INTRODUCTION TO THE THEORY OF COMPUTATION LECTURE NOTES

Professor Anita Wasilewska
Stony Brook University (SUNY at Stony Brook)

Course Text Book

ELEMENTS OF THE THEORY OF COMPUTATION

Harry R. Lewis, and Christos H. Papadimitriou

Prentice Hall, S2nd Edition

Chapter 2
Finite Automata

LECTURE SLIDES

Chapter 2
Finite Automata

Chapter 2
Finite Automata

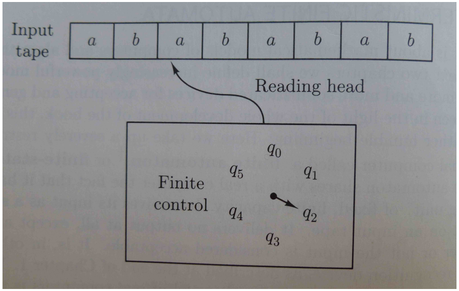
Slides Set 1

PART 1: Deterministic Finite Automata DFA

Deterministic Finite Automata DFA

Simple Computational Model

Here is a picture



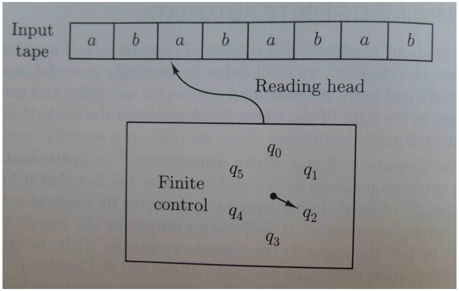
Here are the **components** of the model

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

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

δ is called the **transition function**

Transition Function

Given DFA

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

where

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

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**

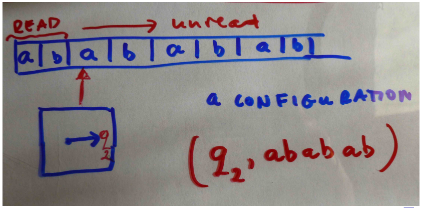
Definition

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

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

A binary relation

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

is called a **transition relation** when for any

$q,\ q' \in K,\ \ w_1, w_2 \in \Sigma^*$ the following holds

$$(q, w_1) \vdash_M (q', w_2)$$

if and only if

1. $w_1 = \sigma w_2$, for some $\sigma \in \Sigma$ (M **looks** at σ)

2. $\delta(q, \sigma) = q'$ (M **moves** from q to q' reading σ in w_1)

Transition Relation

Definition (Transition Relation short definition)

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**

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

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

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

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

Observe that when $n = 1$ the computation (q_1, w_1) **always** exists and is called a computation of the length one

It is also called a rivial 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

Definition

A word $w \in \Sigma^*$ is **accepted** by $M = (K,\ \Sigma,\ \delta,\ s,\ F)$ if and only if **there is** a computation

$$(q_1, w_1),\ (q_2, w_2),\ \ldots, (q_n, w_n)$$

such that $q_1 = s,\ w_1 = w,\ \ w_n = e$ and $q_n = q \in F$

Words Accepted by M

We re-write it as

Definition

A word $w \in \Sigma^*$ is **accepted** by $M = (K,\ \Sigma,\ \delta,\ s,\ F)$ if and only if **there is** a computation

$$(s, w),\ (q_2, w_2),\ \ldots, (q, e) \quad \text{and} \quad q \in F$$

When the computation is such that $q \notin F$ we say that the word w is **not accepted** (rejected) by M

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

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

Examples

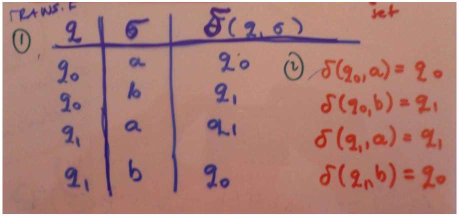
Example 1

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

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

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

is defined as follows



TRANS. F

set

(1)

q	σ	$\delta(q, \sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

(2)

$\delta(q_0, a) = q_0$

$\delta(q_0, b) = q_1$

$\delta(q_1, a) = q_1$

$\delta(q_1, b) = q_0$

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 $ababb \notin L(M)$

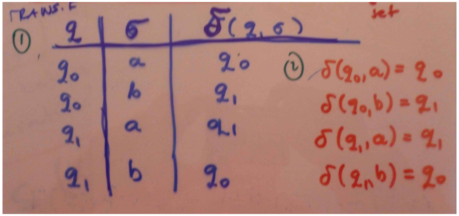
Observe that we always get **unique** computations, as δ is a function, hence he name Deterministic Finite Automaton (DFA)

Examples

Example 2

Let $M_1 = (K, \Sigma, \delta, s, F)$ for all components defined as in M from **Example 1**, except that we take now $F = \{q_0, q_1\}$

We remind that



TRANS. F

①

q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

set

② $\delta(q_0,a) = q_0$

$\delta(q_0,b) = q_1$

$\delta(q_1,a) = q_1$

$\delta(q_1,b) = q_0$

Exercise Show that now $ababb \in L(M_1)$

Language Accepted by M Revisited

We have defined the language **accepted** by M as

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

Question: how to write this definition in a more concise and elegant way

Answer: use the notion (Chapter 1, Lecture 3) of reflexive, transitive closure of $\vdash_M$ denoted by

$$\vdash_M{}^*$$

and now we write the definition of $L(M)$ as follows

Language Accepted by M
Revisited

Definition

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

We write it also using the **existential quantifier** symbol as

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

Language Accepted by M
Revisited

In order to justify the following I **definition**

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

We bring back the general notion of a **path** in a binary relation R and its reflexive, transitive closure R^* (Chapter 1)

It follows **directly** from these definitions that

$$(q_1, w_1) \vdash_M{}^* (q_n, w_n)$$

represents a **path**

$$(q_1, w_1), \ (q_2, w_2) \ \ \ldots, \ \ (q_{n-1}, w_{n-1}, \ (q_n, w_n)$$

in the relation $\vdash_M$, which is defined as a **computation**

$$(q_1, w_1) \vdash_M (q_2, w_2) \ \ \ldots, \ \ (q_{n-1}, w_{n-1} \vdash_M (q_n, w_n)$$

in M from (q_1, w_1) to (q_n, w_n)

Language Accepted by M
Revisited

Hence

$$(s, w) \vdash_M{}^* (q, e)$$

represent a computation

$$(s, w) \vdash_M (q_1, w_1), \quad \ldots, \quad (q_n, w_n) \vdash_M (q, e)$$

from (s, w) to (q, e),

So define the language L(M) as

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

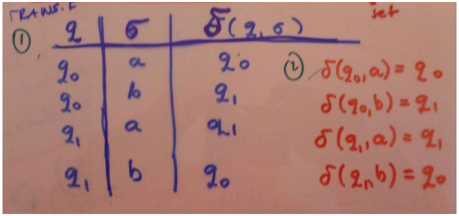
Example

Example

Let $M = (K, \Sigma, \delta, s, F)$ be automaton from our **Example 1**, i.e. we have

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

and the **transition function** $\delta : K \times \Sigma \longrightarrow K$ is defined as follows



TRANS.F

(1)

q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

set

(2)

$\delta(q_0,a) = q_0$

$\delta(q_0,b) = q_1$

$\delta(q_1,a) = q_1$

$\delta(q_1,b) = q_0$

Question Show that $aabba \in L(M)$

Example

We evaluate

$$(q_0, aabba) \vdash_M (q_0, abba) \vdash_M (q_0, bba) \vdash_M$$

$$(q_1, ba) \vdash_M (q_0, a) \vdash_M (q_0, e) \quad \text{and} \quad q_0 = s, \quad q_0 \in F = \{q_0\}$$

This proves that

$$(s, aabba) \vdash_M{}^{*} (q_0, e) \quad \text{for} \quad q_0 \in F$$

By definition

$$aabba \in L(M)$$

General remark

To **define** or to give an example of

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

means that one has to **specify all** its components K, Σ, δ, s, F

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

Exercise

Given $\Sigma = \{a, b\}$ and $K == \{q_0, q_1\}$

1. Define 3 automata M

2. Define an automaton M, such that $L(M) = \emptyset$

3. How many automata M can one define?

Exercise

1. Here are 3 automata $M_1 - M_3$

M₁ : $M_1 = (\ K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \ \delta,\ s = q_0,\ F = \{q_0\})$

$\delta(q_0, a) = q_0,\ \ \delta(q_0, b) = q_0,\ \ \delta(q_1, a) = q_0,\ \ \delta(q_1, b) = q_0$

M₂ : $M_2 = (\ K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \ \delta,\ s = q_0,\ F = \{q_1\})$

$\delta(q_0, a) = q_0,\ \ \delta(q_0, b) = q_0,\ \ \delta(q_1, a) = q_0,\ \ \delta(q_1, b) = q_1$

M₃ : $M_3 = (\ K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \ \delta,\ s = q_0,\ F = \{q_1\})$

$\delta(q_0, a) = q_0,\ \ \delta(q_0, b) = q_1,\ \ \delta(q_1, a) = q_1,\ \ \delta(q_1, b) = q_0$

Exercise

2. Define an automaton M, such that $L(M) = \emptyset$

Answer: The automata M_2 is such that $L(M_2) = \emptyset$ as there is no computation that would **start at initial state** q_0 and
end in the final state q_1 as in M_2

We have that

$$\delta(q_0, a) = q_0, \ \delta(q_0, b) = q_0$$

so we will never reach the **final state** q_1

Exercise

Here is another example:

Let M_4 be defined as follows

$$M_4 = (\ K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \ \delta,\ s = q_0,\ F = \emptyset)$$

$$\delta(q_0, a) = q_0,\ \ \delta(q_0, b) = q_0,\ \ \delta(q_1, a) = q_0,\ \ \delta(q_1, b) = q_0$$

$L(M_4) = \emptyset$ as there **is no** computation that would **start** at initial state q_0 and **end** in the final state as there is no final state

Exercise

3. How many automata M can one define?

Observe that all of M must have $\Sigma = \{a, b\}$ and $K == \{q_0, q_1\}$ so they **differ** on the choices of $\delta : K \times \Sigma \longrightarrow K$

By **Counting Functions Theorem** we have 2^4 possible choices for δ

They also can **differ** on the choices of **final states** F

There as many choices for final states as subsets of $K == \{q_0, q_1\}$, i.e. $2^2 = 4$

Additionally we have to count all combinations of choices of δ with choices of F

Challenge

1. Define an automata M with $\Sigma \neq \emptyset$ such that $L(M) = \emptyset$
2. Define an automata M with $\Sigma = \emptyset$ such that $L(M) \neq \emptyset$
3. Define an automata M with $\Sigma \neq \emptyset$ such that $L(M) \neq \emptyset$
4. Define an automata M with $\Sigma \neq \emptyset$ such that $L(M) = \Sigma^*$
5. Prove that there always exist an automata M such that $L(M) = \Sigma^*$

DFA State Diagram

As we could see the transition functions can be defined in many ways but it is difficult to decipher the workings of the automata they define from their mathematical definition

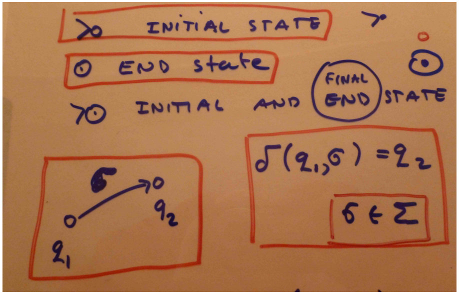
We usually use a much more clear graphical representation of the transition functions that is called a **state diagram**

Definition

The **state diagram** is a directed graph, with certain additional information as shown at the picture on next slide

DFA State Diagram

PICTURE 1



States are represented by the nodes

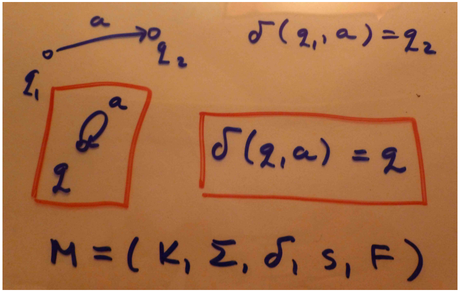
Initial state is shown by a $>\bigcirc$

Final states are indicated by a dot in a circle $\odot$

Initial state that is also a **final state** is pictured as $>\odot$

DFA State Diagram

PICTURE 2



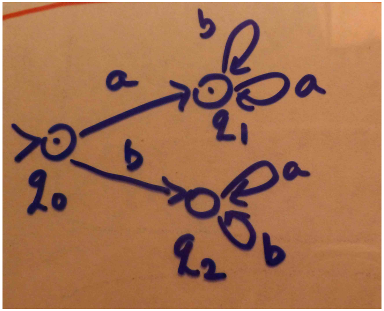
States are represented by the nodes

There is an **arrow labelled** a from node q_1 to q_2 whenever $\delta(q_1, a) = q_2$

A Simple Problem

Problem

Given $M = (K, \Sigma, \delta, s, F)$ described by the following **diagram**

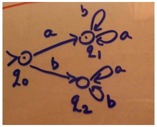


1. List all components of M

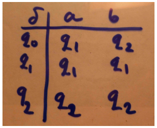
2. Describe $L(M)$ as a **regular expression**

A Simple Problem

Given the **diagram**



Components are: $M = (K,\ \Sigma,\ \delta,\ s,\ F)$ for
$\Sigma = \{a, b\}$, $K = \{q_0, q_1, q_2\}$,
$s = q_0$, $F = \{q_0, q_1\}$ and the **transition function** is given by following table



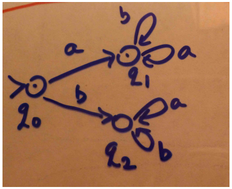
δ	a	b
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2

A Simple Problem

2. Describe $L(M)$ as a **regular expression**, where

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

Let's look again at the **diagram** of M



Observe that the state q_2 **does not influence** the language L(M). We call such state a **trap state** and say:
The state q_2 is a **trap state**
We read from the **diagram** that

$$L(M) = a(a \cup b)^* \cup e \quad \text{as a regular expression}$$

$$L(M) = \{a\} \circ \{a, b\}^* \cup \{e\} \quad \text{as a set}$$

DFA Theorem

DFA Theorem

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

$$e \in L(M) \quad \text{if and only if} \quad s \in F$$

where we **defined** $L(M)$ as follows

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

Proof

Let $e \in L(M)$, then by definition $(s, e) \vdash_M{}^*(q, e)$ and $q \in F$

This is possible only when the computation is of the length one (case $n = 1$), i.e when it is (s, e) and $s = q$, hence $s \in F$

Suppose now that $s \in F$

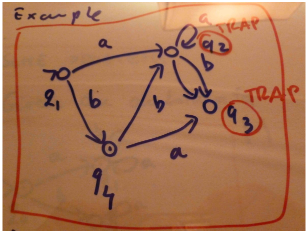
We know that $\vdash_M{}^*$ is reflexive, so $(s, e) \vdash_M{}^*(s, e)$ and as $s \in F$, we get $e \in L(M)$

Definition of TRAP States of M

Definition

A **trap state** of a DFA automaton M is any of its states that **does not influence** the language L(M) of M

Example

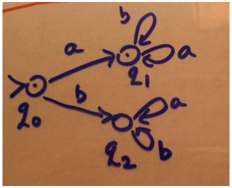


$L(M) = b$ written in shorthand notation, $L(M) = \{b\}$, or $L(M) = \mathcal{L}(b) = \{b\}$

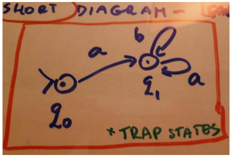
States q_2, q_3 are **trap states**

TRAP States of M

Given a **diagram** of M



The state q_2 is the **trap state** and we can write a **short diagram** of M as follows



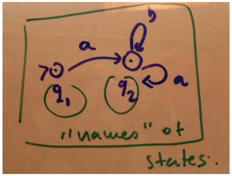
Remember that if you use the **short diagram** you must add statement: " plus **trap states**"

Short and Pattern Diagrams of M

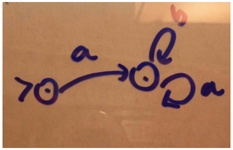
Definition

A diagram of M with some or all of its **trap states** removed is called a **short diagram**

"Our" M becomes



We can "shorten" the diagram even more by removing the **names** of the states

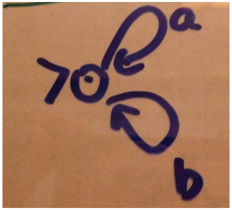


Such diagram, with names of the states removed is called a **pattern diagram**

Pattern Diagrams

Pattern Diagrams are very useful when we want to ”read” the language M directly out of the diagram

Lets look at M_1 given by a diagram



It is obvious that (we write a shorthand notion!)

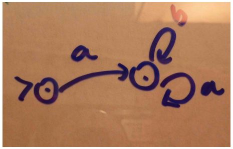
$$L(M_1) = (a \cup b)^* = \Sigma^*$$

Remark that the **regular expression** that defines the language $L(M_1)$ is $\alpha = (a \cup b)^*$

We add the description $L(M_1) = \Sigma^*$ as yet another useful informal **shorthand notation** notation

Pattern Diagrams

The **pattern diagram** for "our" M is



It is obvious that (we write a shorthand notion!) - must add: plus **trap states**

$$L(M) = aL(M_1) \cup e$$

We must add e to the language by **DFA Theorem**, as we have that $s \in F$
Finally we obtain the following regular expression that defines the language and write it as

$$L(M) = a(a \cup b)^* \cup e$$

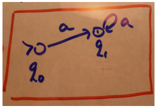
We can also write L(M) in an **informal way** (Σ^* is not a regular expression) as

$$L(M) = \Sigma^* \cup$$

Trap States

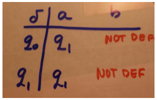
Why do we need trap states?

Let's take $\Sigma = \{a, b\}$ and let M be defined by a diagram



Obviously, the diagram means that M is such that its language is $L(M) = aa^*$

But by definition, $\delta : K \times \Sigma \longrightarrow K$ and we get from the diagram



δ	a	b
q_0	q_1	NOT DEF
q_1	q_1	NOT DEF

We must "complete" definition of δ by making it a function (still preserving the language)

To do so introduce a new state q_2 and make it a **trap state** by defining $\delta(q_0, b) = q_2$, $\delta(q_1, b) = q_2$

Short Problems

For all **short problems** presented here and given on Quizzes and Tests, you have to do the following

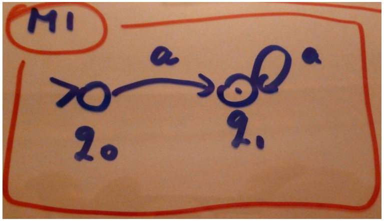
1. Decide and explain whether the given **diagram** represents a DFA or does not, i.e. is not an automatan

2. List all components of M when it represents a DFA

3. Describe $L(M)$ as a **regular expression** when it does represent a DFA

Short Problems

Consider a **diagram** M1



1. Yes, it represents a DFA; δ is a function on $\{q_0, q_1\} \times \{a\}$ and initial state $s = q_0$ exists

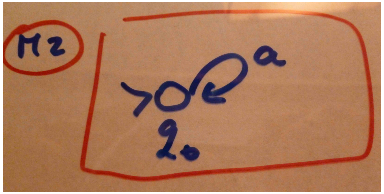
2. $K = \{q_0, q_1\}, \ \Sigma = \{a\}, \ s = q_0, \ F = \{q_1\},$

$\delta(q_0, a) = q_1, \ \ \delta(q_1, a) = q_1$

3. $L(M1) = aa^*$

Short Problems

Consider a **diagram** M2



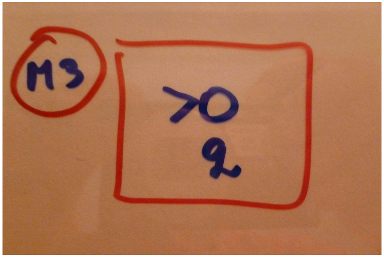
1. Yes, it represents a DFA; δ is a function on $\{q_0\} \times \{a\}$ and initial state $s = q_0$ exists

2. $K = \{q_0\},\ \Sigma = \{a\},\ s = q_0,\ F = \emptyset,\ \delta(q_0, a) = q_0$

3. $L(M2) = \emptyset$

Short Problems

Consider a **diagram** M3



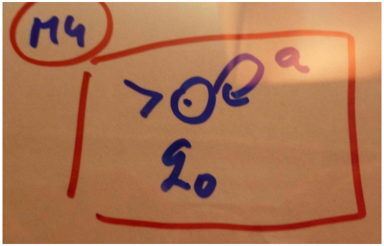
1. Yes, it represents a DFA; initial state $s = q_0$ exists

2. $K = \{q_0\}, \ \Sigma = \emptyset, \ s = q_0, \ F = \emptyset, \ \delta = \emptyset$

3. $L(M3) = \emptyset$

Short Problems

Consider a **diagram** M4



1. Yes, it represents a DFA; initial state $s = q_0$ exists

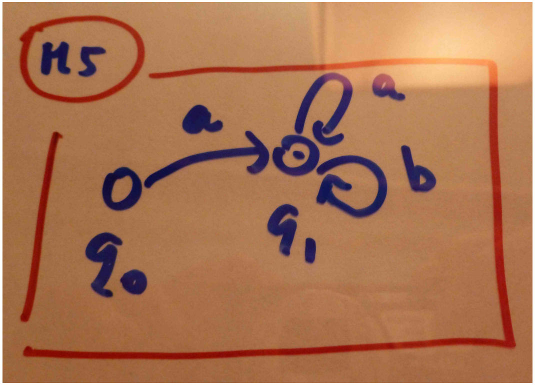
2. $K = \{q_0\},\ \Sigma = \{a\},\ s = q_0,\ F = \{q_0\},\ \delta(q_0, a) = q_0$

3. $L(M4) = a^*$

Remark $e \in L(M4)$ by **DFA Theorem**, as $s = q_0 \in F = \{q_0\}$

Short Problems

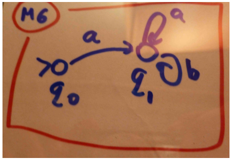
Consider a **diagram** M5



1. NO! it is NOT DFA - **initial state** does not exist

Short Problems

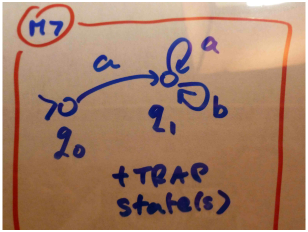
Consider a **diagram** M6



1. NO! Initial state does exist, but δ is not a function; $\delta(q_0, b)$ is **not defined** and we didn't say "plus **trap states**"

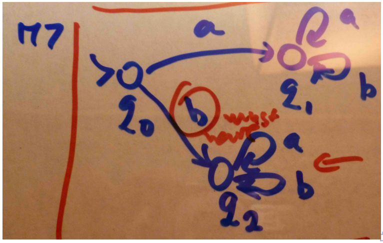
Short Problems

Consider a **diagram** M7



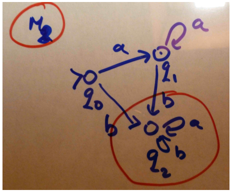
1. Yes! it is DFA

Initial state exists and we can complete definition of δ by adding a **trap state** as pictured below

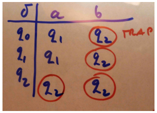


Short Problems

Consider a **diagram** M8



1. Yes! Initial state exists and it is a **short diagram** of a DFA
We make δ a function by adding a **trap state** q_2



δ	a	b	
q_0	q_1	q_2	TRAP
q_1	q_1	q_2	
q_2	q_2	q_2	

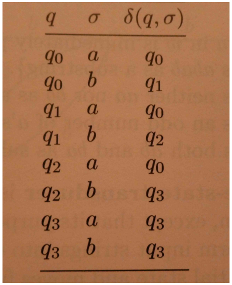
3. $L(M8) = aa^*$
We chose to add one **trap state** but it is possible to add as many as one wishes
Observe that $L(M8) = L(M1)$ and M1, M8 are defined for different alphabets

Two Problems

P1 Let $\Sigma = \{a_1, a_2, \ldots, a_{1025}, \ \ldots, a_{2^{105}} \}$
Draw a **state diagram** of M such that $L(M) = a_{1025(}a_{1025})^*$
P2

1. Draw a **state diagram** of **transition function** δ given by the table below

2. Give an **example** and automaton M with with this δ



q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_2
q_2	a	q_0
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3

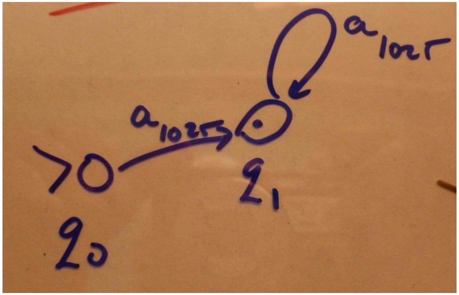
3. Describe the language of M

P1 Solution

P1 Let $\Sigma = \{a_1, a_2, \ldots, a_{1025}, \ \ldots, a_{2^{105}} \}$

Draw a **state diagram** of M such that $L(M) = a_{1025}(a_{1025})^*$

Solution

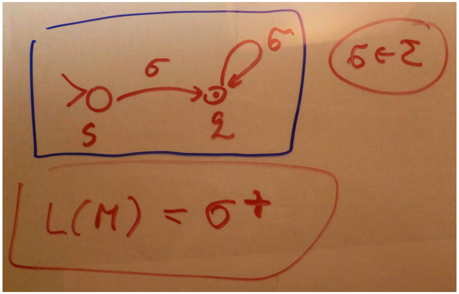


PLUS a LOT of **trap states**!

