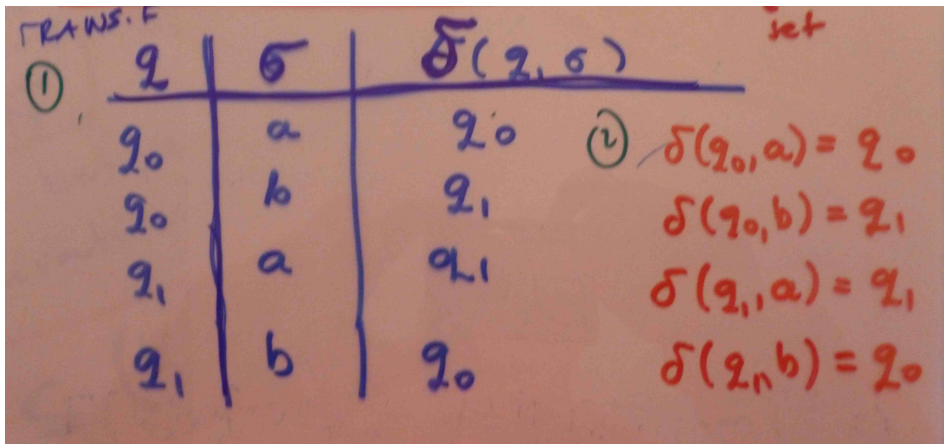
Example

Let $M = (K,\ \Sigma,\ \delta,\ s,\ F)$, where

$K = \{q_0, q_1\}, \quad \Sigma = \{a,\ b\}, \quad s = q_0, \quad F = \{q_0\}$

and the **transition function** $\delta :\ K \times \Sigma \longrightarrow K$

is defined as follows



Question Determine whether $ababb \in L(M)$ or $ababb \notin L(M)$

Examples

Solution

We must evaluate computation that starts with the configuration $(q_0, ababb)$ as $q_0 = s$

$(q_0, ababb) \vdash_M$ use $\delta(q_0, a) = q_0$

$(q_0, babb) \vdash_M$ use $\delta(q_0, b) = q_1$

$(q_1, abb) \vdash_M$ use $\delta(q_1, a) = q_1$

$(q_1, bb) \vdash_M$ use $\delta(q_1, b) = q_0$

$(q_0, b) \vdash_M$ use $\delta(q_0, b) = q_1$

$(q_1, e) \vdash_M$ **end** of computation and $q_1 \notin F = \{q_0\}$

We proved that $ababb \notin L(M)$

Observe that we always get **unique** computations, as δ is a function, hence he name Deterministic Finite Automaton (DFA)

Finite Automata and Regular Languages

PART 1: NON Deterministic Finite Automata NDFA

NDFA: Nondeterministic Finite Automata

Now we add a new powerful feature to the **finite automata**

This feature is called **nondeterminism**

Nondeterminism is essentially the ability to change states in a way that is only **partially determined** by the current state and input symbol, or a string of symbols, empty string included

The automaton, as it reads the input string, may choose at each step to go to any of its states

The choice is not determined by anything in our model , and therefore it is said to be **nondeterministic**

At each step there is always a finite number of choices, hence it is still a **finite automaton**

NDFA - Mathematical Model

Definition

A Nondeterministic Finite Automata is a quintuple

$$M = (K,\ \Sigma,\ \Delta,\ s,\ F)$$

where

K is a finite set of **states**

Σ as an **alphabet**

$s \in K$ is the **initial state**

$F \subseteq K$ is the set of **final states**

Δ is a **finite set** and

$$\Delta \subseteq K \times \Sigma^* \times K$$

Δ is called the **transition relation**

We usually use different symbols for $K,\ \Sigma$, i.e. we have that $K \cap \Sigma = \emptyset$

NDFA Definition

Definition revisited

A Nondeterministic Finite Automata is a quintuple

$$M = (K,\ \Sigma,\ \Delta,\ s,\ F)$$

where

K is a finite set of **states**

$K \neq \emptyset$ because $s \in K$

Σ as an **alphabet**

Σ can be $\emptyset$ - case to consider

$s \in K$ is the **initial state**

$F \subseteq K$ is the set of **final states**

F can be $\emptyset$ - case to consider

Δ is a **finite set** and $\Delta \subseteq K \times \Sigma^* \times K$

Δ is called the **transition relation**

Δ can be $\emptyset$ - case to consider

Some Remarks

R1 We **must** say that Δ is a **finite** set because the set $K \times \Sigma^* \times K$ is countably infinite, i.e. $|K \times \Sigma^* \times K| = \aleph_0$) and we want to have a **finite automata** and we defined it as

$$\Delta \subseteq K \times \Sigma^* \times K$$

R2 The DFA **transition function** $\delta : K \times \Sigma \longrightarrow K$ is (as any function!) a **relation**

$$\delta \subseteq K \times \Sigma \times K$$

R3 The **set** δ is always **finite** as the **set** $K \times \Sigma \times K$ is **finite**

R4 The DFA **transition function** δ is a particular case of the NDFA **transition relation** Δ, hence similarity of notation

Configuration and Transition Relation

Given a NDFA automaton

$$M = (K,\ \Sigma,\ \Delta,\ s,\ F)$$

We define as we did in the case of DFA the notions of a configuration, and a transition relation

Definition

A **configuration** in a NDFA is any tuple

$$(q, w) \in K \times \Sigma^*$$

Configuration and Transition Relation

Definition

A **transition relation** in $M = (K,\ \Sigma,\ \Delta,\ s,\ F)$

defined by the **Class Definition** is a binary relation

$$\vdash_M\ \subseteq\ (K \times \Sigma^*) \times (K \times \Sigma^*)$$

such that $q,\ q'\ \in K,\quad u,\ w \in \Sigma^*$

$$(q, uw)\ \vdash_M\ (q', w)$$

if and only if

$$(q,\ u,\ q') \in \Delta$$

Language Accepted by M

We define, as in the case of the deterministic DFA , the language accepted by the **nondeterministic** M as follows

Definition

$$L(M) = \{w \in \Sigma^* : \ (s, w) \vdash_M{}^* (q, e) \quad \text{for} \quad q \in F\}$$

where $\vdash_M{}^*$ is the reflexive, transitive closure of $\vdash_M$

Finite Automata and Regular Languages

PART 1: Finite Automata and Regular Languages

Automata Equivalency THEOREMS,

Equivalency of Automata

We define now formally an equivalency of automata as follows

Definition

For any two automata M_1, M_2 (deterministic or nondeterministic)

$$M_1 \approx M_2 \quad \text{if and only if} \quad L(M_1) = L(M_2)$$

Now we are going to formulate and prove the main theorem of this part of the Chapter 2, informally stated as

Equivalency Theorem

The notions of a deterministic and a non-dederteministic automata are **equivalent**

Equivalency of Automata Theorems

The automata **Equivalency Theorem** consists of two **Equivalency Theorems**

Equivalency Theorem 1

For any **DFA** M, there is is a **NDFA** M', such that $M \approx M'$, i.e. such that

$$L(M) = L(M')$$

Equivalency Theorem 2

For any **NDFA** M, there is is a **DFA** M', such that $M \approx M'$, i.e. such that

$$L(M) = L(M')$$

Equivalency of Automata Theorems

Equivalency Theorem 1

For any **DFA** M, there is is a **NDFA** M', such that $M \approx M'$, i.e. such that

$$L(M) = L(M')$$

Proof

Any **DFA** M is a particular case of a **DFA** M' because any function δ is a relation

Moreover δ and its a particular case of the relation Δ as $\Sigma \subseteq \Sigma \cup \{e\}$ (for the Book Definition) and $\Sigma \subseteq \Sigma^*$ (for the Class Definition)

This ends the **proof**

Equivalency of Automata Theorems

Equivalency Theorem 2

For any **NDFA** M, there is is a **DFA** M', such that $M \approx M'$, i.e. such that

$$L(M) = L(M')$$

Proof

The proof is far from trivial. It is a constructive proof;

It describes, given a **NDFA** M, a general method of construction step by step of an **DFA** M' that accepts the came language as M.