Σ has 2^{105} elements; we need a **trap state** for each of them except a_{1025}

P1 Solution

Observe that we have a following **pattern** for any $\sigma \in \Sigma$



$$L(M) = \sigma^{+} \quad \text{for any} \quad \sigma \in \Sigma$$

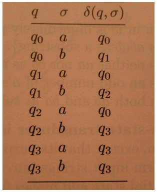
PLUS a LOT of **trap states**! except for the case when $\Sigma = \{\sigma\}$

P2 Solutions

P2

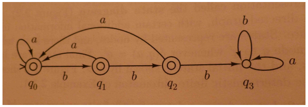
1. Draw a **state diagram** of **transition function** δ given by the table below

2. Give an **example** and automaton M with with this δ



q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_2
q_2	a	q_0
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3

Here is the **example** of M from our book, page 59

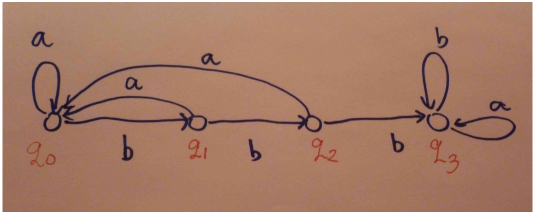


$L(M) = \{w \in \{a,b\}^* :$ w does not contain three consecutive $b's\}$

P2 Solution

Observe that the book example is only one of many possible examples of automata **we can define** based on δ with the following

State diagram:



Two more examples follow

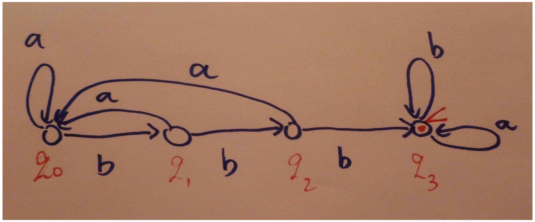
Please invent some more of your own!

Be careful! This diagram is NOT an automaton!!

P2 Examples

Example 1

Here is a full **diagram** of M1



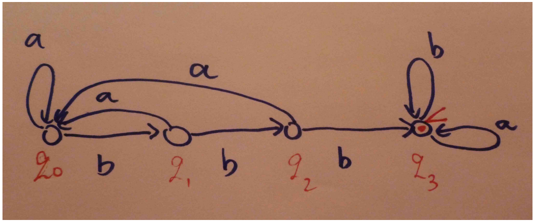
$$L(M) = (a \cup b)^* = \Sigma^*$$

Observe that $e \in L(M1)$ by the DFA **Theorem** and the states q_0, q_1, q_2 are **trap states**

P2 Examples

Example 2

Here is a full **diagram** of M1 from **Example 1**



$$L(M) = (a \cup b)^* = \Sigma^*$$

Observe that we can make **all, or any** of the states q_0, q_1, q_2 as **final states** and they will still will remain the **trap states**

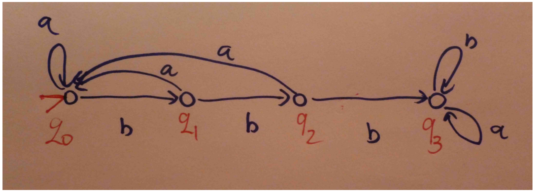
Definition

A **trap state** of a DFA automaton M is any of its states that **does not influence** the language L(M) of M

P2 Examples

Example 3

Here is a full **diagram** of M2 with the same transition function as M1



$$L(M) = \emptyset$$

Observe that $F = \emptyset$ and hence here is no computation that would finish in a **final state**

More Problems

P3 Construct a DFA M such that

$L(M) = \{w \in \{a, b\}^* : \ w$ has abab as a substring $\}$

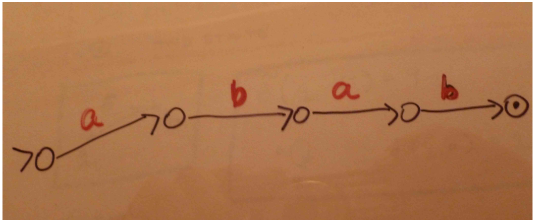
Problems Solutions

P3 Construct a DFA M such that

$$L(M) = \{w \in \{a,b\}^* : \ w \text{ has } abab \text{ as a substring } \}$$

Solution The essential part of the **diagram** must produce abab and it can be surrounded by proper elements on both sides and can be repeated

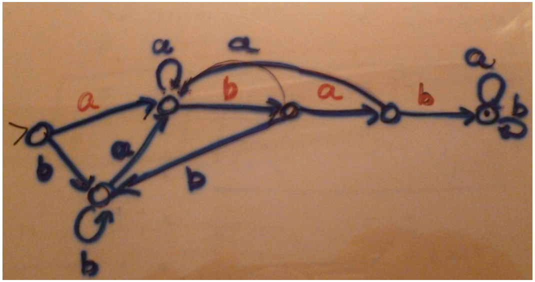
Here is the essential part of the **diagram**



Problems Solutions

We complete the essential part following the fact that it can be surrounded by proper elements on both sides and can be repeated

Here is the **diagram** of M



Observe that this is a **pattern diagram**; you need to add names of states only if you want to list all components

M does not have trap states

More Problems

P4 Construct a DFA M such that

$L(M) = \{w \in \{a, b\}^* :$ every substring of length 4 in word w

contains at least one b $\}$

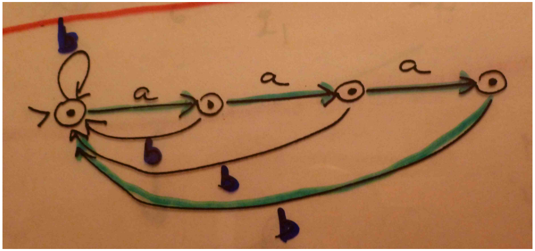
More Problems

P4 Construct a DFA M such that

$L(M) = \{w \in \{a,b\}^* :$ every substring of length 4 in word w

contains at least one b $\}$

Solution Here is a **short pattern diagram** (the trap states are not included)



More Problems

P5 Construct a DFA M such that

$L(M) = \{w \in \{a, b\}^* :$ every word w contains

an even number of sub-strings ba $\}$

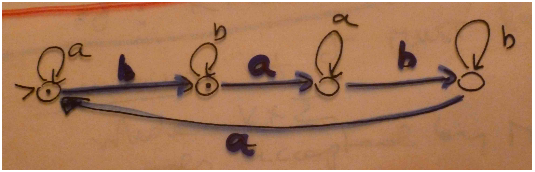
More Problems

P5 Construct a DFA M such that

$L(M) = \{w \in \{a,b\}^* :$ every word w contains

an even number of sub-strings ba $\}$

Solution Here is a **pattern diagram**



Zero is an even number so we must have that $e \in L(M)$, i.e. we have to make the initial state also a final state

More Problems

P6 Construct a DFA M such that

$L(M) = \{w \in \{a, b\}^* :$ each a in w is

immediately preceded and immediately followed by b $\}$

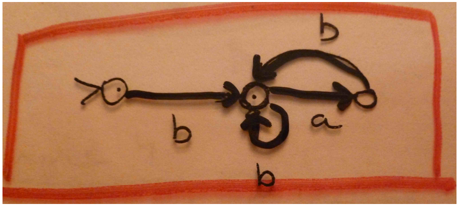
More Problems

P6 Construct a DFA M such that

$L(M) = \{w \in \{a, b\}^* :$ each a in w is

immediately preceded and immediately followed by b $\}$

Solution: Here is a **short pattern diagram** - and we need to say: plus trap states)

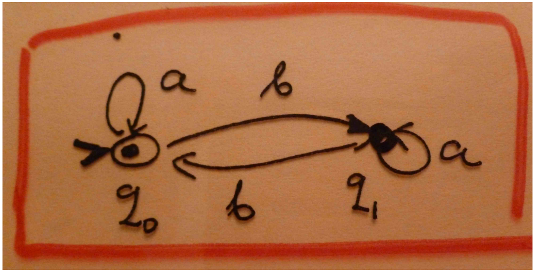


It is a **short diagram** because we omitted needed **trap states** (can be more then one, but one is sufficient)

Complete the diagram as an exercise

More Problems

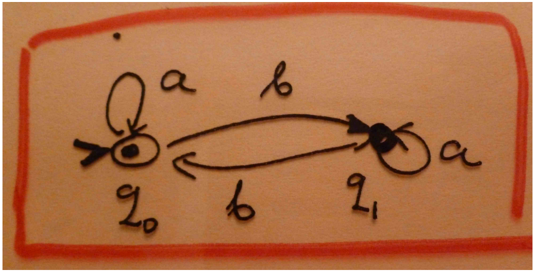
P7 Here is a DFA M defined by the following diagram



Describe $L(M)$ as a **regular expression**

More Problems

P7 Here is a DFA M defined by the following diagram



Describe $L(M)$ as a **regular expression**

Solution

$$L(M) = a^* \cup (a^*ba^*ba^*)^*$$

Observe that $e \in L(M)$ by the DFA **Theorem**

Short Problems

SP1 Given an automaton M1

$$M1 = (\ K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \ \delta,\ s = q_0,\ F = \emptyset)$$

$$\delta(q_0, a) = q_0,\ \ \delta(q_0, b) = q_0,\ \ \delta(q_1, a) = q_0,\ \ \delta(q_1, b) = q_0$$

1. Draw its **state diagram**

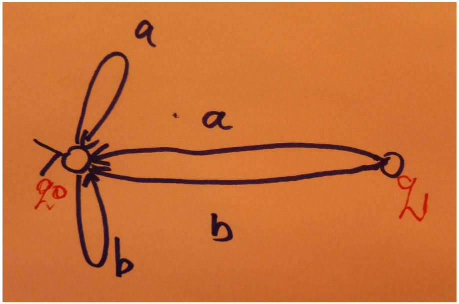
2. List **trap states**, if any

3. Describe L(M1)

SP1 Solution

SP1

1. Here is the **state diagram**



2. q_1 is a **trap state** - M1 never gets there

3. $L(M1) = \emptyset$

Short Problems

SP2 Given an automaton M2

$$M2 = (\ K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \delta,\ s = q_0,\ F = \{q_1\})$$

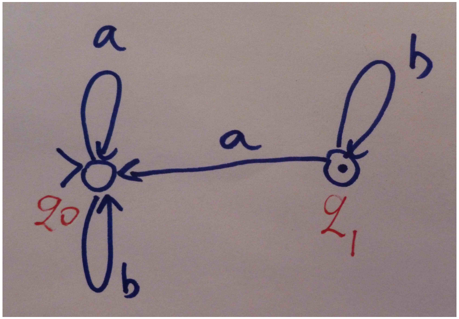
$$\delta(q_0, a) = q_0,\ \ \delta(q_0, b) = q_0,\ \ \delta(q_1, a) = q_0,\ \ \delta(q_1, b) = q_1$$

1. Draw its **state diagram**
2. List **trap states**, if any
3. Describe L(M2)

SP2 Solution

SP2

1. Here is the **state diagram**



2. q_1 is a **trap state** - it does not influence the language of M1

3. $L(M2) = \emptyset$

Short Problems

SP3 Given an automaton M3

$M3 = (K = \{q_0,\ q_1\},\ \Sigma = \{a,\ b\},\ \delta,\ s = q_0,\ F = \{q_1\})$

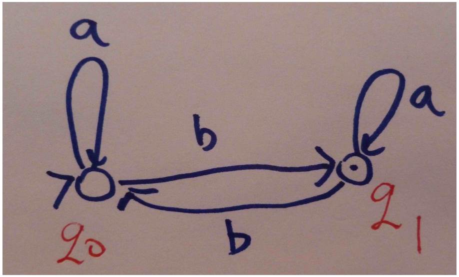
$\delta(q_0, a) = q_0,\ \delta(q_0, b) = q_1,\ \delta(q_1, a) = q_1,\ \delta(q_1, b) = q_0$

1. Draw its **state diagram**
2. List **trap states**, if any
3. Describe L(M3)

SP3 Solution

SP3

1. Here is the **state diagram**



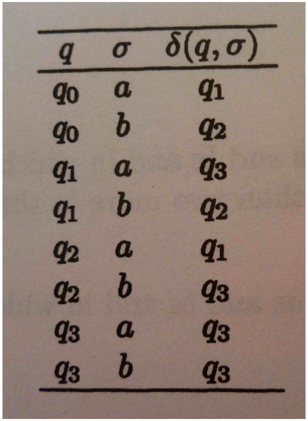
2. There are no **trap states**

3. $L(M3) = a^*b \cup a^*ba^* \cup (a^*ba^*ba^*b)^*$

$L(M3) = a^*ba^* \cup (a^*ba^*ba^*b)^*$

Short Problems

SP4 Given an automaton $M4 = (K, \Sigma, \delta, s, F)$ for $K = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $s = q_0$, $F = \{q_0, q_1, q_2\}$ and δ defined by the table below



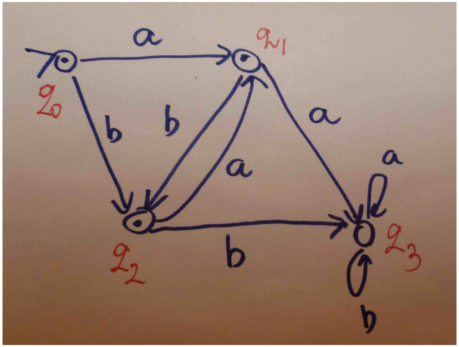
q	σ	$\delta(q,\sigma)$
q_0	a	q_1
q_0	b	q_2
q_1	a	q_3
q_1	b	q_2
q_2	a	q_1
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3

1. Draw its **state diagram**
2. Give a **property** describing L(M4)

SP4 Solution

SP4

1. Here is the **state diagram**

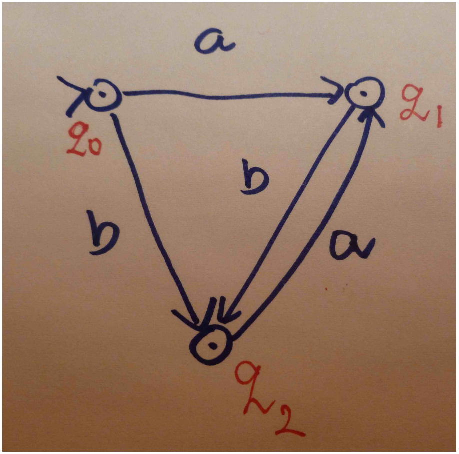


Observe that state q_3 is a **trap state** and the **short diagram** is as follows

SP4 Solution

SP4

1. Here is the **short diagram**

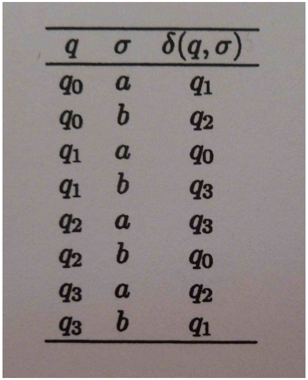


2. The language of M4 is

$L(M4) = \{w \in \Sigma^*$: neither aa nor bb is a substring of $w\}$

Short Problems

SP5 Given an automaton $M5 = (K, \Sigma, \delta, s, F)$ for $K = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $s = q_0$, $F = \{q_1\}$ and δ defined by the table below



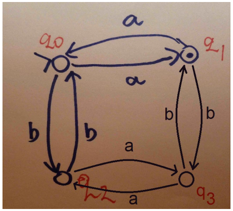
q	σ	$\delta(q, \sigma)$
q_0	a	q_1
q_0	b	q_2
q_1	a	q_0
q_1	b	q_3
q_2	a	q_3
q_2	b	q_0
q_3	a	q_2
q_3	b	q_1

1. Draw its **state diagram**
2. Give a **property** describing L(M5)

SP5 Solution

SP5

1. Here is the **state diagram**



2. $L(M5) = \{w \in \Sigma^* :$ w has an odd number of a 's and an even number of of b 's $\}$

Chapter 2
Finite Automata

Slides Set 1

PART 2: Nondeterministic Finite Automata DFA
Equivalency of DFA and DFA

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

Class Definition

A Nondeterministic Finite Automata is a quintuple

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

where

K is a finite set of **states**

Σ is 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

Class 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 is 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**

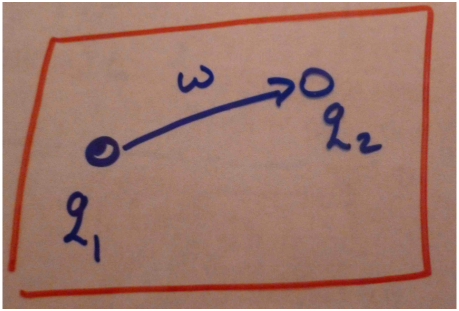
R4 The DFA **transition function** δ is a particular case of the NDFA **transition relation** Δ, hence similarity of notation

NDFA Diagrams

We extend the notion of the **state diagram** to the case of the NDFA in natural was as follows

$(q_1, w, q_2) \in \Delta$ means that M in a state q_1 reads the word $w \in \Sigma^*$ and goes to the state q_2

Picture

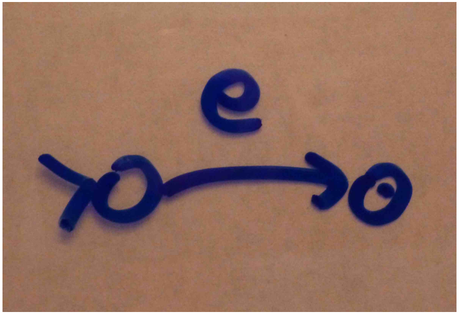


Remember that in particular w = e

Examples

Example 1

Let M be given by a diagram

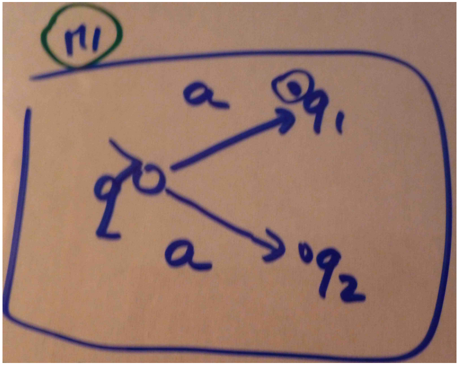


By definition M **is not** a deterministic DFA as it reads $e \in \Sigma^*$

$$L(M) = \{e\}$$

Examples

Example 2
Let M1 be given by a diagram



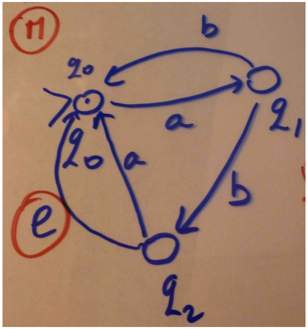
Observe that M1 **is not** a deterministic DFA as $(q, a, q_1) \in \Delta$ and $(q, a, q_2) \in \Delta$ what proves that Δ is **not a function**

$$L(M1) = \{a\}$$

Examples

Example 3

Let M be given by a diagram



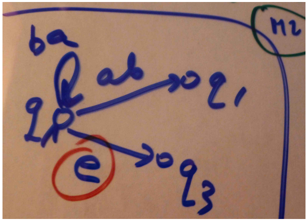
M **is not** a deterministic DFA as $(q_2, e, q_0) \in \Delta$ and this is not admitted in DFA

$\Delta = \{(q_0, a, q_1),\ (q_1, b, q_0),\ (q_1, b, q_2),\ (q_2, a, q_0), (q_2, e, q_0)\}$

Examples

Example 4

Let M be given by a diagram



M **is not** a deterministic DFA as $(q, ab, q_1) \in \Delta$ and this is not admitted in DFA

$\Delta = \{(q, ba, q), (q, ab, q_1), \ (q, e, q_3)\}$ and $F = \emptyset$

$$L(M1) = \emptyset$$

NDFA - Book Definition

Book 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 , the **transition relation** is defined as

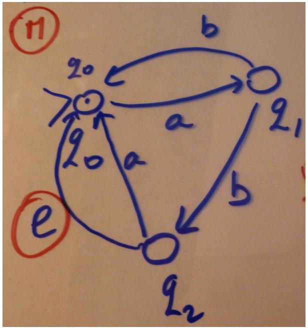
$$\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$$

Observe that Δ is finite set as both K and $\Sigma \cup \{e\}$ are finite sets

Book Definition Example

Example

Let M be automaton from **Example 3** given by a diagram



M follows the **Book Definition** as

$$\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$$

Equivalence of Definitions

The Class and the Book definitions are equivalent

1. We get the **Book Definition** as a particular case of the **Class Definition** as

$$\Sigma \cup \{e\} \subseteq \Sigma^*$$

2. We will show later a general method how to transform any automaton defined by the **Class Definition** into an equivalent automaton defined by the **Book Definition**

When solving problems you can use any of these definitions

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

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

For M defined by the **Book Definition** definition of the Transition Relation is the same but for the fact that

$$u \in \Sigma \cup \{e\}$$

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 $\vdash_M$

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 Statement

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

Equivalency of Automata Theorems

The **Equivalency Statement** 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;