Finite Automata

Hence by the automata **Equivalency Theorems** the deterministic and non deterministic automata ways are **equivalent** and we use a name

FINITE AUTOMATA

when we talk about **deterministic** or **non deterministic** automata

Finite Automata and Regular Languages

PART 1: Finite Automata and Regular Languages Closure THEOREMS, Finite Automata and Regular Languages MAIN THEOREM

Automata Closure Theorem

In order to prove the MAIN THEOREM that establishes a relationship between Finite Automata and Regular languages prove and use the following

Automata CLOSURE THEOREM

The class of languages accepted by **Finite Automata** (FA) is **closed** under the following operations

1. union
2. concatenation
3. Kleene's Star
4. complementation
5. intersection

Observe that we used the term **Finite Automata** (FA) so in the **proof** we can choose a DFA or a NDFA, as we have already proved their **equivalency**

Automata - Languages MAIN THEOREM

A language L is regular if and only if it is accepted by a finite automaton, i.e.

A language L is regular if and only if there is a finite automaton M, such that

$$L = L(M)$$

Regular Languages Closure Theorem

Directly from the the Automata - Languages **Main Theorem** and Automata **Closure Theorem** we get the following

Regular Languages Closure Theorem

The class of REGULAR languages

is closed under the following operations

1. union
2. concatenation
3. Kleene's Star
4. complementation
5. intersection

Regular Languages

PART 2: Regular Languages and non-Regular Languages

Languages that are Not Regular

We know that there are **uncountably** many and exactly C of all languages over any alphabet $\Sigma \neq \emptyset$

We also know that there are only $\aleph_0$, i.e. **infinitely countably** many regular languages

It means that we have **uncountably** many and . exactly C languages that **are not** regular

Reminder

A language $L \subseteq \Sigma^*$ is **regular** if and only if there is a regular expression $r \in \mathcal{R}$ that represents L, i.e. such that

$$L = \mathcal{L}(r)$$

Regular or not Regular Languages

We look now at some simple examples of languages that might be, or not be **regular**

E1 The language $L_1 = a^*b^*$ is **regular** because is defined by a regular expression

E2 The language

$$L_2 = \{a^n b^n : \quad n \geq 0\} \subseteq L_1$$

is not **regular**

We will **prove** prove it using a very important theorem to be proved that is called **Pumping Lemma**

Regular or not Regular Languages

Intuitively we can see that

$$L_2 = \{a^n b^n : \quad n \geq 0\}$$

can't be regular as we can't construct a **finite automaton** accepting it

Such automaton would need to have something like a **memory** to store, count and compare the number of a's with the number of b's

Regular or not Regular Languages

We will define and study in Chapter 3 a new class of **automata** that would accommodate the "memory" problem

They are called **Push Down Automata**

We will **prove** that they accept a larger class of languages, called context free languages

Regular or not Regular Languages

E3 The language $L_3 = a^*$ is **regular** because is defined by a regular expression

E4 The language $L_4 = \{a^n : \quad n \geq 0\}$ is **regular** because in fact $L_3 = L_4$

E5 The language $L_4 = \{a^n : \quad n \in Prime\}$ is **not regular**
We will **prove** it using Pumping Lemma

Regular or not Regular Languages

E6 The language $L_6 = \{a^n : \ n \in EVEN\}$ is **regular** because in fact $L_6 = (aa)^*$

E7 The language

$L_7 = \{w \in \{a, b\}^* : w$ has an equal number of a' s and b's $\}$

is **not regular**

Proof Assume that L_7 is **regular**

We know that $L_1 = a^*b^*$ is regular, hence the language

$L = L_7 \cap L_1$ is **regular,** by the **Closure Theorem:** the class of regular languages is closed under intersection

But obviously, $L = \{a^n b^n : n \in N\}$ and was proved in **E2** to be not regular

This **contradiction** proves that L_7 is **not regular**

Regular and not Regular Languages

E8 The language $L_8 = \{ww^R : \quad w \in \{a, b\}^*\}$

is **not regular**

We prove it using Pumping Lemma

E9 The language $L_9 = \{ww : \quad w \in \{a, b\}^*\}$

is **not regular**

We prove it using Pumping Lemma

Regular or not Regular Languages

E10 The language $L_{10} = \{wcw : \quad w \in \{a, b\}^*\}$

is **not regular**

We prove it using Pumping Lemma

E11 The language $L_{11} = \{w\overline{w} : \quad w \in \{a, b\}^*\}$

where $\overline{w}$ stands for w with each occurrence of a is replaced by b, and vice versa

is **not regular**

We prove it using Pumping Lemma

Regular or not Regular Languages

E12 The language

$$L_{12} = \{xy \in \Sigma^* : \ x \in L \text{ and } y \notin L \text{ for any } \textbf{regular } \ L \subseteq \Sigma^*\}$$

is **regular**

Proof Observe that $L_{12} = L \circ \overline{L}$ where $\overline{L}$ denotes a complement of L, i.e.

$$\overline{L} = \{w \in \Sigma^* : \quad w \in \Sigma^* - L\}$$

L is **regular**, and so is $\overline{L}$, and $L_{12} = L \circ \overline{L}$ is **regular** by the following, already already proved theorem

Closure Theorem The class of Regular Languages is **closed** under $\cup, \cap, -, \circ, ^*$

Regular or not Regular Languages

E13 The language

$$L_{13} = \{w^R : \quad w \in L \quad \text{and} \quad L \quad \text{is regular} \quad \}$$

is **regular**

Definition For any language L we call the language

$$L_R = \{w^R : \quad w \in L\}$$

the reverse language of L

The **E13** says that the following holds

Fact

For any **regular** language L, its reverse language L^R

is **regular**

Regular or not Regular Languages

Fact

For any **regular** language L, its reverse language L^R is **regular**

Proof Let $M = (K, \Sigma, \Delta, s, F)$ be such that L = L(M)

The reverse language L^R is accepted by a finite automata

$$M^R = (K \cup s', \Sigma, \Delta', s', F = \{s\})$$

where $s' \notin K$ and

$$\Delta' = \{(r, w, p) : \quad (p, w, r) \in \Delta, \quad w \in \Sigma^*\} \cup \{(s', e, q) : \quad q \in F\}$$

We used the Lecture Definition of M

Regular and NOT Regular Languages

E14

Any finite language is regular

Proof Let $L \subseteq \Sigma^*$ be a finite language , i.e.

$$L = \emptyset \text{ or } L = \{w_1, w_2, \ldots w_n\} \text{ for } n > 0\}$$

We construct the finite automata M such that

$$L(M) = L = \{w_1\} \cup \{w_2\} \cup \ldots \{w_n\} = L_{w_1} \cup \cdots \cup L_{w_n}$$

as $M = M_{w_1} \cup \cdots \cup M_{w_n} \cup M_{\emptyset}$

Exercises

Exercise 1

Show that the language

$$L = \{xyx^R : \quad x, y \in \Sigma\}$$

is **regular** for any Σ

Exercises

Exercise 1

Show that the language

$$L = \{xyx^R : \quad x, y \in \Sigma\}$$

is **regular** for any Σ

Proof

For any $x \in \Sigma, x^R = x$

Σ is a finite set, hence

$$L = \{xyx : \ x, y \in \Sigma\}$$

is also finite and we just proved that any finite language is **regular**

Exercises

Exercise 2

Show that the class of regular languages **is not closed** with respect to subset relation.

Exercise 3

Given L_1, L_2 regular languages, is $L_1 \cap L_2$ also a regular language?