We will describe, given a **NDFA** M, a general method of

construction step by step of an **DFA** M' that accepts

the came language as M

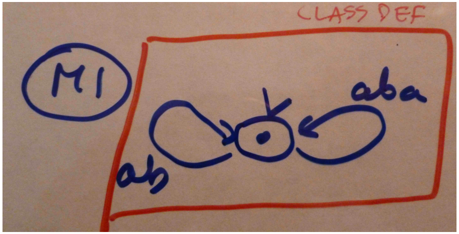
Before we define the **poof** construction we discuss some examples and some general automata properties

EXAMPLES and QUESTIONS

Examples

Example 1

Here is a **diagram** of NDFA M1 - Class Definition

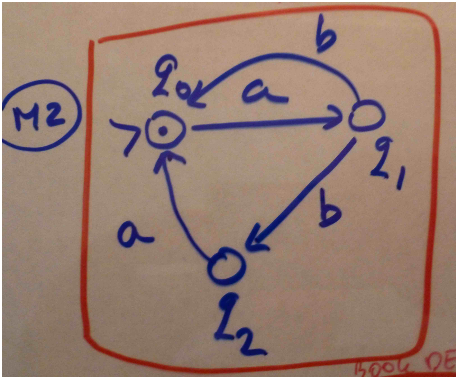


$$L(M1) = (ab \cup aba)^*$$

Examples

Example 2

Here is a **diagram** of NDFA M2 - Book Definition



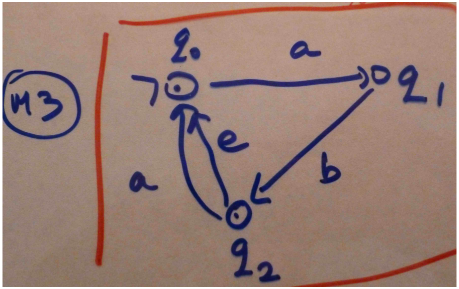
Observe that M2 **is not deterministic** (even if we add "plus trap states) because Δ is **not a function** as $(q_1, b, q_0) \in \Delta$ and $(q_1, b, q_2) \in \Delta$

$$L(M2) = (ab \cup aba)^*$$

Examples

Example 3

Here is a **diagram** of NDFA M3 - Book Definition



Observe that M2 **is not deterministic** $(q_1, e, q_0) \in \Delta$

$$L(M3) = (ab \cup aba)^*$$

Question 1

All automata in **Examples 1-3** accept the same language, hence by definition, they are **equivalent nondeterministic** automata, i.e.

$$M1 \approx M2 \approx M3$$

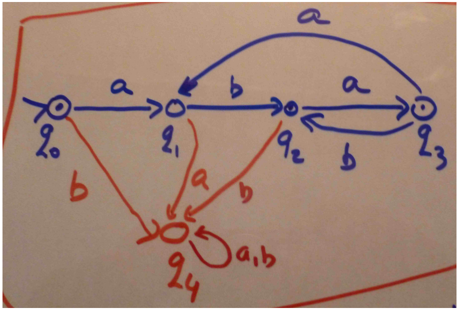
Question 1

Construct a **deterministic** automaton M4 such that

$$M1 \approx M2 \approx M3 \approx M4$$

Question1 Solution

Here is a **diagram** of **deterministic** DFA M4



Observe that q_4 is a **trap state**

$$L(M4) = (ab \cup aba)^*$$

Question 2

Given an alphabet

$$\Sigma = \{a_1,\ a_2, \ldots, a_n\} \quad \text{for} \quad n \geq 2$$

Question 2

Construct a **nondeterministic** automaton M such that

$$L = \{w \in \Sigma^* : \text{ at least one letter from } \Sigma \text{ is missing in } w\}$$

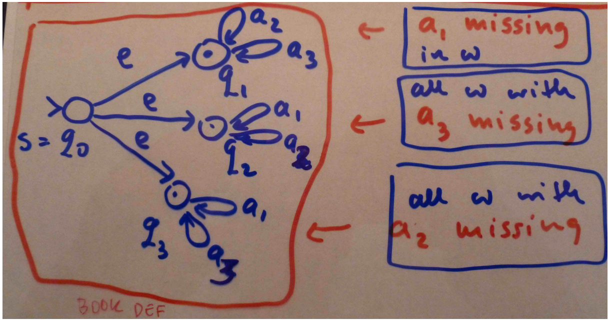
Take $n = 4$, i.e. $\Sigma = \{a_1,\ a_2,\ a_3,\ a_4\}$

Some words in L are:

$$e \in L,\ \ a_1 \in L,\ a_1a_2a_3 \in L,\ a_1a_2a_2a_3a_3 \in L\ a_1a_4a_1a_2 \in L, \ldots$$

Question 2 Solution

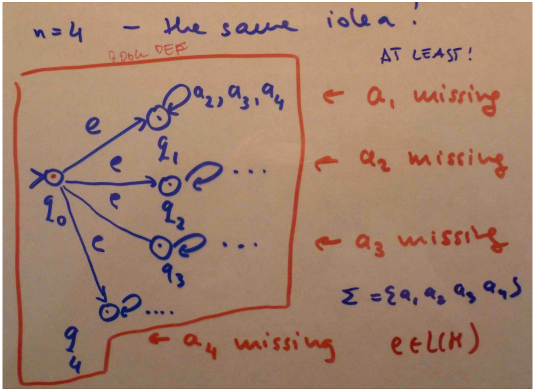
Here is **solution** for $n = 3$, i.e. $\Sigma = \{a_1, a_2, a_3\}$



Write a solution for $n = 4$

Question 2 Solution

Here is the **solution** for $n = 4$, i.e. $\Sigma = \{a_1, \ a_2, \ a_3, \ a_4\}$



Write a **general** form of solution for $n \geq 2$

Question 2 Solution

General case

$M = (K,\ \Sigma,\ \Delta,\ s,\ F)$ for $\Sigma = \{a_1,\ a_2, \dots, a_n\}$ and $n \geq 2$,

$K = \{s = q_0,\ q_1, \dots, q_n\},\ F = K - \{q_0\}$, or $F = K$ and

$$\Delta = \bigcup_{i=1}^{n} \{(q_0, e, q_i)\} \cup \bigcup_{i,j=1}^{n} \{(q_i, a_j, q_i) :\ i \neq j\}$$

$i \neq j$ means that a_i is missing in the loop at state q_i

PROPERTIES
Equivalence of Two Definitions

Equivalence of Two Definitions

Book Definition (BD)

$$\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$$

Class Definition (CD)

Δ is a **finite set** and

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

Fact 1

Any **(BD)** automaton M is a **(CD)** automaton M

Proof

The **(BD)** of Δ is a particular case of the **(CD)** as

$$\Sigma \cup \{e\} \subseteq \Sigma^*$$

Equivalence of Two Definitions

Fact 2

Any **(CD)** automaton M can be transformed into an equivalent **(BD)** automaton M '

Proof

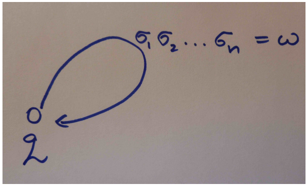
We use a " streching " technique

For any $w \neq e$, $w \in \Sigma^*$ and **(CD)** transition $(q, w, q') \in \Delta$, we transform it into a **sequence** of **(BD)** transactions each reading only $\sigma \in \Sigma$ that will at the end read the whole word $w \in \Sigma^*$

We leave the transactions $(q, e, q') \in \Delta$ unchanged

Stretching Process

Consider $w = \sigma_1, \sigma_2, \ldots \sigma_n$ and a transaction $(q, w, q) \in \Delta$ as depicted on the diagram



We construct Δ' in M ' by **replacing** the transaction $(q, \sigma_1, \sigma_2, \ldots \sigma_n, q)$ by

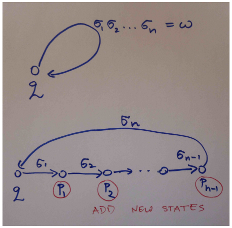
$$(q, \sigma_1, p_1), \ (p_1, \sigma_2, p_2), \ \ldots \ (p_{n-1}, \sigma_n, q)$$

and **adding** new states $p_1, p_2, \ldots p_{n-1}$ to the set K of M making at **this stage**

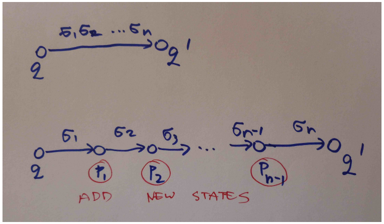
$$K' = K \cup \{p_1, p_2, \ldots p_{n-1}\}$$

Stretching Process

This transformation is depicted on the diagram below



We proceed in a similar way in a case of $w = \sigma_1, \sigma_2, \dots \sigma_n$ and a transaction $(q, w, q') \in \Delta$



Equivalent M’

We proceed to do the ”stretching” for all $(q, w, q') \in \Delta$ for $w \neq e$ and take as

$$K' = K \cup P$$

where $P = \{p : \ p$ added by stretching for all $(q, w, q') \in \Delta\}$

We take as

$$\Delta = \Delta^{\Sigma} \cup \{(q, \sigma_i, p) : \ p \in P, w = \sigma_1, \ \dots \sigma_n, \ \ (q, w, q') \in \Delta\}$$

where

$$\Delta^{\Sigma} = \{(q, \sigma, q') \in \Delta : \ \ \sigma \in (\Sigma \cup \{e\}), \ \ q, q' \in K\}$$

Proof of Equivalency of DFA and NDFA

Equivalency of DFA and NDFA

Let's now go back now to the **Equivalency Statement** that consists of the following 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

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

This is already **proved**

Equivalency Theorem 2

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

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

This is to be proved

Equivalency Theorem

Our goal now is to prove the following

Equivalency Theorem 2

For any **nondeterministic** automaton

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

there is, i.e. we give an algorithm for its construction a **deterministic** automaton

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

such that

$$M \approx M'$$

i.e.

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

General Remark

General Remark

We base the **proof** of the equivalency of DFA and NDFA automata on the **Book Definition** of NDFA

Let's now explore some **ideas** laying behind the main points of the **proof**

They are based on two **differences** between the DFA and NDF automata

We discuss now these **differences** and basic **ideas** how to overcome them, i.e. how to "make" a deterministic automaton out of a nonderetministic one

NDFA and DFA Differences

Difference 1

DFA **transition** function δ even if expressed as a **relation**

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

must be a function, while the NDFA transition relation Δ

$$\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$$

may **not be** a function

NDFA and DFA Differences

Difference 2

DFA **transition** function δ **domain** is the set

$$K \times \Sigma$$

while NDFA **transition** relation Δ **domain** is the set

$$K \times \Sigma \cup \{e\}$$

Observe that the NDFA **transition** relation Δ may contain a configuration (q, e, q') that allows a nondeterministic automaton to **read** the empty word e, what is **not allowed** in the deterministic case

In order to **transform** a nondeterministic M into an equivalent deterministic M' we have to **eliminate** the both Differences 1 and 2

Example

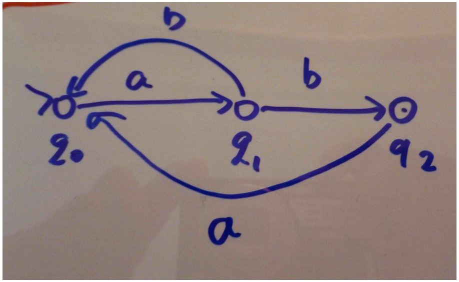
Let's look first at the following

Example

$$M = (\{q_0, q_1, q_2, q_3\},\ \Sigma = \{a, b\},\ \Delta,\ s = q_0,\ F = \{q_2\})$$

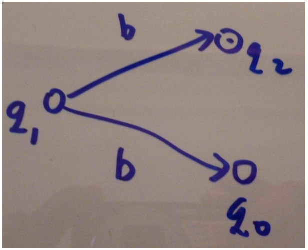
$\Delta = \{(q_0, a, q_1), (q_1, b, q_0), (q_1, b, q_2), (q_2, a, q_0)\}$

Diagram of M



Example

The **non-function** part of the diagram is



Question

How to transform it into a FUNCTION???

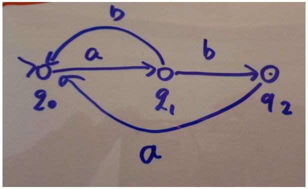
IDEA 1: make the states of M' as some SETS made out of states of M and put in this case

$$\delta(\{q_1\}, b) = \{q_0, q_2\}$$

IDEA ONE

IDEA 1: we make the states of M' as some SETS made out of states of M

We read other transformation from the **Diagram** of M



$\delta(\{q_0\}, a) = \{q_1\}, \quad \delta(\{q_2\}, a) = \{q_0\}$ and of course

$\delta(\{q_1\}, b) = \{q_0, q_2\}$

We make the state $\{q_0\}$ the **initial state** of M' as q_0 was the initial state of M and

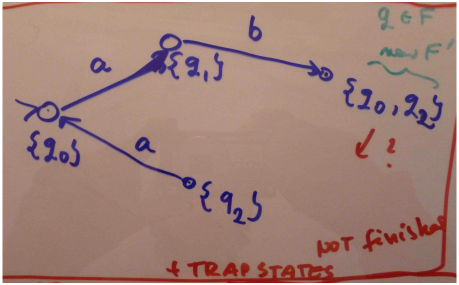
we make the states $\{q_0, q_2\}$ and $\{q_2\}$ **final states** of M' and as q_2 was a **final state** of M

Example

We have constructed a part of

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

The **Unfinished Diagram** is



There will be many **trap states**

IDEA ONE

IDEA ONE General Case

We take as the set K' of states of M' the **set of all subsets** of the set K of states of M

We take as the **initial state** of M' the set $s' = \{s\}$, where s is the initial state of M, i.e. we put

$$K' = 2^K, \quad s' = \{s\}, \quad \delta : 2^K \times \Sigma \longrightarrow 2^K$$

We take as the **set of final states** F' of M' the set

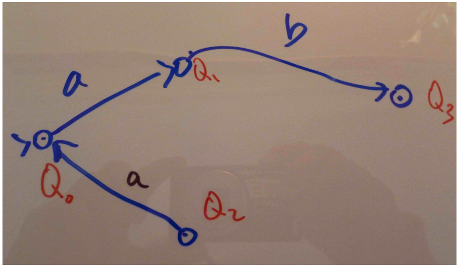
$$F' = \{Q \subseteq K : \quad Q \cap F \neq \emptyset\}$$

The general **definition** of the transition function δ will be given later

Example Revisited

In the case of our **Example** we had $K = \{q_0, q_1, q_2\}$
$K' = 2^K$ has 2^3 states

The portion of the **unfinished diagram** of M' is



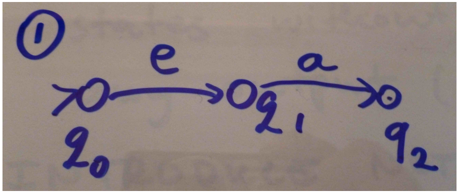
It is obvious that even the finished diagram will have A LOT of **trap states**

Difference 2 and Idea Two

Difference 2 and Idea Two - how to eliminate the e transitions

Example 1

Consider M1



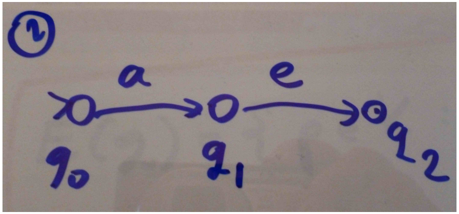
Observe that we can go from q_0 to q_1 reading only e, i.e. without reading any **input** symbol $\sigma \in \Sigma$

$$L(M1) = a$$

Examples

Example 2

Consider M2



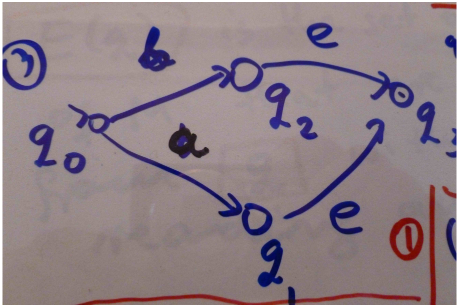
Observe that we can go from q_1 to q_2 reading only e, i.e. without reading any **input** symbol $\sigma \in \Sigma$

$$L(M2) = a$$

Examples

Example 3

Consider M3



Observe that we can go from q_2 to q_3 and from q_1 to q_3 without reading **any input**

$$L(M3) = a \cup b$$

Idea Two - Sets E(q)

The definition of the **transition function** δ of M' uses the following

Idea Two: a move of M' on reading an input symbol $\sigma \in \Sigma$ **imitates** a move of M on input symbol σ, possibly followed by **any** number of e-moves of M

To formalize this idea we need a special definition

Definition of E(q)

For any state $q \in K$, let E(q) be the set of all states in M they are **reachable** from state q without reading **any input**, i.e.

$$E(q) = \{p \in K : \ (q, e) \vdash_M{}^* (p, e)\}$$

Sets E(q)

Fact 1

For any state $q \in K$ we have that $q \in E(q)$

Proof

By definition

$$E(q) = \{p \in K : \ (q, e) \vdash_M{}^* (p, e)\}$$

and by the definition of reflexive, transitive closure $\vdash_M{}^*$ the **trivial path** (case n=1) always exists, hence

$$(q, e) \vdash_M{}^* (q, e)\}$$

what proves that $q \in E(q)$

Sets E(q)

Observe that by definitions of $\vdash_M{}^*$ and E(q) we have the following

Fact 2

1. E(q) is a **closure** of the set $\{q\}$ under the relation

$$\{(p, r) : \text{ there is a transition } (p, e, r) \in \Delta\}$$

2. E(q) can be computed by the following

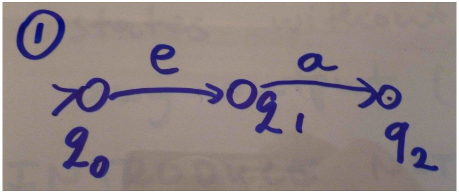
Algorithm

```
Initially set  $E(q) := \{q\}$
while there is  $(p, e, r) \in \Delta$ with $p \in E(q)$ and $r \notin E(q)$
do:  $E(q) := E(q) \cup \{r\}$
```

Example

We go back to the **Example 1**, i.e.

Consider M1



We evaluate

$$E(q_0) = \{q_0, q_1\}, \quad E(q_1) = \{q_1\}, \quad E(q_2) = \{q_2\}$$

Remember that always $q \in E(q)$

Definition of M’

Definition of M’

Given a **nondeterministic** automaton $M = (K,\ \Sigma,\ \Delta,\ s,\ F)$

we define the **deterministic** automaton M’ equivalent to M as

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

where

$$K' = 2^K, \quad s' = \{s\}$$

$$F' = \{Q \subseteq K : \quad Q \cap F \neq \emptyset\}$$

$\delta' : 2^K \times \Sigma \longrightarrow 2^K$ is such that

and for each $Q \subseteq K$ and for each $\sigma \in \Sigma$

$\delta'(Q,\sigma) = \bigcup\{E(p) : \ p \in K$ and $(q,\sigma,p) \in \Delta$ for some $q \in Q\}$

Definition of δ'

Definition of δ'

We re-write the definition of δ' in a a following form that is easier to use

$\delta' : 2^K \times \Sigma \longrightarrow 2^K$ is such that for each $Q \subseteq K$ and for each $\sigma \in \Sigma$

$$\delta'(Q, \sigma) = \bigcup_{p \in K} \{E(p) : \ (q, \sigma, p) \in \Delta \ \text{ for some } \ q \in Q\}$$

We write the above condition in a more clear form as

$$\delta'(Q, \sigma) = \bigcup_{p \in K} \{E(p) : \ \exists_{q \in Q} \ (q, \sigma, p) \in \Delta\}$$

Construction of of M’

Given a **nondeterministic** automaton $M = (K,\ \Sigma,\ \Delta,\ s,\ F)$

Here are the **STAGES** to follow when constructing M’

STAGE 1

1. For all $q \in K$, **evaluate** E(q)

$$E(q) = \{p \in K : \ (q, e) \vdash_M{}^* (p, e)\}$$

2. **Evaluate** initial and final states: $s' = E(s)$ and

$$F' = \{Q \subseteq K : \ Q \cap F \neq \emptyset\}$$

STAGE 2

Evaluate $\delta'(Q, \sigma)$ for $\sigma \in \Sigma,\ Q \in 2^K$

$$\delta'(Q, \sigma) = \bigcup_{p \in K} \{E(p) : \ \exists_{q \in Q}\ (q, \sigma, p) \in \Delta\}$$

Evaluation of δ'

Observe that domain of δ' is $2^K \times \Sigma$ and can be very large

We will **evaluate** δ' only on states that are relevant to the **operation** of M' and making all other states **trap states**
We do so to **assure** that

$$M' \approx M$$

i.e. to be able to **prove** that

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

Having this in mind we adopt the following definition

Evaluation of δ'

Definition

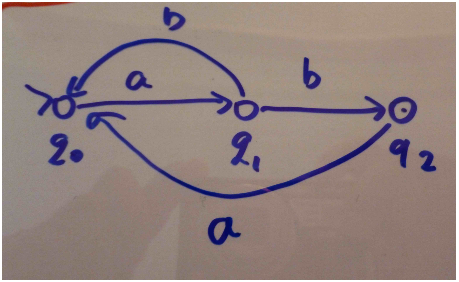
We say that a state $Q \in 2^K$ is **relevant** to the operation of M' and to the language L(M') if it can be **reached** from the **initial state** $s' = E(s)$ by reading some input string

Obviously, any state $Q \in 2^K$ that is **not reachable** from the **initial state** s' is **irrelevant** to the operation of M' and to the language L(M')

Construction of of M' Example

Example

Let M be defined by the following **diagram**



STAGE 1

1. For all $q \in K$, **evaluate** E(q)

M does not have e -transitions so we get

$E(q_0) = \{q_0\}, \quad E(q_1) = \{q_1\}, \quad E(q_2) = \{q_2\}$

2. **Evaluate** initial and some final states: $s' = E(q_0) = \{q_0\}$ and $\{q_2\} \in F'$

δ' Evaluation

STAGE 2

Here is a **General Procedure** for δ' evaluation

Evaluate $\delta'(Q, \sigma)$ only for **relevant** $Q \in 2^K$, i.e. follow the steps below

Step 1 Evaluate $\delta'(s', \sigma)$ for all $\sigma \in \Sigma$, i.e. all states **directly reachable** from s'

Step (n+1)

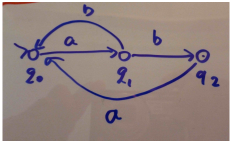
Evaluate δ' on all states that result from the **Step n**, i.e. on all states **already reachable** from s'

Remember

$$\delta'(Q, \sigma) = \bigcup_{p \in K} \{E(p) : \exists_{q \in Q} (q, \sigma, p) \in \Delta\}$$

Example STAGE 2

Diagram



STAGE 2

$$\delta'(Q,\sigma) = \bigcup_{p\in K}\{E(p) :\ \exists_{q\in Q}\ (q,\sigma,p) \in \Delta\}$$

Step 1 We evaluate $\delta'(\{q_0\}, a)$ and $\delta'(\{q_0\}, b)$

We look for the transitions from q_0

We have only one $(q_0, a, q_1) \in \Delta$ so we get $\delta'(\{q_0\}, a) = E(q_1) = \{q_1\}$

There is no transition $(q_0, b, p) \in \Delta$ for any $p \in K$, so we get $\delta'(\{q_0\}, b) = E(p) = \emptyset$