Exercises

Exercise 2

Show that the class of regular languages **is not closed** with respect to subset relation.

Solution

Consider two languages

$$L_1 = \{a^n b^n : \quad n \in N\} \quad \text{and} \quad L_2 = a^* b^*$$

Obviously, $L_1 \subseteq L_2$ and L_1 is a **non-regular** subset of a regular L_2

Exercise 3

Given L_1, L_2 regular languages, is $L_1 \cap L_2$ also a regular language?

Solution

YES, it is because the class of regular languages is closed under $\cap$

Exercises

Exercise 4

Given L_1, L_2 , such that $L_1 \cap L_2$ is a regular language

Does it imply that both languages L_1, L_2 must be regular?

Exercises

Exercise 4

Given L_1, L_2 , such that $L_1 \cap L_2$ is a regular language

Does it imply that both languages L_1, L_2 must be regular?

Solution

NO, it doesn't. Take the following L_1, L_2

$$L_1 = \{a^n b^n : n \in N\} \quad \text{and} \quad L_2 = \{a^n : n \in Prime\}$$

The language $L_1 \cap L_2 = \emptyset$ is a regular language none of L_1, L_2 is regular

Exercises

Exercise 5

Show that the language

$$L = \{xyx^R : \quad x, y \in \Sigma^*\}$$

is regular for any Σ

Exercises

Exercise 5

Show that the language

$$L = \{xyx^R : \quad x, y \in \Sigma^*\}$$

is regular for any Σ

Solution

Take a case of $x = e \in \Sigma^*$

We get a language

$$L_1 = \{eye^R : \ e, y \in \Sigma^*\} \subseteq L$$

and of course $L_1 = \Sigma^*$ and so $\Sigma^* \subseteq L \subseteq \Sigma^*$

Hence $L = \Sigma^*$ and Σ^* is regular

This proves that L is regular

Exercises

Exercise 6

Given a regular language $L \subseteq \Sigma^*$

Show that the language

$$L_1 = \{xy \in \Sigma^* : \quad x \in L \quad \text{and} \quad y \notin L\}$$

is also regular

Exercises

Exercise 6

Given a regular language $L \subseteq \Sigma^*$

Show that the language

$$L_1 = \{xy \in \Sigma^* : \quad x \in L \quad \text{and} \quad y \notin L\}$$

is also regular

Solution

Observe that $L_1 = L \circ (\Sigma^* - L)$

L is regular, hence $(\Sigma^* - L)$ is regular (closure under complement), and so is L_1 by closure under concatenation

PUMPING LEMMA

PART 3: PUMPING THEOREM for Regular Languages

PUMPING LEMMA

Pumping Lemma

Pumping Lemma is one of a general class of Theorems called **pumping theorems**

They are called **pumping theorems** because they assert the existence of certain points in certain strings where a substring can be repeatedly inserted (pumping) without affecting the acceptability of the string

We present here two versions of the Pumping Lemma

First is the Lecture Notes version from the first edition of the Book and the second is the Book version (page 88) from the new edition

The Book version is a slight generalization of the Lecture version

Pumping Lemma 1

Pumping Lemma 1

Let L be an infinite regular language over $\Sigma \neq \emptyset$

Then **there are** strings $x, y, z \in \Sigma^*$ such that

$$y \neq e \quad \text{and} \quad xy^n z \in L \quad \text{for all} \quad n \geq 0$$

Observe that the Pumping Lemma 1 says that in an infinite regular language L, there is a word $w \in L$ that can be **re-written** as $w = xyz$ in such a way that $y \neq e$ and we "pump" the part y any number of times and still have that such obtained word is still in L, i.e. that $xy^n z \in L$ for all $n \geq 0$

Hence the name Pumping Lemma

Role of Pumping Lemma

We use the Pumping Lemma as a **tool** to carry **proofs** that some languages **are not regular**