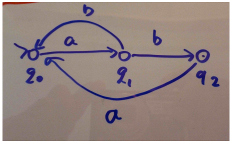
Example STAGE 2

By the **Step 1** we have that all states directly reachable from s' are $\{q_2\}$ and $\emptyset$

Step 2 Evaluate δ' on all states that result from the **Step 1**; i.e. on states $\{q_1\}$ and $\emptyset$

Obviously $\delta'(\emptyset, a) = \emptyset$ and $\delta'(\emptyset, b) = \emptyset$

To evaluate $\delta'(\{q_1\}, a)$, $\delta'(\{q_1\}, b)$ we first look at all transitions $(q_1, a, p) \in \Delta$ on the diagram

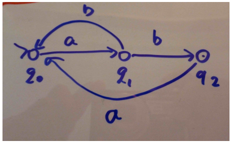


There is no transition $(q_1, a, p) \in \Delta$ for any $p \in K$, so

$$\delta'(\{q_1\}, a) = \emptyset \text{ and } \delta'(\emptyset, a) = \emptyset, \ \delta'(\emptyset, b) = \emptyset$$

Example STAGE 2

Step 2 To evaluate $\delta'(\{q_1\}, b)$ we now look at all transitions $(q_1, b, p) \in \Delta$ on the diagram



Here they are: (q_1, b, q_2), (q_1, b, q_0)

$\delta'(Q, \sigma) = \bigcup_{p \in K}\{E(p) : \exists_{q \in Q}\ (q, \sigma, p) \in \Delta\}$

$\delta'(\{q_1\}, b) = E(q_2) \cup E(q_0) = \{q_2\} \cup \{q_0\} = \{q_0, q_2\}$

We evaluated

$$\delta'(\{q_1\}, b) = \{q_0, q_2\}, \quad \delta'(\{q_1\}, a) = \emptyset$$

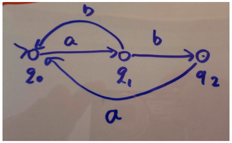
We also have that the state $\{q_0, q_2\} \in F'$

Example STAGE 2

Step 3 Evaluate δ' on all states that result from the **Step 2**; i.e. on states $\{q_0, q_2\}$, $\emptyset$

Obviously $\delta'(\emptyset, a) = \emptyset$ and $\delta'(\emptyset, b) = \emptyset$

To evaluate $\delta'(\{q_0, q_2\}, a)$ we look at all transitions (q_0, a, p) and (q_2, a, p) on the diagram



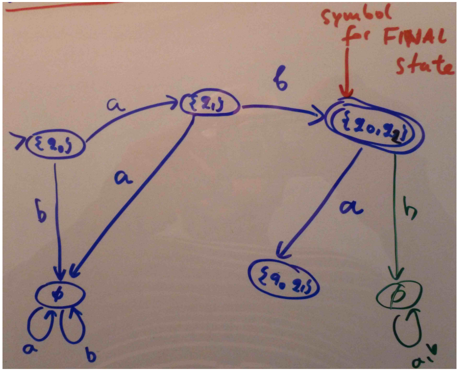
Here they are: (q_0, a, q_1), (q_2, a, q_0)

$$\delta'(\{q_0, q_2\}, a) = E(q_1) \cup E(q_0) = \{q_0, q_1\}$$

Similarly $\delta'(\{q_0, q_2\}, b) = \emptyset$

Diagram Steps 1 - 3

Here is the **Diagram** of M’ after finishing STAGE 1 and **Steps 1-3** of the STAGE 2

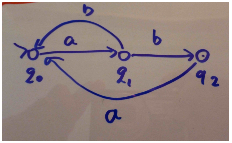


Example STAGE 2

Step 4 Evaluate δ' on all states that result from the **Step 3**; i.e. on states $\{q_0, q_1\}$, $\emptyset$

Obviously $\delta'(\emptyset, a) = \emptyset$ and $\delta'(\emptyset, b) = \emptyset$

To evaluate $\delta'(\{q_0, q_1\}, a)$ we look at all transitions (q_0, a, p) and (q_1, a, p) on the diagram



Here there is one (q_0, a, q_1), and **there is no** transition (q_1, a, p) for any $p \in K$, so

$$\delta'(\{q_0, q_1\}, a) = E(q_1) \cup \emptyset = \{q_1\}$$

Similarly

$$\delta'(\{q_0, q_1\}, b) = \{q_0, q_2\}$$

Example STAGE 2

Step 5 Evaluate δ' on all states that result from the **Step 4**; i.e. on states $\{q_1\}$ and $\{q_0, q_2\}$

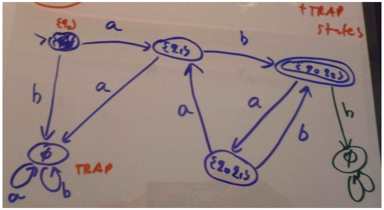
Observe that we have already evaluated $\delta'(\{q_1\}, \sigma)$ for all $\sigma \in \Sigma$ in **Step 2** and $\delta'(\{q_0, q_2\}, \sigma)$ in **Step 3**

The process of defining $\delta'(Q, \sigma)$ for **relevant** $Q \in 2^K$ is hence **terminated**

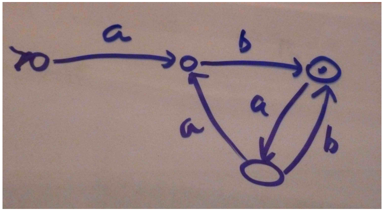
All other states are **trap states**

Diagram of of M’

Here is the **Diagram** of the **Relevant Part** of M’



and here is its **short pattern diagram** version



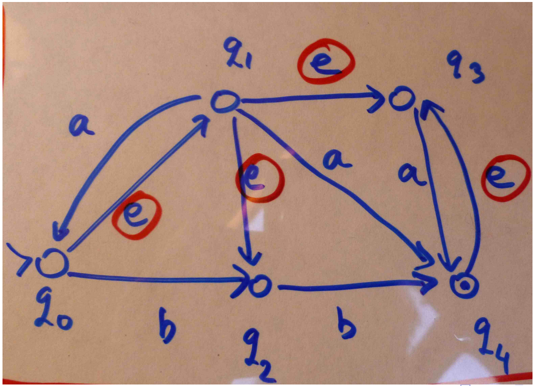
Book Example

Book Example

Here is the nondeterministic M from book page 70

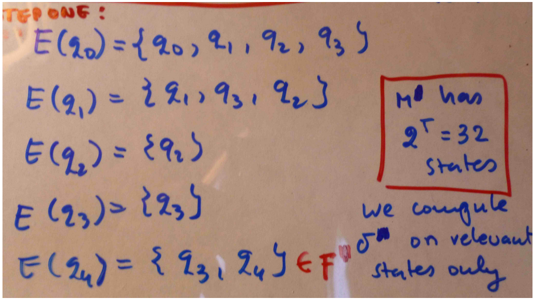
Exercise Read the example and re- write it as an exercise stage by stage as we did in class - it means follow the previous example

Diagram of M



Book Example

STAGE 1



TEP ONE:

$E(q_0) = \{q_0, q_1, q_2, q_3\}$

$E(q_1) = \{q_1, q_3, q_2\}$

$E(q_2) = \{q_2\}$

$E(q_3) = \{q_3\}$

$E(q_4) = \{q_3, q_4\} \in F'$

$\boxed{M' \text{ has } 2^5 = 32 \text{ states}}$

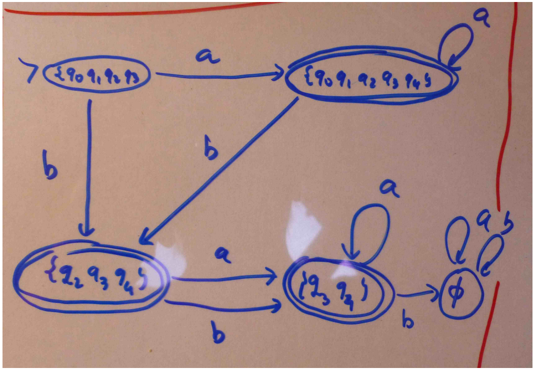
we compute δ' on relevant states only

STAGE 2 evaluation are on page 72

Evaluate them independently of the book

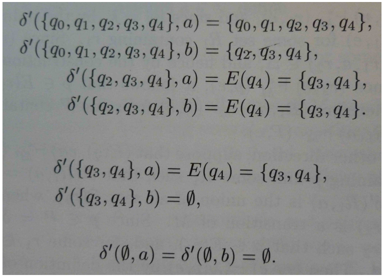
Book Example

Diagram of M’



Book Example

Some **book computations**

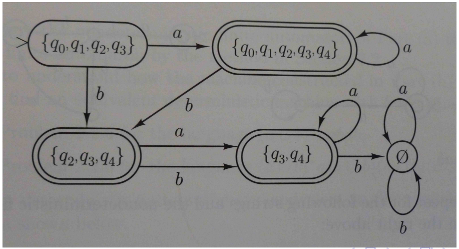


$$\delta'(\{q_0,q_1,q_2,q_3,q_4\},a) = \{q_0,q_1,q_2,q_3,q_4\},$$
$$\delta'(\{q_0,q_1,q_2,q_3,q_4\},b) = \{q_2,q_3,q_4\},$$
$$\delta'(\{q_2,q_3,q_4\},a) = E(q_4) = \{q_3,q_4\},$$
$$\delta'(\{q_2,q_3,q_4\},b) = E(q_4) = \{q_3,q_4\}.$$

$$\delta'(\{q_3,q_4\},a) = E(q_4) = \{q_3,q_4\},$$
$$\delta'(\{q_3,q_4\},b) = \emptyset,$$

$$\delta'(\emptyset,a) = \delta'(\emptyset,b) = \emptyset.$$

Book Diagram



NDFA and DFA Differences Revisited

Difference 1 Revisited

DFA transition function δ even if expressed as a relation

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

must be a function, while the NDFA transition relation Δ

$\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$

may **not be a function**

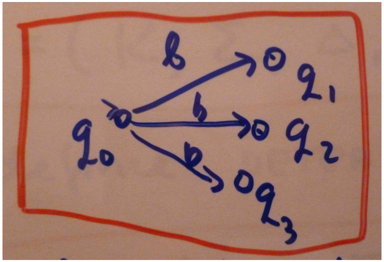
Difference 2 Revisited

DFA transition function δ **domain** is the set $K \times \Sigma$ while

It is obvious that the definition of δ' solves the **Difference 2**

Difference 1

Given a **non-function diagram** of M

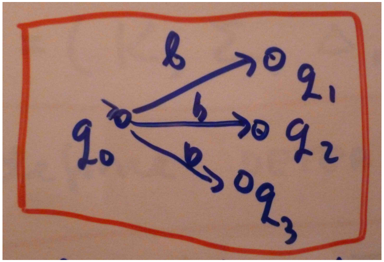


Proposed IDEA of f solving the **Difference 1** was to make the states of M' as some subsets of the set of states of M and put in this case

$$\delta'(\{q_0\}, b) = \{q_1, q_2, q_3\}$$

Exercise

Given the **diagram** of M


$$\begin{tikzcd} & q_1 \\ q_0 \arrow[ur, "b"] \arrow[r, "b"] \arrow[dr, "b"] & q_2 \\ & q_3 \end{tikzcd}$$

Exercise

Show that the definition of δ'

$$\delta'(Q,\sigma) = \bigcup_{p\in K}\{E(p) : \exists_{q\in Q}\ (q,\sigma,p)\in\Delta\}$$

does exactly what we have proposed, i.e show that

$$\delta'(\{q_0\},b) = \{q_1,q_2,q_3\}$$

Proof of Equivalency Theorem

Equivalency Theorem
For any **nondeterministic** automaton

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

there is (we have given an algorithm for its construction) a **deterministic** automaton

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

such that

$$M \approx M' \quad \text{i.e.} \quad L(M) = L(M')$$

Proof
M' is deterministic directly from the definition because the formula

$$\delta'(Q,\sigma) = \bigcup_{p \in K} \{E(p) :\ \exists_{q \in Q}\ (q,\sigma,p) \in \Delta\}$$

defines a function and is well defined for a all $Q \in 2^K$ and $\sigma \in \Sigma$.

Proof of Equivalency Theorem

We now claim that the following Lemma holds and we will prove equivalency $M \approx M'$ from the Lemma

Lemma

For any word $w \in \Sigma^*$ and any states $p, q \in K$

$$(q, w) \vdash_M{}^* (p, e) \quad \text{if and only if} \quad (E(q), w) \vdash_{M'}{}^* (P, e)$$

for some set P such that $p \in P$

We carry the **proof** of the **Lemma** by induction on the length $|w|$ of w

Base Step $|w| = 0$; this is possible only when t $w = e$ and we must show

$$(q, e) \vdash_M{}^* (p, e) \quad \text{if and only if} \quad (E(q), e) \vdash_{M'}{}^* (P, e)$$

for some P such that $p \in P$

Proof of Lemma

Base Step We must show that

$(q, e) \vdash_M{}^* (p, e)$ if and only if $\exists_P (p \in P \cap (E(q), e) \vdash_{M'}{}^* (P, e)))$

Observe that $(q, e) \vdash_M{}^* (p, e)$ just says that $p \in E(q)$ and the right side of statement holds for $P = E(q)$

Since M' is deterministic the statement $\exists_P (p \in P \cap (E(q), e) \vdash_{M'}{}^* (P, e)))$ is equivalent to saying that $P = E(q)$ and since $p \in P$ we get $p \in E(q)$ what is equivalent to the left side

This completes the proof of the basic step

Inductive step is similar and is given as in the book page 71

Proof of The Theorem

We have just proved that for any $w \in \Sigma^*$ and any states $p, q \in K$

$$(q, w) \vdash_M{}^* (p, e) \quad \text{if and only if} \quad (E(q), w) \vdash_{M'}{}^* (P, e)$$

for some set P such that $p \in P$

The **proof** of the **Equivalency Theorem** continues now as follows

Proof of The Theorem

We have to prove that L(M) = L(M')

Let' s take a word $w \in \Sigma^*$

We have (by definition of $L(M)$) that $w \in L(M)$

if and only if $(s, w) \vdash_M{}^* (f, e)$ for $f \in F$

if and only if $(E(s), w) \vdash_M{}^* (Q, e)$ for some Q such that $f \in Q$ (by the **Lemma**)

if and only if $(s', w) \vdash_M{}^* (Q, e)$ for some $Q \in F$ (by definition of M')

if and only if $w \in L(M')$

Hence L(M) = L(M')

This end the **proof** of the **Equivalency Theorem**

Finite Automata

We have proved that the class **(CD)** and book **(BD)** definitions of a nondeterministic automaton are **equivalent**

Hence by the **Equivalency Theorem** deterministic and ondeterministic automata defined by **any** of the both ways are **equivalent**

We will use now a name

FINITE AUTOMATA

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

Chapter 2
Finite Automata

Slides Set 2

PART 3: Finite Automata and Regular Expressions

PART 4: Languages that are Not Regular

Chapter 2
Finite Automata

Slides Set 2

PART 3: Finite Automata and Regular Expressions

Finite Automata and Regular Expressions

The goal of this part of chapter 2 is to prove a **theorem** that establishes a **relationship** between Finite Automata and Regular languages, i.e to **prove** that following

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

$$L = L(M)$$

Closure Theorem

To achieve our goal we first prove the following

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**

Closure Theorem

Remember that languages are **sets**, so we have the set em[] operations $\cup$, $\cap$, $-$, defined for any $L_1, L_2 \subseteq \Sigma^*$, i.e the languages

$$L = L_1 \cup L_2, \quad L = L_1 \cap L_2, \quad L = \Sigma^* - L_1$$

We also defined the languages specific operations of concatenation and Kleene's Star , i.e. the

languages

$$L = L_1 \circ L_2 \quad \text{and} \quad L = {L_1}^*$$

Closure Under Union

1. The class of languages accepted by Finite Automata (FA) is **closed** under union

Proof

Let M_1, M_2 be two NDFA finite automata

We **construct** a NDF automaton M, such that

$$L(M) = L(M_1) \cup L(M_2)$$

Let $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and

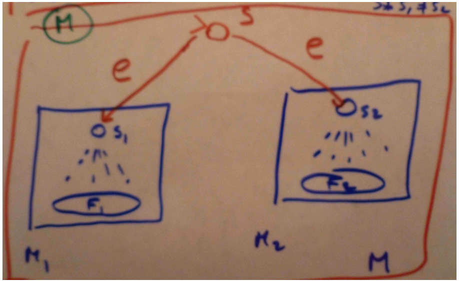
$M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$

Where (we rename the states, if needed)

$\Sigma = \Sigma_1 \cup \Sigma_2, \quad s_1 \neq s_2, \quad K_1 \cap K_2 = \emptyset \quad F_1 \cap F_2 = \emptyset$

Closure Under Union

We **picture** M, such that $L(M) = L(M_1) \cup L(M_2)$ as follows



M goes nondeterministically to M_1 or to M_2 reading nothing so we get

$$w \in L(M) \quad \text{if and only if} \quad w \in M_1 \quad \text{or} \quad w \in M_2$$

and hence

$$L(M) = L(M_1) \cup L(M_2)$$

Closure Under Union

We **define formally**

$$M = M_1 \cup M_2 = (K,\ \Sigma,\ \Delta,\ s,\ F)$$

where

$K = K_1 \cup K_2 \cup \{s\}$ for $s \notin K_1 \cup K_2$

s is a **new** state and

$F = F_1 \cup F_2,\quad \Delta = \Delta_1 \cup \Delta_2 \cup \{(s, e, s_1),\ (s, e, s_2)\}$

for s_1 - initial state of M_1 and

s_2 the initial state of M_2

Observe that by Mathematical Induction we construct,

for any $n \geq 2$ an automaton $M = M_1 \cup M_2 \cup \ \dots\ M_n$ such that

$$L(M) = L(M_1) \cup L(M_2) \cup \ \dots\ L(M_n)$$

Closure Under Union

Formal proof

Directly from the definition we get

$w \in L(M)$ if and only if

$\exists_q((q \in F = F_1 \cup F_2) \cap ((s, w) \vdash_M{}^*(q, e))$ if and only if

$\exists_q(((q \in F_1) \cup (q \in F_2)) \cap ((s, w) \vdash_M{}^*(q, e))$ if and only if

$\exists_q((q \in F_1) \cap ((s, w) \vdash_M{}^*(q, e)) \cup$

$\exists_q((q \in F_2) \cap ((s, w) \vdash_M{}^*(q, e)))$ if and only if

$w \in L(M_1) \cup w \in L(M_2)$, what proves that

$$L(M) = L(M_1) \cup L(M_2)$$

We used the following Law of Quantifiers

$$\exists_x(A(x) \cup B(x)) \equiv (\exists_x A(x) \cup \exists_x B(x))$$

Examples

Example 1

Diagram of M_1 such that $L(M_1) = aba^*$ is

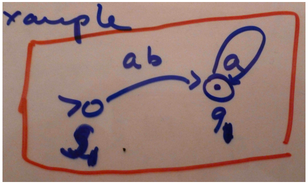
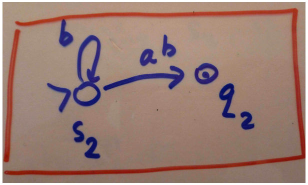


Diagram of M_2 such that $L(M_2) = b^*ab$ is



We construct $M = M_1 \cup M_2$ such that

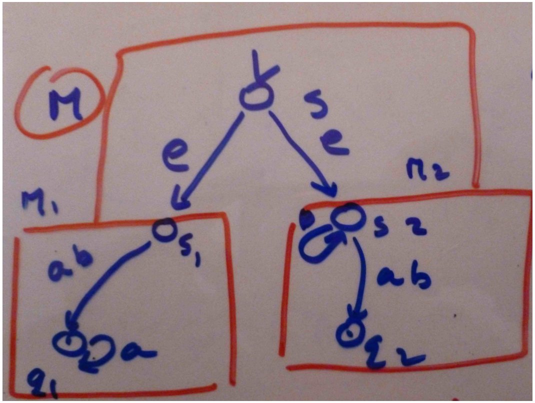
$$L(M) = aba^* \cup b^*ab = L(M_1) \cup L(M_2)$$

as follows

Examples

Example 1

Diagram of M such that $L(M) = aba^* \cup b^*ab$ is



Examples

Example 2

Diagram of M_1 such that $L(M_1) = b^*abc$ is

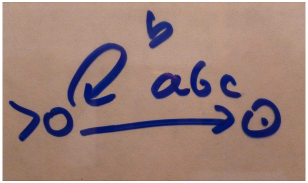
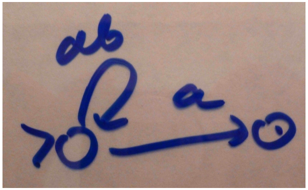


Diagram of M_2 such that $L(M_2) = (ab)^*a$ is



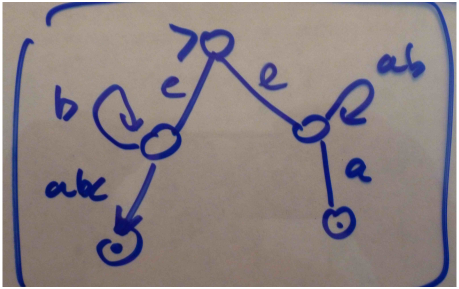
We construct $M = M_1 \cup M_2$ such that

$$L(M) = b^*abc \cup (ab)^*a = L(M_1) \cup L(M_2)$$

as follows

Examples

Diagram of M such that $L(M) = b^*abc \cup (ab)^*a$ is



This is a schema diagram

If we need to **specify** the components we put **names** on states on the diagrams

Closure Under Concatenation

2. The class of languages accepted by Finite Automata is **closed** under concatenation

Proof

Let $M_1,\ M_2$ be two NDFA

We **construct** a NDF automaton M, such that

$$L(M) = L(M_1) \circ L(M_2)$$

Let $M_1 = (K_1,\ \Sigma,\ \Delta_1,\ s_1,\ F_1)$ and

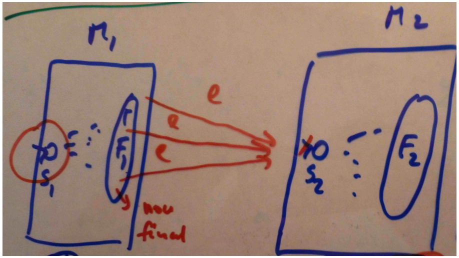
$M_2 = (K_2,\ \Sigma,\ \Delta_2,\ s_2,\ F_2)$

Where (if needed we re-name states)

$\Sigma = \Sigma_1 \cup \Sigma_2, \quad s_1 \neq s_2, \quad K_1 \cap K_2 = \emptyset \quad F_1 \cap F_2 = \emptyset$

Closure Under Concatenation

We **picture** M, such that $L(M) = L(M_1) \circ L(M_2)$ as follows



The final states from F_1 of M_1 become **internal** states of M

The initial state s_2 of M_2 becomes an **internal** state of M

M goes nondeterministically from ex-final states of M_1 to the ex-initial state of M_2 **reading** nothing

Closure Under Concatenation

We **define formally**

$$M = M_1 \circ M_2 = (K,\ \Sigma,\ \Delta,\ s_1,\ F_2)$$

where

$K = K_1 \cup K_2$

s_1 of M_1 is the initial state

F_2 of M_2 is the set of final states

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(q, e, s_2) :$ for $q \in F_1\}$

Directly from the definition we get

$w \in L(M)$ iff $w = w_1 \circ w_2$ for $w_1 \in L_1,\ w_2 \in L_2$

and hence

$$L(M) = L(M_1) \circ L(M_2)$$

Examples

Diagram of M_1 such that $L(M_1) = aba^*$ is

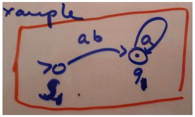
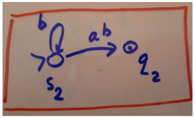


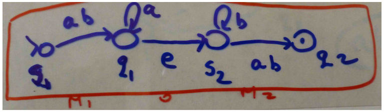
Diagram of M_2 such that $L(M_2) = b^*ab$ is



We construct $M = M_1 \circ M_2$ such that

$$L(M) = aba^* \circ b^*ab = L(M_1) \circ L(M_2)$$

as follows



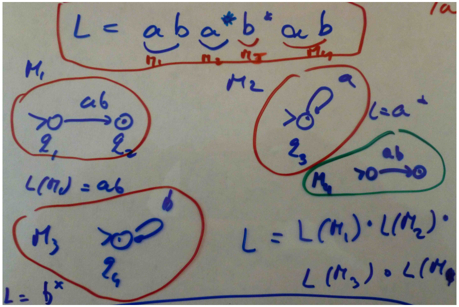
Examples

Given a language $L = aba^*b^*ab$

Observe that we can reprezent L as, for example, the following concatenation

$$L = ab \circ a^* \circ b^* \circ ab$$

Then we construct "easy" automata M_1, M_2, M_3, M_4 as follows



Examples

We know, by Mathematical Induction that we can construct, for any $n \geq 2$ an automaton

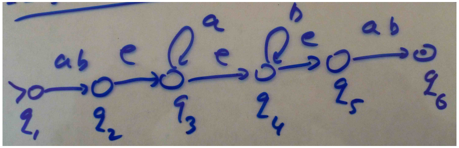
$$M = M_1 \circ M_2 \circ \ \circ M_n$$

such that

$$L(M) = L(M_1) \circ \ \dots \ \circ \ L(M_n)$$

In our case n=4 and we get

Diagram of M



and $L(M) = aba^*b^*ab$

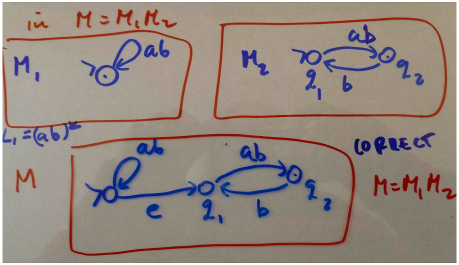
Question

Question

Why we have to go be the transactions (q, e, s_2) between M_1 and M_2 while constructing $M = M_1 \circ M_2$?

Example of a construction when we can't SKIP the transaction (q, e, s_2)

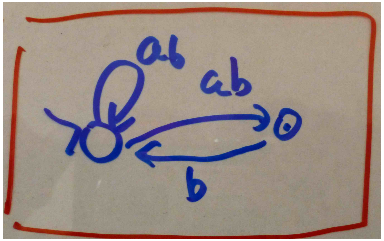
Here is a **correct** construction of $M = M_1 \circ M_2$



Observe that $abbabab \notin L(M)$

Question

Here is a construction of $M' = M_1 \circ M_2$ without the transaction (q, e, s_2)



Observe that $abbabab \in L(M')$ and $abbabab \notin L(M)$

We hence proved that skipping the transactions (q, e, s_2) between M_1 and M_2 leads to automata accepting different languages

Closure Under Kleene's Star

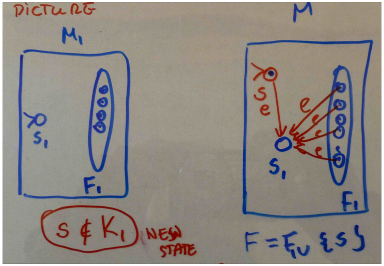
3. The class of languages accepted by Finite Automata is **closed** under Kleene's Star

Proof Let $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$

We **construct** a NDF automaton $M = M_1{}^*$, such that

$$L(M) = L(M_1)^*$$

Here is a **diagram**



Closure Under Kleene's Star

Given $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$

We **define formally**

$$M = M_1{}^* = (K, \Sigma, \Delta, s, F)$$

where

$K = K_1 \cup \{s\}$ for $s \notin K_1$

s is new initial state, s_1 becomes an internal state

$F = F_1 \cup \{s\}$

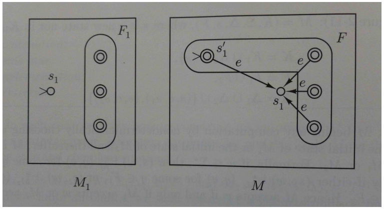
$\Delta = \Delta_1 \cup \{(s, e, s_1)\} \cup \{(q, e, s_1) :$ for $q \in F_1\}$

Directly from the definition we get

$$L(M) = L(M_1)^*$$

Closure Under Kleene's Star

The Book **diagram** is



Given $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$

We define

$$M_1{}^* = (K_1 \cup \{s\}, \Sigma, \Delta, s, F_1 \cup \{s\})$$

where s is a new initial state and

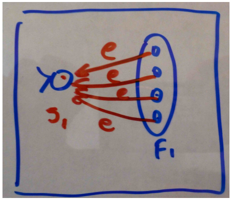
$\Delta = \Delta_1 \cup \{(s, e, s_1)\} \cup \{(q, e, s_1) :$ for $q \in F_1\}$

Two Questions

Here **two questions** about the construction of $M = M_1{}^*$

Q1 Why do we need to make the NEW initial state s of M also a FINAL state?

Q2 Why can't SKIP the introduction of the NEW initial state and design $M = M_1{}^*$ as follows



Q1 + Q2 give us answer why we construct $M = M_1{}^*$ as we did, i.e. provides the motivation for the correctness of the construction

Question 1 Answer

Observe that the definition of $M = M_1{}^*$ must be correct for ALL automata M_1 and hence in particular for M_1 such that $F_1 = \emptyset$,

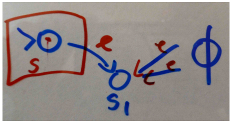
In this case we have that $L(M_1) = \emptyset$

But we know that

$$L(M) = L(M_1)^* = \emptyset^* = \{e\}$$

This proves that $M = M_1{}^*$ must accept e, and hence we must make s of M also a FINAL state

Diagram



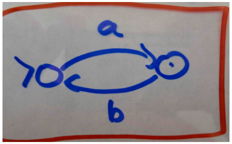
Question 2 Answer

Q2 Why can't SKIP the introduction of the NEW initial state and design $M = {M_1}^*$

Here is an **example**

Let M_1 , such that $L(M_1) = a(ba)^*$

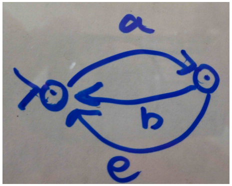
M_1 is defined by a **diagram**



$$L(M_1)^* = (a(ba)^*)^*$$

Question 2 Answer

Here is a **diagram** of M where we skipped the introduction of a new initial state



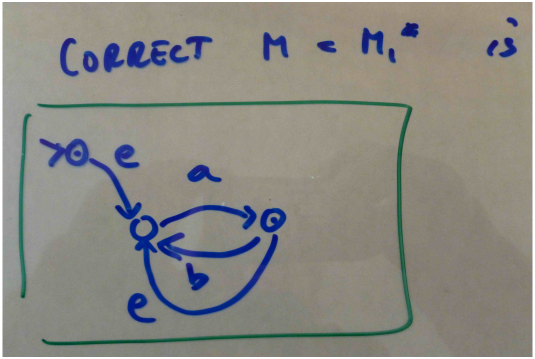
Observe that $ab \in L(M)$, but

$$ab \notin (a(ba)^*)^* = L(M_1)^*$$

This proves **incorrectness** of the above construction

Correct Diagram

The CORRECT **diagram** of $M = M_1{}^*$ is



Exercise 1

Exercise 1

Construct M such that

$$L(M) = (ab^*ba \cup a^*b)^*$$

Observe that

$$L(M) = (L(M_1) \cup L(M_2))^*$$

and

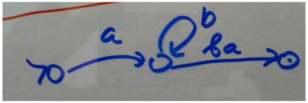
$$M = (M_1 \cup M_2)^*$$

Exercise 1

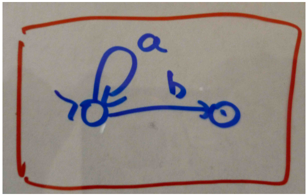
Solution

We construct M such that $L(M) = (ab^*ba \cup a^*b)^*$ in the following steps using the **Closure Theorem** definitions

Step 1 Construct M_1 for $L(M_1) = ab^*ba$

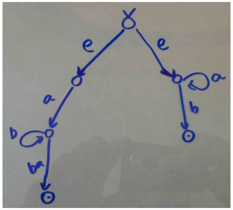


Step 2 Construct M_2 for $L(M_2) = a^*b$

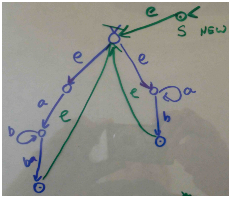


Exercise

Step 3 Construct $M_1 \cup M_2$



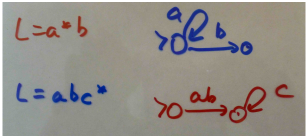
Step 4 Construct $M = (M_1 \cup M_2)^*$



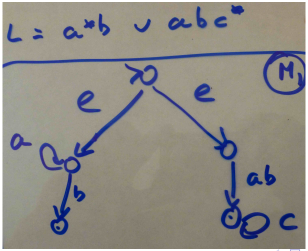
$$L(M) = (ab^*ba \cup a^*b)^*$$

Exercise 2

Exercise 2
Construct M such that $L(M) = (a^*b \cup abc^*)a^*b^*$
Solution We construct M in the following steps using the **Closure Theorem** definitions
Step 1 Construct N_1, N_2 for $L = a^*b$ and $L = abc^*$

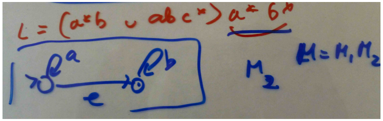


Step 2 Construct $M_1 = N_1 \cup N_2$

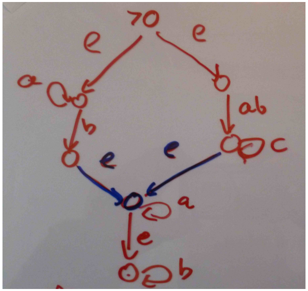


Exercise 2

Step 3 Construct M_2 for $L = a^*b^*$



Step 4 Construct $M = (M_1 \circ M_2)^*$



$$L(M) = (a^*b \cup abc^*)a^*b^*$$

Back to Closure Theorem

CLOSURE THEOREM

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

1.	union	**proved**
2.	concatenation	**proved**
3.	Kleene’s Star	**proved**
4.	complementation	
5.	intersection	

Observe that we used the term **Finite Automata** (FA) so in the

proof we can choose a DFA or NDFA, as we have already proved their **equivelency**

Closure Under Complementation

4. The class of languages accepted by Finite Automata is **closed** under complementation

Proof Let

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

be a **deterministic** finite automaton DFA

The complementary language $\overline{L} = \Sigma^* - L(M)$ is accepted by the DFA denoted by $\overline{M}$ that is identical with M except that final and nonfinal states are interchanged, i.e. we define

$$\overline{M} = (K,\ \Sigma,\ \delta,\ s,\ K - F)$$

and we have

$$L(\overline{M}) = \Sigma^* - L(M)$$

Closure Under Intersection

4. The class of languages accepted by Finite Automata is **closed** under intersection

Proof 1

Languages are sets so we have have the following property

$$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$$

Given finite automata M_1, M_2 such that

$$L_1 = L(M_1) \quad \text{and} \quad L_2 = L(M_2)$$

We construct M such that $L(M) = L_1 \cap L_2$ as follows

1. Transform M_1, M_2 into equivalent DFA automata N_1, N_2

2. Construct $\overline{N_1}$, $\overline{N_2}$ and then $N = \overline{N_1} \cup \overline{N_2}$

3. Transform NDF automaton N into equivalent DFA automaton N'

4. $M = \overline{N'}$ is the required finite automata

This is an indirect Construction

Homework: describe the direct construction

Closure Theorem

CLOSURE THEOREM

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

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

Observe that we used the term **Finite Automata** (FA) so in the

proof we can choose a DFA or NDFA, as we have already proved their **equivelency**

Intersection Direct Construction

Direct Construction

Case 1 deterministic

Given **deterministic** automata $M_1,\ M_2$ such that

$$M_1 = (K_1,\ \Sigma_1,\ \delta_1,\ s_1,\ F_1), \qquad M_2 = (K_2,\ \Sigma_2,\ \delta_2,\ s_2,\ F_2)$$

We construct $M = M_1 \cap M_2$ such that $L(M) = L(M_1) \cap L(M_2)$ as follows

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

where . $\Sigma = \Sigma_1 \cup \Sigma_2$

$$K = K_1 \times K_2, \qquad s = (s_1, s_2), \qquad F = F_1 \times F_2$$

$$\delta((q_1, q_2),\ \sigma) = (\delta_1(q_1,\ \sigma),\ \delta_2(q_2,\ \sigma))$$

Intersection Direct Construction

Proof of correctness of the construction

$w \in L(M)$ if and only if

$((s_1, s_2), w) \vdash_M{}^* ((f_1, f_2), e))$ and $f_1 \in F_1, f_2 \in F_2$

if and only if

$(s_1, w) \vdash_{M_1}{}^* (f_1, e)$ for $f_1 \in F_1$ and

$(s_2, w) \vdash_{M_2}{}^* (f_2, e)$ for $f_2 \in F_2$

if and only if

$w \in L(M_1)$ and $w \in L(M_2)$

if and only if

$w \in L(M_1) \cap L(M_2)$

Intersection Direct Construction

Direct Construction

Case 2 nondeterministic

Given **nondeterministic** automata $M_1,\ M_2$ such that

$$M_1 = (K_1,\ \Sigma_1,\ \Delta_1,\ s_1,\ F_1), \qquad M_2 = (K_2,\ \Sigma_2,\ \Delta_2,\ s_2,\ F_2)$$

We construct $M = M_1 \cap M_2$ such that $L(M) = L(M_1) \cap L(M_2)$ as follows

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

where $\Sigma = \Sigma_1 \cup \Sigma_2$

$$K = K_1 \times K_2, \quad s = (s_1, s_2), \quad F = F_1 \times F_2$$

and Δ is defined as follows

Intersection Direct Construction

Δ is defined as follows

$$\Delta = \Delta' \cup \Delta'' \cup \Delta'''$$

$\Delta' = \{((q_1, q_2), \sigma, (p_1, p_2)) : \ (q_1, \sigma, p_1) \in \Delta_1$ and $(q_2, \sigma, p_2) \in \Delta_2, \ \sigma \in \Sigma\}$

$\Delta'' = \{((q_1, q_2), \sigma, (p_1, p_2)) : \ \sigma = e, \ (q_1, e, p_1) \in \Delta_1$ and $q_2 = p_1\}$

$\Delta'' = \{((q_1, q_2), \sigma, (p_1, p_2)) : \ \sigma = e, \ (q_2, e, p_2) \in \Delta_2$ and $q_1 = p_1\}$

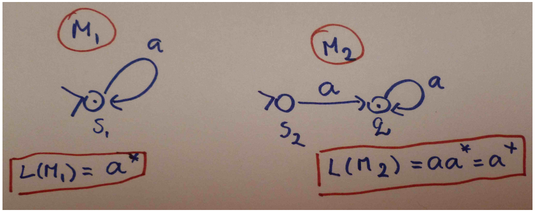
Observe that if M_1, M_2 have each at most n states, our direct construction of produces $M = M_1 \cap M_2$ with at most n^2 states.

The **indirect** construction from the proof of the theorem might generate M with up to $2^{2^{n+1}+1}$ states

Direct Construction Example

Example

Let M_1, M_2 be given by the following **diagrams**



Observe that $L(M_1) \cap L(M_2) = a^* \cap a^+ = a^+$

Direct Construction Example

Formally $M_1, \ M_2$ are defined as follows

$$M_1 = (\{s_1\}, \ \{a\}, \ \delta_1, \ s_1, \ \{s_1\}), \ \ M_2 = (\{s_2, q\}, \ \{a\}, \ \delta_2, \ s_2, \ \{q\})$$

for $\delta_1(s_1, a) = s_1$ and $\delta_2(s_2, a) = q, \ \ \delta_2(q, a) = q$

By the deterministic case **definition** we have that
$M = M_1 \cap M_2$ is

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

for $\Sigma = \{a\}$

$$K = K_1 \times K_2 = \{s_1\} \times \{s_2, q\} = \{(s_1, s_2), \ (s_1, g) \ \}$$

$$s = (s_1, s_2), \quad F = \{s_1\} \times \{q\} = \{(s_1, q)\}$$

Direct Construction Example

By definition

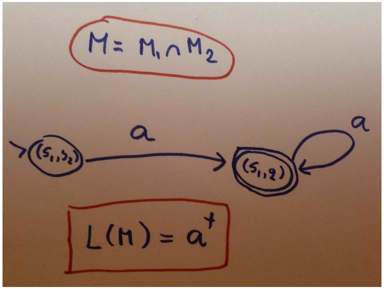
$$\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$$

In our case we have

$$\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a)) = (s_1, q),$$

$$\delta((s_1, q), a) = (\delta_1(s_1, a), \delta_2(q, a)) = (s_1, q)$$

The **diagram** of $M = M_1 \cap M_2$ is



Main Theorem

Now our goal is to prove a theorem that established the relationship between languages and finite automata

This is the most important Theorem of this section so we call it a Main Theorem

Main Theorem

A language L is regular

if and only if

L is accepted by a finite automata

Main Theorem

The **Main Theorem** consists of the following two parts

Theorem 1

For any a regular language L

there is a e finite automata M, such that $L = L(M)$

Theorem 2

For any a finite automata M, the language L(M) is regular

Main Theorem

Definition

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

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

Reminder: the function $\mathcal{L} : \mathcal{R} \longrightarrow 2^{\Sigma^*}$ is defined recursively as follows

1. $\mathcal{L}(\emptyset) = \emptyset, \quad \mathcal{L}(\sigma) = \{\sigma\}$ for all $\sigma \in \Sigma$

2. If $\alpha, \beta \in \mathcal{R}$, then

$$\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha) \circ \mathcal{L}(\beta) \quad \text{concatenation}$$

$$\mathcal{L}(\alpha \cup \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta) \quad \text{union}$$

$$\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^* \quad \text{Kleene's Star}$$

Regular Expressions Definition

Reminder

We define a $\mathcal{R}$ of **regular expressions** over an alphabet Σ as follows

$\mathcal{R} \subseteq (\Sigma \cup \{(,\ ),\ \emptyset,\ \cup,\ *\})^*$ and $\mathcal{R}$ is the smallest set such that

1. $\emptyset \in \mathcal{R}$ and $\Sigma \subseteq \mathcal{R}$, i.e. we have that

$$\emptyset \in \mathcal{R} \text{ and } \forall_{\sigma\in\Sigma}\ (\sigma \in \mathcal{R})$$

2. If $\alpha, \beta \in \mathcal{R}$, then

$(\alpha\beta) \in \mathcal{R}$ concatenation

$(\alpha \cup \beta) \in \mathcal{R}$ union

$\alpha^* \in \mathcal{R}$ Kleene's Star

Proof of Main Theorem Part 1

Now we are going to **prove** the first part of the Main Theorem, i.e.

Theorem 1

For any a regular language L

there is a finite automata M, such that $L = L(M)$

Proof

By definition of regular language, L is regular if and only if

there is a regular expression $r \in \mathcal{R}$ that represents L, what

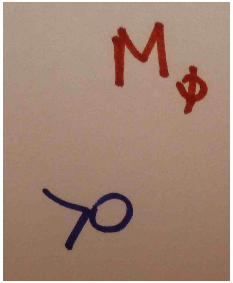
we write in **shorthand** notation as L = r

Given a regular language, L, we **construct** a finite automaton M such that L(M) = L recursively following the definition of the set $\mathcal{R}$ of **regular expressions** as follows

Proof Theorem 1

1. $r = \emptyset$, i.e. the language is $L = \emptyset$

Diagram of M, such that $L(M) = \emptyset$ is

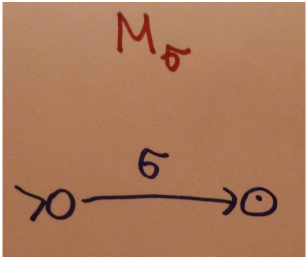


We denote M as $M = M_\emptyset$

Proof Theorem 1

2. $r = \sigma$, for any $\sigma \in \Sigma$ i.e. the language is $L = \sigma$

Diagram of M, such that $L(M) = \emptyset$ is



We denote M as $M = M_\sigma$

Proof Theorem 1

3. $r \neq \emptyset, \quad r \neq \sigma$

By the recursive definition, we have that $L = r$ where

$$r = \alpha \cup \beta, \quad r = \alpha \circ \beta, \quad r = \alpha^*$$

for any $\alpha, \beta \in \mathcal{R}$

We construct as in the proof of the **Closure Theorem** the automata

$$M_r = M_\alpha \cup M_\beta, \quad M_r = M_\alpha \circ M_\beta, \quad M_r = (M_r)^*$$

respectively, and it ends the proof

Example

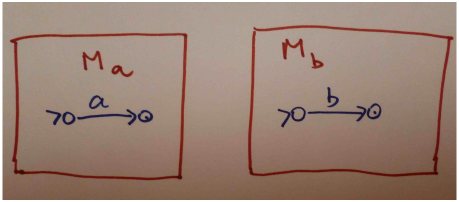
Use construction defined in the proof of **Theorem 1** to construct an automaton M such that

$$L(M) = (ab \cup aab)^*$$

We construct M in the following stages

Stage 1

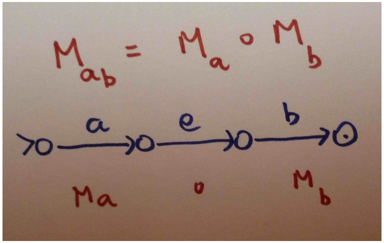
For $a, b \in \Sigma$ we construct M_a and M_b



Example

Stage 2

For ab, aab we use M_a and M_b and **concatenation** construction to construct M_{ab}