Proof METHOD

Given an infinite language L we want to PROVE it to be NOT REGULAR

We proceed as follows

1. We assume that L is REGULAR

2.Hence by Pumping Lemma we get that there is a word $w \in L$ that can be **re-written** as $w = xyz,\ y \neq e$, and $xy^n z \in L$ for all $n \geq 0$

3. We examine the fact $xy^n z \in L$ for all $n \geq 0$

4. If we get a CONTRADICTION we have proved that the language L is **not regular**

Proof of Pumping Lemma 1

Pumping Lemma 1

Let L be an infinite regular language over $\Sigma \neq \emptyset$

Then **there are** strings $x, y, z \in \Sigma^*$ such that

$$y \neq e \quad \text{and} \quad xy^n z \in L \quad \text{for all natural number} \quad n \geq 0$$

Proof

Since L is regular, L is accepted by a deterministic finite automaton

$$M = (K, \Sigma, \delta, s, F)$$

Suppose that M has n states, i.e. $|K| = n$ for $n \geq 1$

Since L is **infinite**, M accepts some string $w \in L$ of length n or greater, i.e.

there is $w \in L$ such that $|w| = \mathbf{k} > \mathbf{n}$ and

$$w = \sigma_1 \sigma_2 \ldots \sigma_k \quad \text{for} \quad \sigma_i \in \Sigma, \quad 1 = 1, 2, \ldots, k$$

Proof of Pumping Lemma 1

Consider a **computation** of $w = \sigma_1\sigma_2\ldots\sigma_k \in L$:

$$(q_0, \sigma_1\sigma_2\ldots\sigma_k) \vdash_M (q_1, \sigma_2\ldots\sigma_k), \vdash_M$$

$$\ldots\ \ldots \vdash_M (q_{k-1}, \sigma_k), \vdash_M (q_k, e)$$

where q_0 is the initial state s of M and q_k is a final state of M

Since $|w| = \mathbf{k} > \mathbf{n}$ and M has only n states, by **Pigeon Hole Principle** we have that

there exist i and j, $0 \leq i < j \leq k$, such that $q_i = q_j$

That is, the string $\sigma_{i+1}\ldots\sigma_j$ is nonempty since $i + 1 \leq j$ and **drives** M from state q_i **back** to state q_i

But then this string $\sigma_{i+1}\ldots\sigma_j$ could be **removed** from w, or we could **insert** any number of its **repetitions** just after just after σ_j and M would still accept such string

Proof of Pumping Lemma 1

We just showed by **Pigeon Hole Principle** we have that M that accepts $w = \sigma_1\sigma_2\ldots\sigma_k \in L$ also **accepts** the string

$$\sigma_1\sigma_2\ldots\sigma_i(\sigma_{i+1}\ldots\sigma_j)^n\sigma_{j+1}\ldots\sigma_k \quad \text{for each} \quad n \geq 0$$

Observe that $\sigma_{i+1}\ldots\sigma_j$ is non-empty string since $i+1 \leq j$

That means that there exist strings

$$\mathbf{x} = \sigma_1\sigma_2\ldots\sigma_i, \quad \mathbf{y} = \sigma_{i+1}\ldots\sigma_j, \quad \mathbf{z} = \sigma_{j+1}\ldots\sigma_k \quad \text{for} \quad y \neq e$$

such that

$$y \neq e \quad \text{and} \quad xy^nz \in L \quad \text{for all} \quad n \geq 0$$

Proof of Pumping Lemma 1

The computation of M that accepts xy^nz is as follows

$$(q_o, xy^nz) \vdash_M^* (q_i, y^nz) \vdash_M^* (q_i, y^{n-1}z)$$

$$\vdash_M^* \ldots \vdash_M^* (q_i, y^{n-1}z) \vdash_M^* (q_k, e)$$

This **ends** the proof

Observe that the proof holds for **any word** $w \in L$ with $|w| \geq n$, where n is the number of states of deterministic M that accepts L