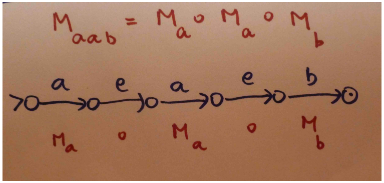


$$M_{ab} = M_a \circ M_b$$

$$\begin{tikzcd} \arrow[r] & \circ \arrow[r, "a"] & \circ \arrow[r, "e"] & \circ \arrow[r, "b"] & \odot \end{tikzcd}$$

$$M_a \circ M_b$$

and M_{aab}



$$M_{aab} = M_a \circ M_a \circ M_b$$

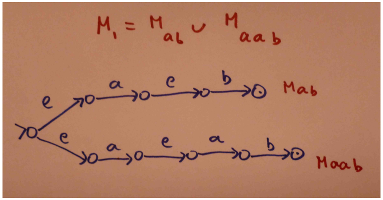
$$\begin{tikzcd} \arrow[r] & \circ \arrow[r, "a"] & \circ \arrow[r, "e"] & \circ \arrow[r, "a"] & \circ \arrow[r, "e"] & \circ \arrow[r, "b"] & \odot \end{tikzcd}$$

$$M_a \circ M_a \circ M_b$$

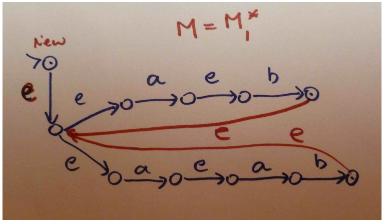
Example

Stage 3

We use **union** construction to construct $M_1 = M_{ab} \cup M_{aab}$



Stage 4 We use Kleene's **star** construction to construct $M = M_1{}^*$



Exercise

Use construction defined in the proof of **Theorem 1** to construct an automaton M such that

$$L(M) = (a^* \cup abc \cup a^*b)^*$$

We construct (draw diagrams) M in the following stages

Stage 1

Construct $M_a, \quad M_b, \quad M_c$

Stage 2

Construct $M_1 = M_{abc}$

Stage 3

Construct $M_2 = {M_a}^*$

Stage 4

Construct $M_3 = {M_a}^* M_b$

Stage 5

Construct $M_4 = M_1 \cup M_2 \cup M_3$

Stage 6

Construct $M = {M_4}^*$

Main Theorem Part 2

Theorem 2

For any a finite automaton M there is a regular expression $r \in \mathcal{R}$, such that

$$L(M) = r$$

Proof

The proof is **constructive**; given M we will give an algorithm how to recursively generate the regular expression r , such that $L(M) = r$

We assume that M is nondeterministic

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

We use the BOOK definition, i.e.

$$\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$$

Proof of Theorem 2

We put states of M into a one- to - one sequence

$$K: \quad s = q_1,\ q_2,\ \dots\ q_n \text{ for } n \geq 1$$

We build r using the following expressions

$$R(i,\ j,\ k) \quad \text{for } i, j = 1,\ 2,\ \dots\ n, \quad k = 0,\ 1,\ 2,\ \dots\ n$$

$$R(i,\ j,\ k) = \{w \in \Sigma^*; \quad (q_i,\ w) \vdash_{M,k}{}^{*} (q_j, w')\}$$

R(i, j, k) is the set of all words ”spelled” by all PATHS from q_i to q_j in such way that we **do not pass** through an intermediate state numbered k+1 or greater

Observe that $\neg(m \geq k+1) \equiv m \leq k$ so we get the following

Proof of Theorem 2

We say that a PATH has a RANK k when

$$(q_i,\ w) \vdash_{M,k}{}^{*} (q_j, w')$$

I.e. when M can pass ONLY through states numbered $m \leq k$ while going from q_i to q_j

RANK 0 **case** $k = 0$

$$R(i,\ j,\ 0) = \{w \in \Sigma^*;\quad (q_i,\ w) \vdash_{M,0}{}^{*} (q_j, w')\}$$

This means; M ”goes” from q_i to q_j only through states numbered $m \leq 0$

There is **no** such states as $K = \{q_1,\ q_2,\ \dots\ \ q_n\}$

Proof of Theorem 2

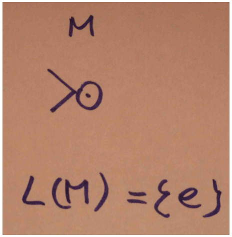
Hence $R(i,\ j,\ 0)$ means that M "goes" from q_i to q_j DIRECTLY, i.e. that

$$R(i,\ j,\ 0) = \{w \in \Sigma^*;\quad (q_i,\ w) \vdash_M{}^* (q_j, w')\}$$

Reminder: we use the BOOK definition so

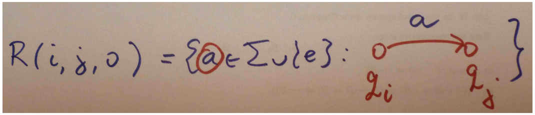
$$R(i,\ j,\ 0) = \begin{cases} a \in \Sigma \cup \{e\} & \text{if } i \neq j \text{ and } (q_i, a, q_j) \in \Delta \\ \{e\} \cup a \in \Sigma \cup \{e\} & \text{if } i = j \text{ and } (q_i, a, q_j) \in \Delta \end{cases}$$

Observe that we need $\{e\}$ in the second equation to include the following special case



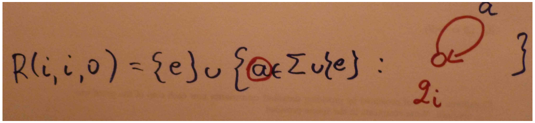
Proof of Theorem 2

We read $R(i, j, 0)$ from the **diagram** of M as follows



$$R(i,j,0) = \left\{ a \in \Sigma \cup \{e\} : \begin{tikzcd} q_i \arrow[r, "a"] & q_j \end{tikzcd} \right\}$$

and



$$R(i,i,0) = \{e\} \cup \left\{ a \in \Sigma \cup \{e\} : \begin{tikzcd} q_i \arrow[loop above, "a"] \end{tikzcd} \right\}$$

Proof of Theorem 2

RANK n **case** $k = n$

$$R(i,\ j,\ n) = \{w \in \Sigma^*;\quad (q_{\ i},\ w) \vdash_{M,n}{}^{*} (q_{\ j}, w')\}$$

This means; M "goes" from q_i to q_j through states numbered $m \leq n$

It means that M "goes" all states as $|K| = n$

It means that M will read any $w \in \Sigma$ and hence

$$R(i,\ j,\ n) = \{w \in \Sigma^*;\quad (q_i,\ w) \vdash_{M}{}^{*} (q_j, e)\}$$

Observe that

$$w \in L(M) \quad \text{iff} \quad w \in R(1,\ j,\ n) \quad \text{and} \quad q_j \in F$$

Proof of Theorem 2

By definition of the $L(M)$ we get

$$L(M) = \bigcup \{R(1, j, n) : \quad q_j \in F\}$$

Fact

All sets $R(i, j, k)$ are regular and hence L(M) is also regular

Proof by induction on k

Base case: k =0

All sets R(i, j, 0) are FINITE, hence are regular

Proof of Theorem 2

Inductive Step

The **recursive formula** for $R(i,\ j,\ k)$ is

$$R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1)R(k, k, k-1)^{*}R(k, j, k-1)$$

where n is the number of states of M and
$k = 0, \ldots, n, \quad i, j = 1, \ldots, n$

By Inductive assumption, all sets
$R(i, j, k-1),\ R(i, k, k-1),\ R(k, k, k-1),\ R(k, j, k-1)$ are regular and by the **Closure Theorem** so is the set $R(i, j, k)$

This **ends** the proof of **Theorem 2**

Observe that the recursive formula for $R(i,\ j,\ k)$ computes r such that L(M) = r

Example

Example

For the automaton M such that

$$M = (\{q_1, q_2, q_3\},\ \{a, b\},\ s = q_1,$$

$$\Delta = \{(q_1, b, q_2), (q_1, a, q_3), (q_2, a, q_1), (q_2, b, q_1),$$

$$(q_3, a, q_1), (q_3, b, q_1)\},\ F = \{q_1\})$$

Evaluate 4 steps, in which you must include at least one $R(i, j, 0)$, in the construction of regular expression that defines $L(M)$

Example

Reminder

$$L(M) = \bigcup \{R(1,\ j,\ n) : \quad q_j \in F\}$$

$$R(i,j,k) = R(i,j,k-1) \cup R(i,k,k-1)R(k,k,k-1)^* R(k,j,k-1)$$

$$R(i,\ j,\ 0) = \begin{cases} a \in \Sigma \cup \{e\} & \text{if } i \neq j \text{ and } (q_i, a, q_j) \in \Delta \\ \{e\} \cup a \in \Sigma \cup \{e\} & \text{if } i = j \text{ and } (q_i, a, q_j) \in \Delta \end{cases}$$

Example Solution

Solution

Step 1 $L(M) = R(1,1,3)$

Step 2

$R(1,1,3) = R(1,1,2) \cup R(1,3,2)R(3,3,2)^{*}R(3,1,2)$

Step 3

$R(1,1,2) = R(1,1,1) \cup R(1,2,1)R(2,2,1)^{*}R(2,1,1)$

Step 4

$R(1,1,1) = R(1,1,0) \cup R(1,1,0)R(1,1,0)^{*}R(1,1,0)$ and

$R(1,1,0) = \{e\} \cup \emptyset = \{e\}$, so we get

$R(1,1,1) = \{e\} \cup \{e\}\{e\}^{*}\{e\} = \{e\}$

Generalized Automata

Generalized Automaton

Definition
We define now a **Generalized Automaton** GM as the following generalization of of a nondeterministic automaton $M = (K, \Sigma, \Delta, s, F)$ as follows

$$GM = (K_G, \Sigma_G, \Delta_G, s_G, F_G)$$

1. GM has a single final state, i,e. $F_G = \{f\}$
2. $\Sigma_G = \Sigma \cup \mathcal{R}_0$ where $\mathcal{R}_0$ is a FINITE subset of the set $\mathcal{R}$ of **regular expressions** over Σ
3. Transitions of GM may be labeled not only by symbols in $\Sigma \cup \{e\}$ but also by **regular expressions** $r \in \mathcal{R}$, i.e. Δ_G is a FINITE set such that

$$\Delta_G \subseteq K \times (\Sigma \cup \{e\} \cup \mathcal{R}) \times K$$

4. There is no transition going into the initial state s nor out of the final state f
if $(q, u, p) \in \Delta_G$, then $q \neq f, \ p \neq s$

Generalized Automata

Given a nondeterministic automaton

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

We present now a new method of construction of a regular expression $r \in \mathcal{R}$ that defines $L(M)$, i.e. such that $L(M) = r$ by the use of the notion of of **Generalized Automaton**

The method consists of a construction of a sequence of generalized automata that are all equivalent to M

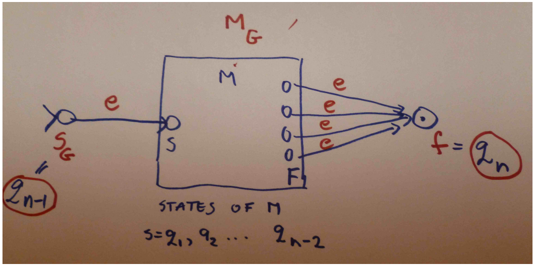
Construction

Steps of construction are as follows

Step 1

We **extend** M to a generalized automaton M_G, such that $L(M) = L(M_G)$ as depicted on the diagram below

Diagram of M_G



M_G Definition

Definition of M_G

We re-name states of M as $s = q_1, q_2, \dots, q_{n-2}$ for appropriate n and make the initial state $s = q_1$ and all final states of M the internal non-final states of G_M

We ADD TWO states: initial and one final, which me name q_{n-1}, q_n, respectively, i.e. we put

$$s_G = q_{n-1} \quad \text{and} \quad f = q_n$$

We take

$$\Delta_G = \Delta \cup \{(q_{n-1}, e, s)\} \cup \{(q, e, q_n) : \quad q \in F\}$$

Obviously $L(M) = L(M_G)$, and so $M \approx M_G$

States of G_M Elimination

We construct now a sequence $GM1, GM2, \dots, GM(n-2)$ such that

$$M \approx M_G \approx GM1 \approx \cdots \approx GM(n-2)$$

where $GM(n-2)$ has only **two states** q_{n-1} and q_n and only **one transition** (q_{n-1}, r, q_n) for $r \in \mathcal{R}$, such that

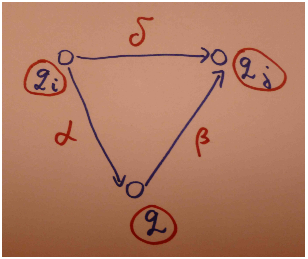
$$L(M) = r$$

We construct the sequence $GM1, GM2, \dots, GM(n-2)$ by eliminating states of M one by one following rules given by the following diagrams

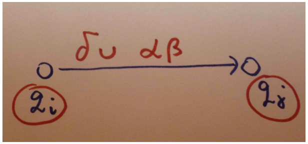
States of G_M Elimination

Case 1 of state **elimination**

Given a fragment of *GM* **diagram**



we **transform** it into

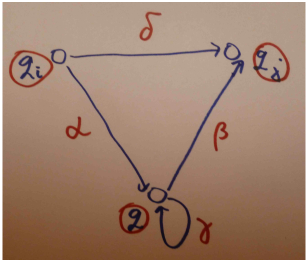


The state $q \in K$ has been **eliminated** preserving the language of *GM* and we constructed $GM' \approx GM$

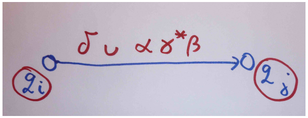
States of G_M Elimination

Case 2 of state **elimination**

Given a fragment of *GM* **diagram**



we **transform** it into



The state $q \in K$ has been **eliminated** preserving the language of *GM* and we constructed $GM' \approx GM$

Example 1

Example 1

Use the Generalized Automata Construction and States of G_M Elimination procedure to evaluate $r \in \mathcal{R}$, such that

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

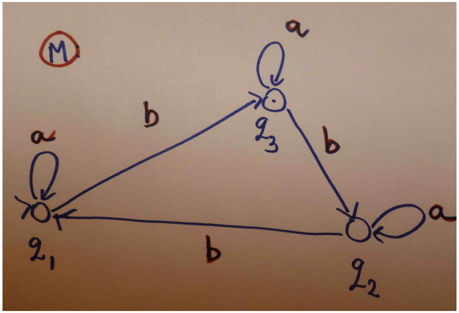
, where M is an automata that accepts the language

$$L = \{w \in \{a,b\}^* : \quad w \text{ has } \; 3k+1 \; b'\text{s}, \; \text{for some} \; k \in N\}$$

This is the Book example, page 80

Example 1

The **Diagram** of M is



Step 1

We extend M with $K = \{q_1, q_2, q_3\}$ to a generalized M_G by adding two states

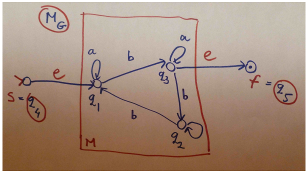
$$s_G = q_4 \quad \text{and} \quad f = q_5$$

We take

$$\Delta_G = \Delta \cup \{(q_4, e, q_1)\} \cup \{(q_3, e, q_5)\}$$

Example 1

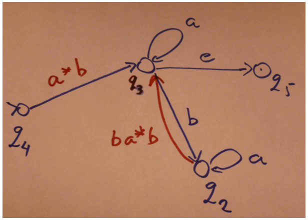
The **Diagram** of M_G is



Step 2

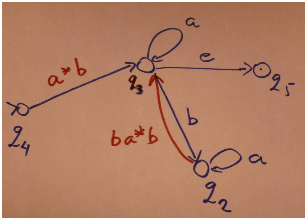
We construct $GM1 \approx M_G \approx M$ by **elimination** of q_1

The **Diagram** of $GM1$ is



Example 1

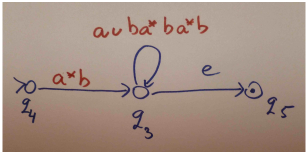
The **Diagram** of *GM1* is



Step 3

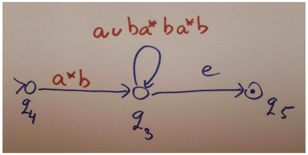
We construct $GM2 \approx GM1$ by **elimination** of q_2

The **Diagram** of *GM2* is



Example 1

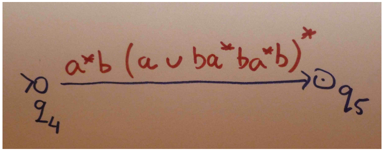
The **Diagram** of *GM2* is



Step 4

We construct $GM3 \approx GM2$ by **elimination** of q_3

The **Diagram** of *GM2* is



$$L(GM3) = a^*b(a \cup ba^*ba^*b)^* = L(M)$$

Example 2

Example 2

Given the automaton

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

where

$$K = \{q_1, q_2, q_3\}, \quad \Sigma = \{a, b\}, \quad s = q_1, \quad F = \{q_1\}$$

$$\Delta = \{(q_1, b, q_2),\ (q_1, a, q_3),\ (q_2, a, q_1),$$

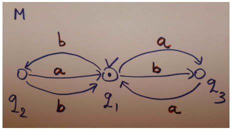
$$(q_2, b, q_1),\ (q_3, a, q_1),\ (q_3, b, q_1)$$

Use the Generalized Automata Construction and States of G_M Elimination procedure to evaluate $r \in \mathcal{R}$, such that

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

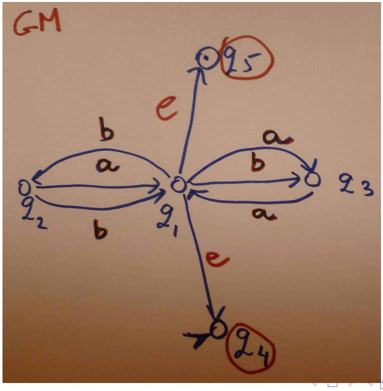
Example 2

The **diagram** of M is



Step 1

The **diagram** of $M_G \approx M$ is



Example 2

Step 1

The components of $M_G \approx M$ are

$$M_G = (K = \{q_1, q_2, q_3, q_4, q_5\}, \ \Sigma = \{a, b\}, \quad s_G = q_4,$$

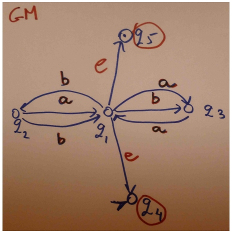
$$\Delta_G = \{(q_1, b, q_2), \ (q_1, a, q_3), \ (q_2, a, q_1),$$

$$(q_2, b, q_1), (q_3, a, q_1), (q_3, b, q_1), \ (q_4, e, q_1),$$

$$(q_1, e, q_5)\}, \qquad F = \{q_5\})$$

Example 2

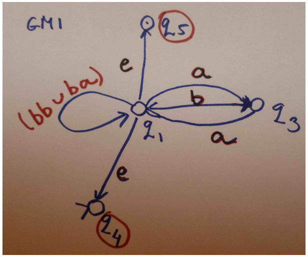
The **Diagram** of M_G is



Step 2

We construct $GM1 \approx M_G \approx M$ by **elimination** of q_2

The **Diagram** of $GM1$ is



Example 2

Step 2

The components of $GM1 \approx M_G \approx M$ are

$$GM1 = (K = \{q_1, q_3, q_4, q_5\}, \quad \Sigma = \{a, b\}, \quad s_G = q_4$$

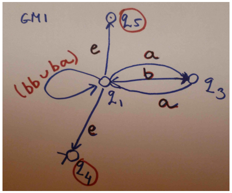
$$\Delta_G = \{(q_1, a, q_3), \ (q_1, (bb \cup ba), q_1),$$

$$(q_3, a, q_1), \ (q_3, b, q_1), \ (q_4, e, q_1),$$

$$(q_1, e, q_5)\}, \quad F = \{q_5\})$$

Example 2

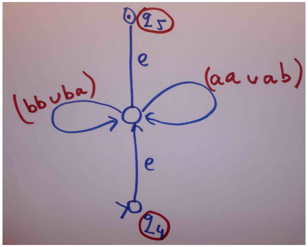
The **Diagram** of *GM1* is



Step 3

We construct $GM2 \approx GM1$ by **elimination** of q_3

The **Diagram** of *GM2* is



Example 2

Step 3

The components of $GM2 \approx GM1 \approx M_G \approx M$ are

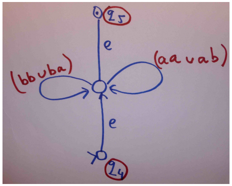
$$GM2 = (K = \{q_1,\ q_4,\ q_5\},\quad \Sigma = \{a, b\},\quad s_G = q_4$$

$$\Delta_G = \{(q_1, (bb \cup ba), q_1),\ \ (q_1, (aa \cup ab), q_1),$$

$$(q_4, e, q_1), (q_1, e, q_5)\},\quad F = \{q_5\})$$

Example 2

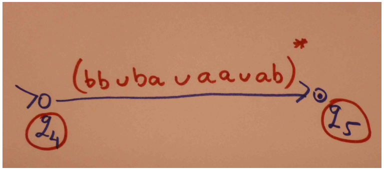
The **Diagram** of *GM2* is



Step 4

We construct $GM3 \approx GM2$ by **elimination** of q_1

The **Diagram** of *GM3* is

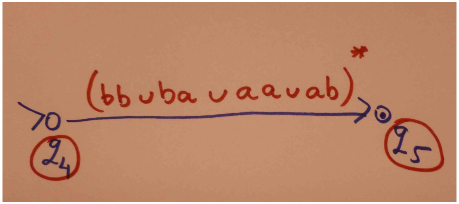


Example 2

We have constructed

$$GM3 \approx GM2 \approx GM1 \approx M_G \approx M$$

The **Diagram** of $GM3$ is



Hence the language

$$L(GM3) = (bb \cup ba \cup aa \cup ab)^* = ((a \cup b)(a \cup b))^* = L(M)$$

Chapter 2
Finite Automata

Slides Set 2

PART 4: Languages that are Not Regular

Languages that are Not Regular

We know that there are **uncountably** many and exactly $\mathcal{C}$ 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 $\mathcal{C}$ 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 $L_4 = \{a^n : \ n \geq 0\}$ is **regular** because in fact $L_3 = L_4$

E5 The language $L_4 = \{a^n : \ n \in Prime\}$ 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, as the class of regular languages is closed under intersection

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

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

Regular aor 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 languages accepted by Finite Automata FA is **closed** under $\cup, \cap, -, \circ, ^*$

Regular or not Regular Languages

E13 The language

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

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

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

We used the Lecture Definition of M

Regular and NOT Regular Languages

Proof of **E13** pictures

Diagram of M

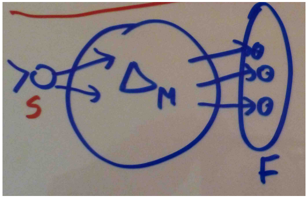
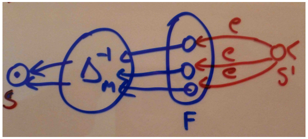


Diagram of M^R



Regular and NOT Regular Languages

E14

Any finite language is regular

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

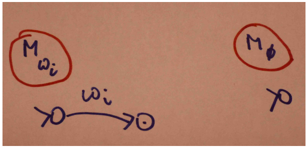
$$L = \emptyset \text{ or } L = \{w_1, w_2, \ldots w_n\} \text{ for } n > 0\}$$

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

where



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

Review Questions

Review Questions

Write SHORT answers

Q1

For any language $L \subseteq \Sigma^*, \ \Sigma \neq \emptyset$ there is a deterministic automata M, such that $L = L(M)$

Q2

Any regular language has a finite representation.

Q3

Any finite language is regular

Q4

Given L_1, L_2 languages over Σ, then $((L_1 \cap (\Sigma^* - L_2)) \cup L_2)L_1$ is a regular regular language

Review Questions

SHORT answers

Q1

For any language $L \subseteq \Sigma^*, \ \Sigma \neq \emptyset$ there is a deterministic automata M, such that $L = L(M)$

True only when L is regular

Q2

Any regular language has a finite representation.