We get hence another version of the **Pumping Lemma 1**

Pumping Lemma 2

Pumping Lemma 2

Let L be an infinite regular language over $\Sigma \neq \emptyset$

Then **there is** an integer $n \geq 1$ such that for **any word** $w \in L$ with lengths greater then n, i.e. $|w| \geq n$ **there are** $x, y, z \in \Sigma^*$ such that w can be re-written as $w = xyz$ and

$$y \neq e \quad \text{and} \quad xy^i z \in L \quad \text{for all} \quad i \geq 0$$

Proof

Since L is regular, it is accepted by a deterministic finite automaton M that has $n \geq 1$ **states**

This is our integer $n \geq 1$

Let w be **any word** in L such that $|w| \geq n$

Such **words exist** as L is infinite

The rest of the proof exactly the same as in case of **Pumping Lemma 1**

Pumping Lemma

We write the **Pumping Lemma 2** symbolically using quantifiers symbols as follows

Pumping Lemma 2

Let L be an **infinite regular** language over $\Sigma \neq \emptyset$

Then the following holds

$$\exists_{n \geq 1} \forall_{w \in L} \big(|w| \geq n \ \Rightarrow$$

$$\exists_{x,y,z \in \Sigma^*} \ (w = xyz \ \cap \ y \neq e \ \cap \ \forall_{i \geq 0} (xy^i iz \in L)) \big)$$

Book Pumping Lemma

Book Pumping Lemma is a STRONGER version of the **Pumping Lemma 2**

It applies to any any regular language, not to an infinite regular language, as the **Pumping Lemmas 1, 2**

Book Pumping Lemma

Book Pumping Lemma

Let L be a regular language over $\Sigma \neq \emptyset$

Then **there is** an integer $n \geq 1$ such that **any word** $w \in L$ with $|w| \geq n$ can be re-written as $w = xyz$ such that

$$y \neq e, \quad |xy| \leq n, \quad x, y, z \in \Sigma^* \quad \text{and} \quad xy^i z \in L \quad \text{for all} \quad i \geq 0$$

Proof The proof goes exactly as in the case of Pumping Lemmas 1, 2

Notice that from the proof of Pumping Lemma 1

$$x = \sigma_1 \sigma_2 \dots \sigma_i, \quad z = \sigma_{j+1} \ \dots \ \sigma_k\} \quad \text{for} \quad 0 \leq i < j \leq n$$

and so by definition $|xy| \leq n$ for n being the **number of states** of the deterministic M that **accepts** L

Pumping Lemma Applications

Use Pumping Lemma to **prove** the following

Fact 1

The language $L \subseteq \{a, b\}^*$ defined as follows

$$L = \{a^n b^n : \quad n > 0\}$$

IS NOT regular

Obviously, L is infinite and we use the Lecture version

Pumping Lemma 1

Let L be an infinite regular language over $\Sigma \neq \emptyset$

Then for **there is** a word $w \in L$ and **there are** $x, y, z \in \Sigma^*$ such that w can be re-written as $w = xyz$ and

$$y \neq e \quad \text{and} \quad xy^n z \in L \quad \text{for all} \quad n \geq 0$$

Pumping Lemma Applications

Reminder: we proceed as follows

1. We assume that L is REGULAR

2. Hence by **Pumping Lemma 1** we get that **there is** a word $w \in L$ that can be **re-written** as $w = xyz$ for $y \neq e$ and $xy^n z \in L$ for all $n \geq 0$

3. We examine the fact $xy^n z \in L$ for all $n \geq 0$

4. If we get a CONTRADICTION we have proved that L is NOT REGULAR

Pumping Lemma Applications

Assume that $L = \{a^m b^m : \quad m \geq 0\}$ is REGULAR

L is infinite, hence **Pumping Lemma 1** applies and we know that **there is** a word $w \in L$ and **there are** $x, y, z \in \Sigma^*$ such that w can be **re-written** as $w = xyz$ and $y \neq e$ and $xy^n z \in L$ for all $n \geq 0$