True by definition of regular language and the fact that regular expression is a finite string

Q3

Any finite language is regular

True as we proved it

Q4

Given L_1, L_2 languages over Σ, then

$((L_1 \cap (\Sigma^* - L_2)) \cup L_2)L_1$ is a regular regular language

True only when both are regular languages

Review Questions for Quiz

Write SHORT answers

Q5

For any finite automata M

$$L(M) = \bigcup \{R(1, j, n) : \quad q_j \in F\}$$

Q6

Σ in any Generalized Finite Automaton includes some regular expressions

Q7

Pumping Lemma says that we can always prove that a language is not regular

Q8

$L = \{a^n c^n : n \geq 0\}$ is regular

Review Questions

SHORT answers

Q5

For any finite automata M

$$L(M) = \bigcup \{R(1, j, n) : \quad q_j \in F\}$$

True only when M has n states and they are put in 1-1 sequence and $q_1 = s$

Q6

Σ in any Generalized Finite Automaton includes some regular expressions

True by definition

Review Questions

Q7

Pumping Lemma says that we can always prove that a language is not regular

Not True PL serves as a **tool** for proving that some languages are not regular

Q8

$L = \{a^n c^n : n \geq 0\}$ is regular

Not True we proved by PL that it is not regular

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

Pumping Lemma

We present here **two versions** of the Pumping Lemma

First is the Lecture Notes version adopted from the first edition of the Book

The second is the Book version (page 88) from the second edition

The Book version is a slight **generalization** of the Lecture version

Pumping Lemma

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

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

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

Problem

Given an infinite language L we want to **prove it** to be nor 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 $xy^nz \in L$ for all $n \geq 0$

3. We examine the fact $xy^nz \in L$ 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

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

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

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 $lw| = \mathbf{k} > \mathbf{n}$ and

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

Proof of Pumping Lemma

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 σ_j

and M would still accept such string

Proof of Pumping Lemma

We just showed by **Pigeon Hole Principle** that automaton 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

$\mathbf{x} = \sigma_{\mathbf{1}}\sigma_{\mathbf{2}}\ldots\sigma_{\mathbf{i}}$, $\mathbf{y} = \sigma_{\mathbf{i+1}}\ldots\sigma_{\mathbf{j}}$, $\mathbf{z} = \sigma_{\mathbf{j+1}}\ldots\sigma_{\mathbf{k}}$ for $y \neq e$

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

The computation of M that accepts $xy^n z$ is as follows

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

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

This **ends** the proof

Observe that the proof of the holds for 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

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

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 in infinite

The rest of the proof exactly the same as in the previous case of the 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

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

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

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

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

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

Notice that from the proof of Pumping Lemma 1

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

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

Book Pumping Lemma

We write the Book Pumping Lemma symbolically using quantifiers symbols as follows

Book Pumping Lemma

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

Then the following holds

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

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

Book Pumping Lemma

A natural question arises:

WHY the Book Pumping Lemma applies also when L is a **finite** regular language?

We know that when L is a **finite** regular language the Lecture Pumping Lemma does not apply

Book Pumping Lemma

Let's look at an example of a finite, and hence a regular language

$$L = \{a, b, ab, bb\}$$

Observe that the condition

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

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

of the Book Pumping Lemma **holds** because there exists $n = 3$ such that the conditions becomes as follows

Book Pumping Lemma

Take $n = 3$, or any $n \geq 3$ we get statement:

$$\exists_{n=3} \forall_{w \in L} \; (\; |w| \geq 3 \;\; \Rightarrow$$

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

Observe that the above is a TRUE statement because the statement $|w| \geq 3$ is FALSE for all $w \in L = \{a, b, ab, bb\}$

By definition, the implication $FALSE \Rightarrow (anything)$ is always TRUE, hence the whole statement is TRUE

Book Pumping Lemma

The same reasoning applies for any **finite** (and hence regular) language

In general, let L be any **finite** language

Let $m = max\{|w| : w \in L\}$

Such m **exists** because L is finite

Take $n = m + 1$ as the n in the condition of the Book Pumping Lemma

The Lemma condition is TRUE for **all** $w \in L$, because the statement

$|w| \geq m + 1$ is FALSE for **all** $w \in L$

By definition, the implication $FALSE \Rightarrow (anything)$ is always TRUE, hence the whole statement is TRUE

Pumping Lemma Applications

Pumping Lemma Applications

We ese now 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 can use the Lecture version, i.e. the following

Pumping Lemma Applications

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

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

Pumping Lemma Applications

Reminder: 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 $w = xyz$ 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, 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 $n \geq 0$

There are **three** possibilities for $y \neq e$

We will show that in **each case** we prove that $xy^n z \in L$ is impossible, i.e. we get a contradiction

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

$$a^{p+nq+r} b^s \ \in L \quad \text{iff} \quad [p + nq + r = s$$

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

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 $xy^nz \in L$ for all $n \geq 0$ becomes as follows

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

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

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

By definition of L

$$a^pb^{r+nq+r} \in L \quad \text{iff} \quad [p = r + qn + s$$

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 i 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 $m \in Prime$

Pumping Lemma Applications

Then

$$x = a^p, \quad y = a^q, \quad z = a^r \text{ for } \quad p \geq 0, \quad q > 0, \quad r \geq 0$$

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

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

It means that for all n, p, q, r the following condition hold

C $n \geq 0, \ p \geq 0, \quad q > 0, \quad r \geq 0,$ and $p + nq + r \in Prime$

But this is IMPOSSIBLE

Pumping Lemma Applications

Take n = p + 2q + r + 2 and **evaluate:**

$$p + nq + r = p + (p + 2q + r + 2)q + r =$$

$$p(1 + q) + 2q(q + 1) + r(q + 1) = (q + 1)(p + 2q + r)$$

By the above and the condition **C** we get that

$$p + nq + r \in Prime \quad \text{and} \quad p + nq + r = (q + 1)(p + 2q + r)$$

and both factors are natural numbers greater than 1 what is a CONTRADICTION

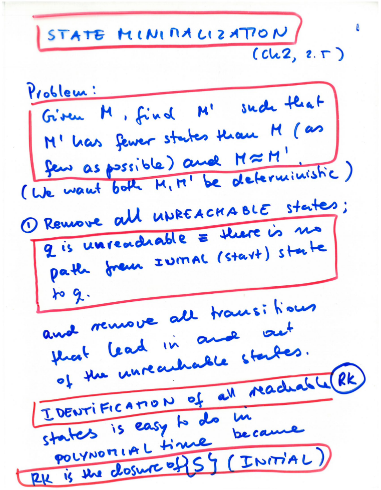
This **ends the proof**

Chapter 2
Finite Automata

Slides Set 3

PART 5: State Minimization

State Minimalization



STATE MINIMALIZATION

(Ch2, 2.5)

Problem:

Given M, find M' such that M' has fewer states than M (as few as possible) and $M \approx M'$.

(We want both M, M' be deterministic)

① Remove all UNREACHABLE states;

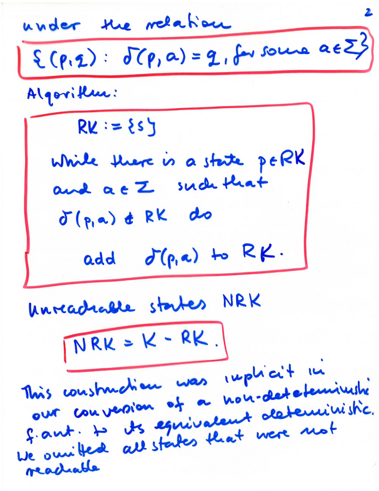
q is unreachable ≡ there is no path from INITIAL (start) state to q.

and remove all transitions that lead in and out of the unreachable states.

IDENTIFICATION of all reachable (RK) states is easy to do in POLYNOMIAL time because

RK is the closure of $\{S\}$ (INITIAL)

State Minimalization



2

under the relation

$$\boxed{\{(p,q) : \delta(p,a) = q, \text{ for some } a \in \Sigma\}}$$

Algorithm:

```
RK := {s}
while there is a state $p \in RK$
and $a \in \Sigma$ such that
$\delta(p,a) \notin RK$ do
    add $\delta(p,a)$ to RK.
```

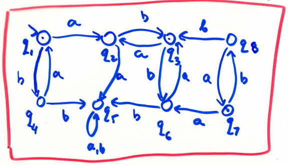
Unreachable states NRK

$$\boxed{NRK = K - RK.}$$

This construction was implicit in our conversion of a non-deterministic f.aut. to its equivalent deterministic. We omitted all states that were not reachable

State Minimalization

EXAMPLE M given by a diagram



$$\begin{tikzcd}
q_1 \arrow[r, "a"] \arrow[d, bend right, "b"'] & q_2 \arrow[r, bend left, "b"] \arrow[d, "a"] & q_3 \arrow[l, bend left, "a"] \arrow[d, bend right, "b"'] & q_8 \arrow[l, "b"'] \arrow[d, bend right, "a"'] \\
q_4 \arrow[u, bend right, "a"'] \arrow[r, "b"] & q_5 \arrow[loop below, "a,b"] & q_6 \arrow[l, "b"'] \arrow[u, bend right, "a"'] & q_7 \arrow[l, "a"] \arrow[u, bend right, "b"']
\end{tikzcd}$$

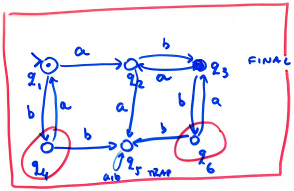
$L = L(M) = (ab \cup ba)^*$

q_7 is UNREACHABLE state

We can cross out q_7 and all transitions in and out of q_7.

We get the following state diagram:

State Minimalization

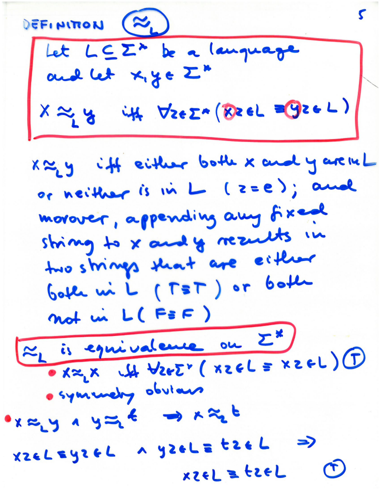


Look at states q_4 and q_6

If we are in either state, precisely the same strings lead the automaton to acceptance!

We will call such states equivalent and we will "merge" them into one one state.

State Minimalization



DEFINITION $\textcircled{\approx_L}$ 5

Let $L \subseteq \Sigma^*$ be a language and let $x, y \in \Sigma^*$

$x \approx_L y$ iff $\forall z \in \Sigma^* (xz \in L \equiv yz \in L)$

$x \approx_L y$ iff either both x and y are in L or neither is in L ($z = e$); and morover, appending any fixed string to x and y results in two strings that are either both in L ($T \equiv T$) or both not in L ($F \equiv F$)

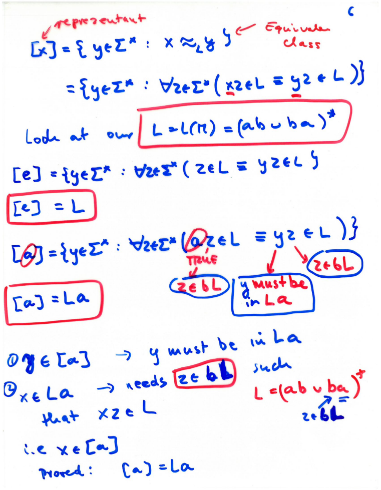
$\approx_L$ is equivalence on Σ^*

- $x \approx_L x$ iff $\forall z \in \Sigma^* (xz \in L \equiv xz \in L)$ $\textcircled{T}$
- symmetry obvious
- $x \approx_L y \wedge y \approx_L t \Rightarrow x \approx_L t$

$xz \in L \equiv yz \in L \wedge yz \in L \equiv tz \in L \Rightarrow$

$xz \in L \equiv tz \in L$ $\textcircled{T}$

State Minimalization



representant

Equivalence class

$[x] = \{ y \in \Sigma^* : x \approx_L y \}$

$= \{ y \in \Sigma^* : \forall z \in \Sigma^* (\underline{x}z \in L \equiv \underline{y}z \in L) \}$

Look at our $\boxed{L = L(M) = (ab \cup ba)^*}$

$[e] = \{ y \in \Sigma^* : \forall z \in \Sigma^* (z \in L \equiv yz \in L \}$

$\boxed{[e] = L}$

$[a] = \{ y \in \Sigma^* : \forall z \in \Sigma^* (az \in L \equiv yz \in L) \}$

TRUE

$\boxed{z \in bL}$ $\boxed{y \text{ must be in } La}$ $\boxed{z \in bL}$

$\boxed{[a] = La}$

① $y \in [a] \rightarrow y$ must be in La

② $x \in La \rightarrow$ needs $\boxed{z \in bL}$ such that $xz \in L$

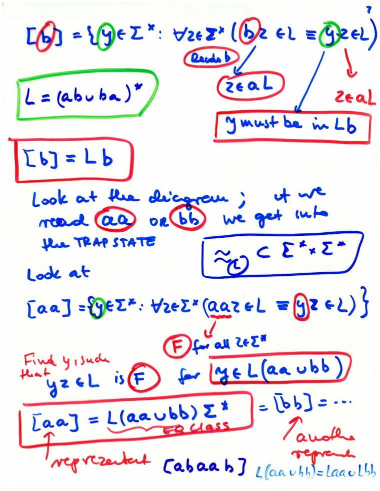
$L = (ab \cup ba)^*$

$z \in bL$

i.e $x \in [a]$

Proved: $[a] = La$

State Minimalization



$[b] = \{y \in \Sigma^* : \forall z \in \Sigma^* (bz \in L \equiv yz \in L)$

Reads b

$z \in aL$

$z \in aL$

y must be in Lb

$L = (ab \cup ba)^*$

$[b] = Lb$

Look at the diagram; if we read aa or bb we get into the TRAP STATE

$\approx_{(L)} \subset \Sigma^* \times \Sigma^*$

Look at

$[aa] = \{y \in \Sigma^* : \forall z \in \Sigma^* (aaz \in L \equiv yz \in L)\}$

F for all $z \in \Sigma^*$

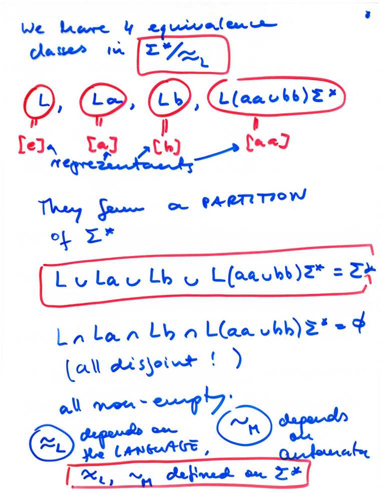
Find y, such that $yz \in L$ is F for $y \in L(aa \cup bb)$

$[aa] = L(aa \cup bb)\Sigma^*$ EQ class $= [bb] = \dots$

another represent

representant $[abaab]$ $L(aa \cup bb) = Laa \cup Lbb$

State Minimalization



We have 4 equivalence classes in $\boxed{\Sigma^*/\approx_L}$

L, La, Lb, $L(aa \cup bb)\Sigma^*$

$[e]$, $[a]$, $[b]$, $[aa]$

representants

They form a PARTITION of Σ^*

$$\boxed{L \cup La \cup Lb \cup L(aa \cup bb)\Sigma^* = \Sigma^*}$$

$$L \cap La \cap Lb \cap L(aa \cup bb)\Sigma^* = \emptyset$$

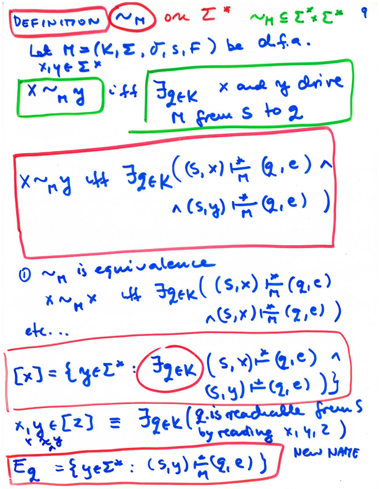
(all disjoint !)

all non-empty.

$\approx_L$ depends on the LANGUAGE, $\sim_M$ depends on automata

$\boxed{\approx_L, \sim_M \text{ defined on } \Sigma^*}$

State Minimalization



$\boxed{\text{DEFINITION}}$ $\sim_M$ on Σ^* $\sim_M \subseteq \Sigma^* \times \Sigma^*$ 9

Let $M = (K, \Sigma, \delta, s, F)$ be d.f.a.

$x, y \in \Sigma^*$

$\boxed{x \sim_M y}$ iff $\boxed{\exists q \in K \text{ x and y drive M from s to q}}$

$$\boxed{x \sim_M y \text{ iff } \exists q \in K \left((s, x) \vdash_M^* (q, e) \wedge (s, y) \vdash_M^* (q, e) \right)}$$

① $\sim_M$ is equivalence

$x \sim_M x$ iff $\exists q \in K \left((s, x) \vdash_M^* (q, e) \wedge (s, x) \vdash_M^* (q, e) \right)$

etc...

$$\boxed{[x] = \{ y \in \Sigma^* : \exists q \in K \left((s, x) \vdash^* (q, e) \wedge (s, y) \vdash^* (q, e) \right) \}}$$

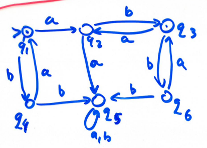
$x, y \in [z] \equiv \exists q \in K$ (q is reachable from s by reading x, y, z)

$x \sim y$

$$\boxed{E_q = \{ y \in \Sigma^* : (s, y) \vdash_M^* (q, e) \}}$$ NEW NAME

State Minimalization

BACK TO OUR EXAMPLE 10



$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[r, "a"] \arrow[d, bend right, "b"'] & q_2 \arrow[r, bend left, "b"] \arrow[d, "a"] & q_3 \arrow[l, bend left, "a"] \arrow[d, bend right, "b"'] \\
& q_4 \arrow[u, bend right, "a"'] \arrow[r, "b"] & q_5 \arrow[loop below, "{a,b}"] & q_6 \arrow[l, "b"'] \arrow[u, bend right, "a"']
\end{tikzcd}$$

M

$\sim_M$

$L(M) = (ab \cup ba)^*$

$E_{q_1} = (ba)^*$ $= [ba] = [baba] = [e] \ldots$

Six equivalence classes!

$E_{q_2} = a\cancel{b}La$

Form a PARTITION

$E_{q_3} = abL$

$a \in E_{q_2}$

$E_{q_4} = b(ab)^*$

[illegible]$_M$

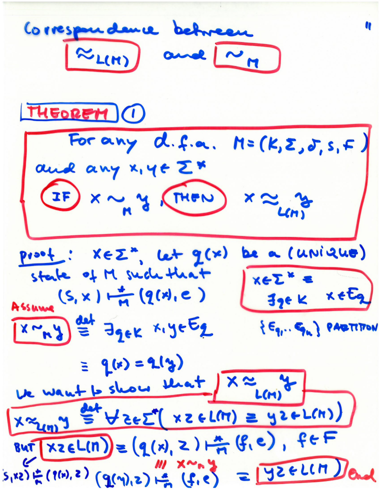
$E_{q_5} = L(bb \cup aa)\Sigma^*$

$$\boxed{E_q = \{y \in \Sigma^* : (s, y) \vdash_M^* (q, e)\}}$$

$E_{q_6} = abLb$

$\{E_{q_1}, E_{q_2} \ldots E_{q_6}\}$ PARTITION of Σ^*

State Minimalization



Correspondence between $\boxed{\approx_{L(M)}}$ and $\boxed{\sim_M}$

"

$\boxed{\text{THEOREM}}$ 1

$$\boxed{\begin{array}{l}\text{For any d.f.a. } M=(K,\Sigma,\delta,s,F) \\ \text{and any } x,y\in\Sigma^* \\ \text{IF } x\sim_M y, \text{ THEN } x\approx_{L(M)} y\end{array}}$$

<u>proof</u>: $x\in\Sigma^*$, let $q(x)$ be a (UNIQUE) state of M such that

$(s,x)\vdash_M^* (q(x),e)$

$\boxed{x\in\Sigma^* \equiv \exists q\in K\ x\in E_q}$

$\{E_{q_1},\ldots E_{q_n}\}$ PARTITION

Assume

$\boxed{x\sim_M y} \overset{\text{def}}{\equiv} \exists q\in K\ x,y\in E_q$

$\equiv q(x)=q(y)$

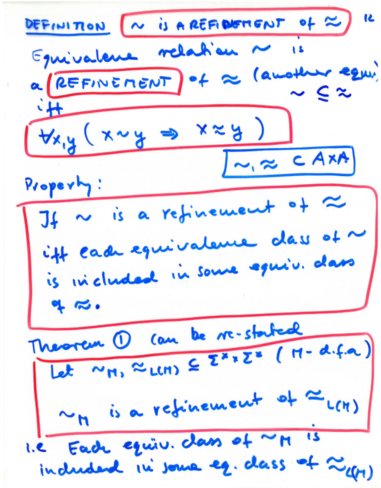
We want to show that $\boxed{x\approx_{L(M)} y}$

$\boxed{x\approx_{L(M)} y \overset{\text{def}}{\equiv} \forall z\in\Sigma^*(xz\in L(M)\equiv yz\in L(M))}$

BUT $\boxed{xz\in L(M)} \equiv (q(x),z)\vdash_M^*(f,e)$, $f\in F$

$s,xz)\vdash_M^*(q(x),z)$ $(q(y),z)\vdash_M^*(f,e)$ $\overset{x\sim_M y}{\equiv}$ $\boxed{yz\in L(M)}$ End

State Minimalization



DEFINITION $\boxed{\sim \text{ is A REFINEMENT of } \approx}$ 12

Equivalence relation $\sim$ is a $\boxed{\text{REFINEMENT}}$ of $\approx$ (another equiv)

$\sim \subseteq \approx$

iff

$\boxed{\forall_{x,y} (x \sim y \Rightarrow x \approx y)}$

$\boxed{\sim, \approx \subset A \times A}$

Property:

If $\sim$ is a refinement of $\approx$ iff each equivalence class of $\sim$ is included in some equiv. class of $\approx$.

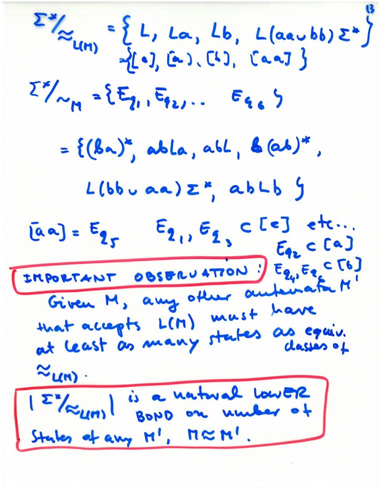
Theorem ① can be re-stated

Let $\sim_M, \approx_{L(M)} \subseteq \Sigma^* \times \Sigma^*$ (M - d.f.a)

$\sim_M$ is a refinement of $\approx_{L(M)}$

i.e Each equiv. class of $\sim_M$ is included in some eq. class of $\approx_{L(M)}$

State Minimalization



13

$$\Sigma^*/_{\approx_{L(M)}} = \{ L, La, Lb, L(aa \cup bb)\Sigma^* \}$$
$$= \{ [e], [a], [b], [aa] \}$$

$$\Sigma^*/_{\sim_M} = \{ E_{q_1}, E_{q_2}, \ldots \; E_{q_6} \}$$

$$= \{ (ba)^*, abLa, abL, b(ab)^*, L(bb \cup aa)\Sigma^*, abLb \}$$

$[aa] = E_{q_5}$ $\quad E_{q_1}, E_{q_3} \subset [e]$ etc...