Let $w \in L$ be such word, i.e. $xyz = a^m b^m$ for some $m \geq 0$
We will show that $xy^n z \in L$ for all $n \geq 0$ is **impossible**
and this **contradiction** proves that L is NOT REGULAR

Pumping Lemma Applications

Consider $w = xyz \in L$, i.e. $xyz = a^m b^m$ for **some** $m \geq 0$

We have to consider the following cases

Case 1

y consists entirely of a's

Case 2

y consists entirely of b's

Case 3

y contains both some a's followed by some b's

We will show that in each case assumption that $xy^n z \in L$ for all n leads to CONTRADICTION

Pumping Lemma Applications

Consider $w = xyz \in L$, i.e. $xyz = a^m b^m$ for some $m \geq 0$

Case 1: y consists entirely of a's

So x **must** consists entirely of a's only and z **must** consists of some a's followed by some b's

Remember that only we must have that $y \neq e$

We have the following situation

$x = a^p$ for $p \geq 0$ as x can be empty

$y = a^q$ for $q > 0$ as y must be nonempty

$z = a^r b^s$ for $r \geq 0$, $s > 0$ as we must have some b's

Pumping Lemma Applications

The condition $xy^n z \in L$ for all $n \geq 0$ becomes as follows

$$a^p (a^q)^n a^r b^s = a^{p+nq+r} b^s \in L$$

for all p, q, n, r, s such that the following conditions hold

C1: $p \geq 0, \quad q > 0, \quad n \geq 0, \quad r \geq 0, \quad s > 0$

By definition of L

$a^{p+nq+r} b^s \in L$ ifand only if $p + nq + r = s$

Take case: $p = 0, \quad r = 0, \quad q > 0, \quad n = 0$

We get $s = 0$ CONTRADICTION with **C1:** $s > 0$

Pumping Lemma Applications

Consider $xyz = a^m b^m$ for some $m \geq 0$

Case 2: y consists of b's only

So x **must** consists of some a's followed by some b's and z **must** have only b's, possibly none

We have the following situation

$x = a^p b^r$ for $p > 0$ as y has at least one b and $r \geq 0$

$y = b^q$ for $q > 0$ as y must be nonempty

$z = b^s$ for $s \geq 0$

Pumping Lemma Applications

The condition $xy^n z \in L$ for all $n \geq 0$ becomes as follows

$$a^p b^r (b^q)^n b^s = a^p b^{r+nq+r} \in L$$

for all p, q, n, r, s such that the following conditions hold

C2: $p > 0,\ r \geq 0 \quad q > 0, \quad n \geq 0, \quad s \geq 0$

By definition of L

$$a^p b^{r+nq+r} \in L \quad \text{if and only if} \quad p = r + qn + s$$

Take case: $r = 0, \quad n = 0, \quad q > 0$

We get $p = 0$ CONTRADICTION with **C2:** $p > 0$

Pumping Lemma Applications

Consider $xyz = a^m b^m$ for some $m \geq 0$

Case 3: y contains both a's and a's

So $y = a^p b^r$ for $p > 0$ and $r > 0$

Case $y = b^r a^p$ is impossible

Take case: $y = ab$, $x = e$, $z = e$ and $n = 2$

By Pumping Lemma we get that $y^2 \in L$

But this is a CONTRADICTION with $y^2 = abab \notin L$

We covered all cases and it **ends the proof**

Pumping Lemma Applications

Use Pumping Lemma to **prove** the following

Fact 2

The language $L \subseteq \{a\}^*$ defined as follows

$$L = \{a^n : \quad n \in Prime\}$$

IS NOT regular

Obviously, L is infinite and we use the Lecture version

Proof

Assume that L is regular, hence as L is infinite, so there is a word $w \in L$ that can be **re-written** as $w = xyz$ for $y \neq e$ and $xy^n z \in L$ for all $n \geq 0$

Consider $w = xyz \in L$, i.e. $xyz = a^m$ for **some** $m > 0$ and $m \in Prime$