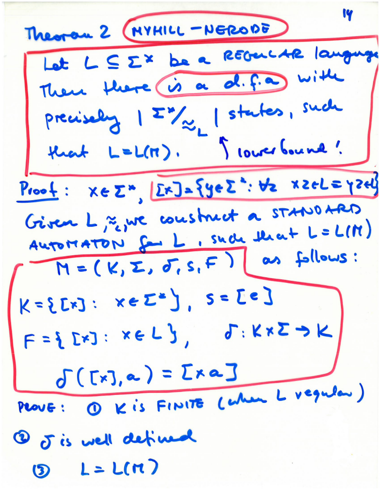
$E_{q_2} \subset [a]$

$\boxed{\text{IMPORTANT OBSERVATION:}}$ $E_{q_4}, E_{q_6} \subset [b]$

Given M, any other automata M' that accepts $L(M)$ must have at least as many states as equiv. classes of $\approx_{L(M)}$.

$$\boxed{\left|\Sigma^*/_{\approx_{L(M)}}\right| \text{ is a natural LOWER BOND on number of states of any } M',\ M \approx M'.}$$

State Minimalization



14

Theorem 2 MYHILL – NERODE

Let $L \subseteq \Sigma^*$ be a REGULAR language. Then there is a d.f.a with precisely $|\Sigma^*/\approx_L|$ states, such that $L = L(M)$. ↑ lower bound!

Proof: $x \in \Sigma^*$, $[x] = \{y \in \Sigma^* : \forall z \; xz \in L \equiv yz \in L\}$

Given L, $\approx_L$, we construct a STANDARD AUTOMATON for L, such that $L = L(M)$ as follows:

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

$K = \{[x] : x \in \Sigma^*\}$, $s = [e]$

$F = \{[x] : x \in L\}$, $\delta : K \times \Sigma \to K$

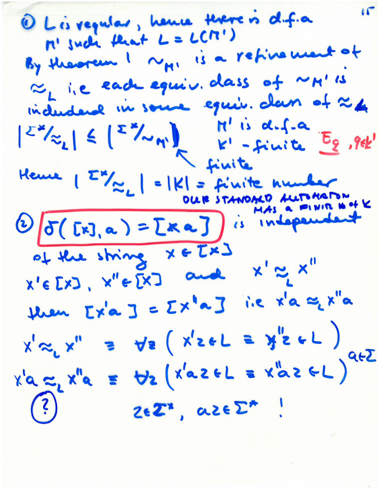
$\delta([x], a) = [xa]$

PROVE: ① K is FINITE (when L regular)

② δ is well defined

③ $L = L(M)$

State Minimalization



15

① L is regular, hence there is d.f.a M' such that $L = L(M')$

By theorem 1 $\sim_{M'}$ is a refinement of $\approx_L$ i.e each equiv. class of $\sim_{M'}$ is included in some equiv. class of $\approx_L$

$|\Sigma^*/\approx_L| \leq |\Sigma^*/\sim_{M'}|$ ← finite

M' is d.f.a

K' - finite E_q, $q \in K'$

Hence $|\Sigma^*/\approx_L| = |K| =$ finite number

OUR STANDARD AUTOMATON HAS A FINITE # of K

② $\boxed{\delta([x], a) = [xa]}$ is independent of the string $x \in [x]$

$x' \in [x]$, $x'' \in [x]$ and $x' \approx_L x''$

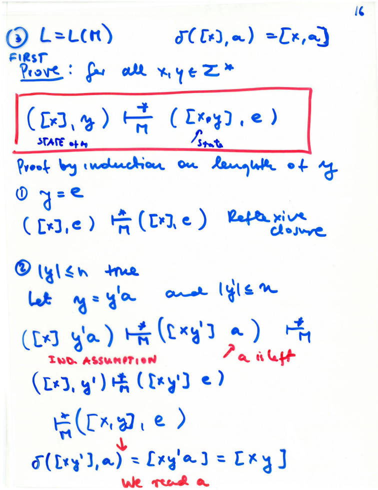
then $[x'a] = [x''a]$ i.e $x'a \approx_L x''a$

$$x' \approx_L x'' \equiv \forall z \, (x'z \in L \equiv x''z \in L)$$

$$x'a \approx_L x''a \equiv \forall z \, (x'az \in L \equiv x''az \in L)^{a \in \Sigma}$$

$\textcircled{?}$ $\quad z \in \Sigma^*$, $az \in \Sigma^*$!

State Minimalization



16

③ $L = L(M)$ $\delta([x], a) = [x, a]$

FIRST

Prove: for all $x, y \in \Sigma^*$

$$\boxed{(\underset{\text{STATE of M}}{[x]}, y) \vdash^*_M (\underset{\text{State}}{[x \cdot y]}, e)}$$

Proof by induction on length of y

① $y = e$

$([x], e) \vdash^*_M ([x], e)$ Reflexive closure

② $|y| \leq n$ true

Let $y = y'a$ and $|y'| \leq n$

$([x]\ y'a) \vdash^*_M ([xy']\ a) \vdash^*_M$

IND. ASSUMPTION a is left

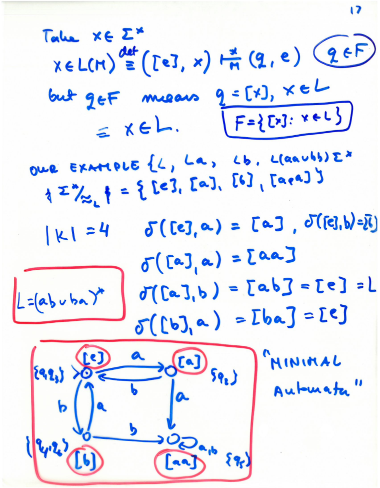
$([x], y') \vdash^*_M ([xy']\ e)$

$\vdash^*_M ([x, y], e)$

$\delta([xy'], a) = [xy'a] = [xy]$

we read a

State Minimalization



17

Take $x \in \Sigma^*$

$x \in L(M) \overset{def}{\equiv} ([e], x) \vdash_M^* (q, e)$ $\boxed{q \in F}$

but $q \in F$ means $q = [x]$, $x \in L$

$\equiv x \in L.$ $\boxed{F = \{[x] : x \in L\}}$

our EXAMPLE $\{L, La, Lb, L(aa \cup bb)\Sigma^*\}$

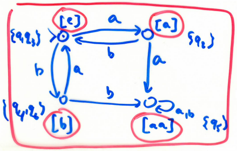
$|\Sigma^*/_{\approx_L}| = \{[e], [a], [b], [aa]\}$

$|K| = 4$ $\delta([e], a) = [a]$, $\delta([e], b) = [b]$

$\delta([a], a) = [aa]$

$\boxed{L = (ab \cup ba)^*}$ $\delta([a], b) = [ab] = [e] = L$

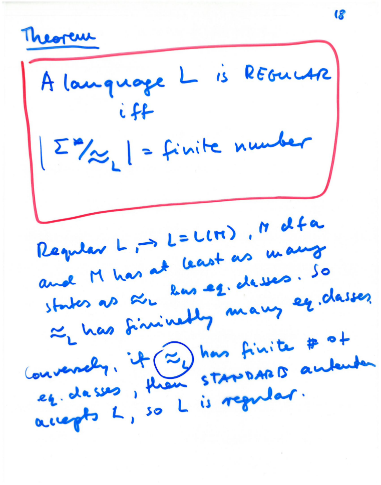
$\delta([b], a) = [ba] = [e]$



$$\begin{tikzcd}
\{q_1 q_3\} \arrow[r, phantom, ">"] & {[e]} \arrow[r, bend left, "a"] \arrow[d, bend right, "b"'] & {[a]} \arrow[l, bend left, "b"] \arrow[d, "a"] & \{q_2\} \\
\{q_4, q_6\} & {[b]} \arrow[u, bend right, "a"'] \arrow[r, "b"] & {[aa]} \arrow[loop right, "{a,b}"] & \{q_5\}
\end{tikzcd}$$

"MINIMAL Automata"

State Minimalization



18

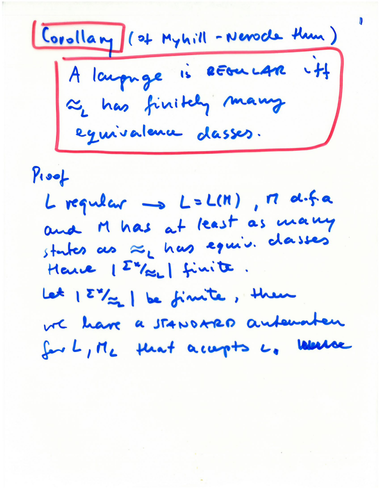
Theorem

A language L is REGULAR
iff
$|\Sigma^*/\approx_L|$ = finite number

Regular L, → $L = L(M)$, M dfa and M has at least as many states as $\approx_L$ has eq. classes. So $\approx_L$ has finitely many eq. classes

Conversely, if $\approx_L$ has finite # of eq. classes, then STANDARD automaton accepts L, so L is regular.

State Minimalization



$\boxed{\text{Corollary}}$ (of Myhill - Nerode thm)

$\boxed{\text{A language is REGULAR iff } \approx_L \text{ has finitely many equivalence classes.}}$

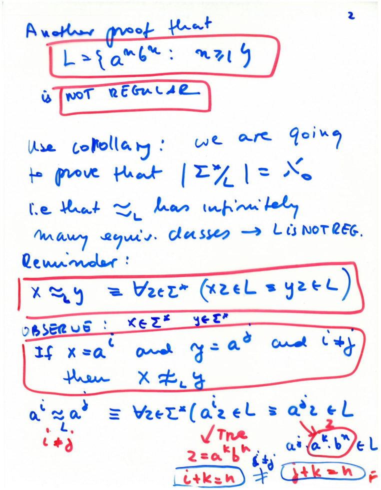
Proof

L regular $\rightarrow$ $L = L(M)$, M d.f.a and M has at least as many states as $\approx_L$ has equiv. classes

Hence $|\Sigma^*/\approx_L|$ finite.

Let $|\Sigma^*/\approx_L|$ be finite, then we have a STANDARD automaton for L, M_L that accepts L. ~~hence~~

State Minimalization



2

Another proof that $\boxed{L = \{a^n b^n : n \geq 1\}}$ is $\boxed{\text{NOT REGULAR}}$

Use corollary: we are going to prove that $|\Sigma^*/_L| = \aleph_0$ i.e that $\approx_L$ has infinitely many equiv. classes → L is NOT REG.

Reminder:

$$\boxed{x \approx_L y \equiv \forall z \in \Sigma^* (xz \in L \equiv yz \in L)}$$

OBSERVE: $x \in \Sigma^*$ $y \in \Sigma^*$

$$\boxed{\text{If } x = a^i \text{ and } y = a^j \text{ and } i \neq j \text{ then } x \not\approx_L y}$$

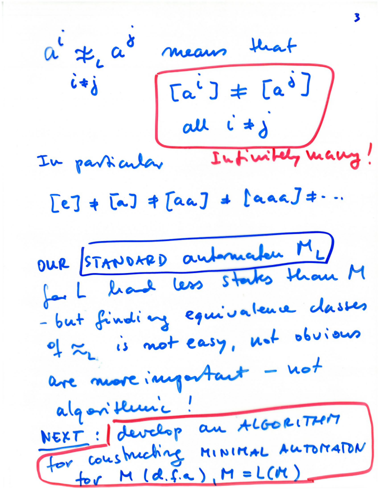
$$\underset{i \neq j}{a^i \approx_L a^j} \equiv \forall z \in \Sigma^* (a^i z \in L \equiv a^j z \in L$$

✓True

$z = a^k b^n$ $i \neq j$ $a^i \cdot \overset{z}{\boxed{a^k \cdot b^n}} \in L$

$\boxed{i + k = n} \neq \boxed{j + k = n}$ F

State Minimalization



3

$a^i \not\approx_L a^j$ means that
$i \neq j$

$\boxed{[a^i] \neq [a^j] \text{ all } i \neq j}$

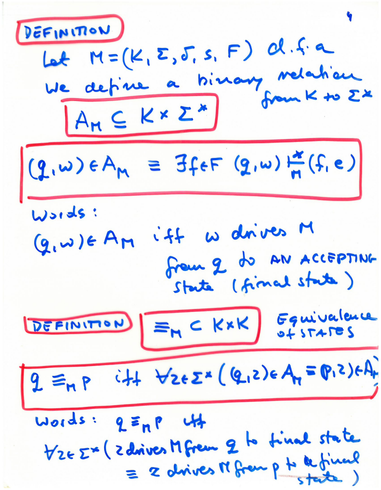
In particular   Infinitely many!

$[e] \neq [a] \neq [aa] \neq [aaa] \neq \dots$

OUR $\boxed{\text{STANDARD automaton } M_L}$
for L had less states than M
- but finding equivalence classes
of $\approx_L$ is not easy, not obvious
are more important - not
algorithmic!

NEXT: develop an ALGORITHM
for constructing MINIMAL AUTOMATON
for M (d.f.a), M = L(M)

State Minimalization



4

DEFINITION

Let $M=(K,\Sigma,\delta,s,F)$ d.f.a

We define a binary relation from K to Σ^*

$$\boxed{A_M \subseteq K\times\Sigma^*}$$

$$\boxed{(q,w)\in A_M \equiv \exists f\in F\ (q,w)\vdash_M^{*}(f,e)}$$

Words:

$(q,w)\in A_M$ iff w drives M from q to AN ACCEPTING state (final state)

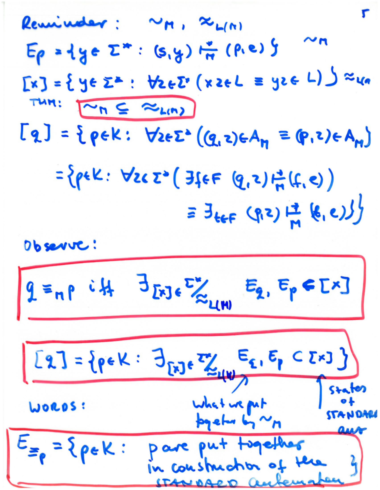
DEFINITION $\boxed{\equiv_M \subset K\times K}$ Equivalence of STATES

$$\boxed{q\equiv_M p \text{ iff } \forall z\in\Sigma^*\,((q,z)\in A_M \equiv (p,z)\in A_M)}$$

Words: $q\equiv_M p$ iff

$\forall z\in\Sigma^*$ (z drives M from q to final state $\equiv$ z drives M from p to a final state)

State Minimalization



5

Reminder: $\sim_M$, $\approx_{L(M)}$

$E_p = \{y \in \Sigma^* : (s,y) \vdash_M^* (p,e)\}$ $\sim_M$

$[x] = \{y \in \Sigma^* : \forall z \in \Sigma^* (xz \in L \equiv yz \in L)\}$ $\approx_{L(M)}$

THM: $\boxed{\sim_M \subseteq \approx_{L(M)}}$

$[q] = \{p \in K : \forall z \in \Sigma^* ((q,z) \in A_M \equiv (p,z) \in A_M\}$

$= \{p \in K : \forall z \in \Sigma^* (\exists f \in F \; (q,z) \vdash_M^* (f,e))$

$\equiv \exists_{t \in F} \; (p,z) \vdash_M^* (t,e))\}$

Observe:

$$\boxed{q \equiv_M p \text{ iff } \exists_{[x] \in \Sigma^*/\approx_{L(M)}} E_q, E_p \subseteq [x]}$$

$$\boxed{[q] = \{p \in K : \exists_{[x] \in \Sigma^*/\approx_{L(M)}} E_q, E_p \subset [x]\}}$$

what we put together by $\sim_M$

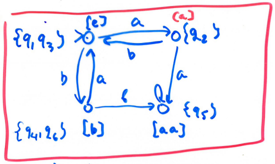
states of STANDARD aut

WORDS:

$\boxed{E_{\equiv_p} = \{p \in K : \text{p are put together in construction of the STANDARD automaton}\}}$

State Minimalization

Example (STANDARD)



$$\begin{tikzcd}
\{q_1 q_3\}\ [e] \arrow[r, bend left, "a"] \arrow[d, bend right, "b"'] & \{q_2\}\ [a] \arrow[l, bend left, "b"] \arrow[d, "a"] \\
\{q_4, q_6\}\ [b] \arrow[u, bend right, "a"'] \arrow[r, "b"] & \{q_5\}\ [aa] \arrow[loop above, "a b"]
\end{tikzcd}$$

We had:

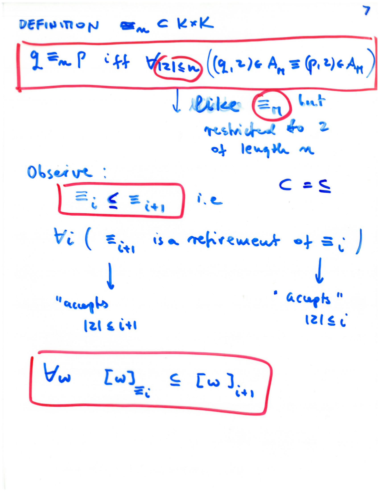
$$E_{q_1}, E_{q_3} \subseteq [e]$$
$$E_{q_2} \subseteq [a]$$
$$E_{q_4}, E_{q_6} \subseteq [b]$$
$$E_{q_5} \subseteq [aa]$$

So we get:

$$\boxed{K/_{\equiv_M} = \{\{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}\}}$$

equivalence classes of $\equiv_M$

State Minimalization



DEFINITION $\equiv_n \subset K \times K$

$$\boxed{q \equiv_n p \text{ iff } \forall_{|z| \le n} \left((q,z) \in A_M \equiv (p,z) \in A_M \right)}$$

$\downarrow$ like $\equiv_M$ but restricted to z of length n

Observe :

$\boxed{\equiv_i \le \equiv_{i+1}}$ i.e $\subset = \le$

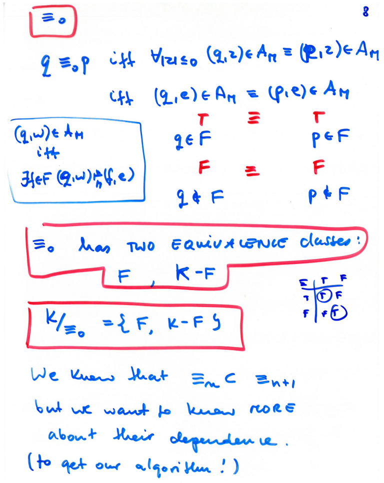
$\forall i$ ($\equiv_{i+1}$ is a refirement of $\equiv_i$)

$\downarrow$ "accepts $|z| \le i+1$

$\downarrow$ "accepts" $|z| \le i$

$$\boxed{\forall w \quad [w]_{\equiv_i} \subseteq [w]_{i+1}}$$

State Minimalization



$\boxed{\equiv_0}$

8

$q \equiv_0 p$ iff $\forall_{|z|\le 0}\ (q,z)\in A_M \equiv (p,z)\in A_M$

iff $(q,e)\in A_M \equiv (p,e)\in A_M$

$\boxed{\begin{array}{c}(q,w)\in A_M \\ \text{iff} \\ \exists_{f\in F}\ (q,w)\vdash^*_M (f,e)\end{array}}$

$$\overset{T}{q\in F} \quad \equiv \quad \overset{T}{p\in F}$$

$$\overset{F}{q\notin F} \quad \equiv \quad \overset{F}{p\notin F}$$

$\equiv$	T	F
T	T	F
F	F	T

$\boxed{\equiv_0 \text{ has TWO EQUIVALENCE classes: } F,\ K-F}$

$\boxed{K/_{\equiv_0} = \{F,\ K-F\}}$

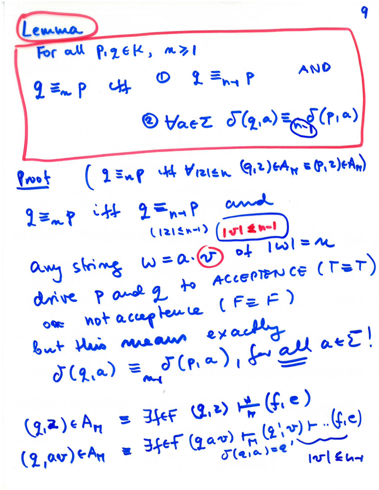
We know that $\equiv_n \subset \equiv_{n+1}$

but we want to know MORE

about their dependence.

(to get our algorithm!)

State Minimalization



9

Lemma

For all $p, q \in K$, $n \geq 1$

$q \equiv_n p$ iff ① $q \equiv_{n-1} p$ AND

② $\forall a \in \Sigma$ $\delta(q,a) \equiv_{\boxed{n-1}} \delta(p,a)$

<u>Proof</u> ($q \equiv_n p$ iff $\forall |z| \leq n$ $(q,z) \in A_M \equiv (p,z) \in A_M$)

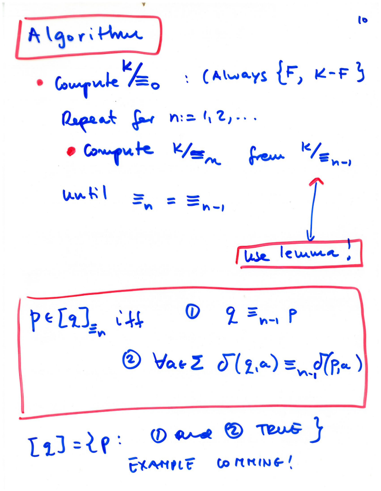
$q \equiv_n p$ iff $q \equiv_{n-1} p$ and
$(|z| \leq n-1)$ $\boxed{|v| \leq n-1}$

any string $w = a \cdot \boxed{v}$ of $|w| = n$
drive p and q to ACCEPTENCE $(T \equiv T)$
~~or~~ not acceptence $(F \equiv F)$
but this means exactly
$\delta(q,a) \equiv_{n-1} \delta(p,a)$, for <u>all</u> $a \in \Sigma$!

$(q,z) \in A_M \equiv \exists f \in F$ $(q,z) \vdash_M^* (f,e)$

$(q,av) \in A_M \equiv \exists f \in F$ $(q\,a\,v) \vdash_M \underset{\delta(q,a)=q'}{(q',v)} \underbrace{\vdash \dots (f,e)}_{|v| \leq n-1}$

State Minimalization



10

$\boxed{\text{Algorithm}}$

- Compute $K/\equiv_0$: (Always $\{F, K-F\}$

```
Repeat for $n := 1, 2, \dots$
    • Compute $K/\equiv_n$ from $K/\equiv_{n-1}$
until $\equiv_n = \equiv_{n-1}$
```

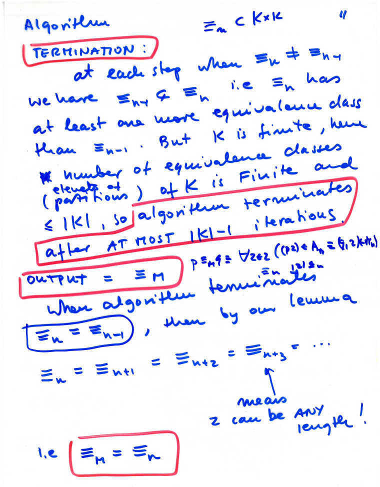
$\boxed{\text{Use lemma!}}$

$$\boxed{p \in [q]_{\equiv_n} \text{ iff } \quad ①\ q \equiv_{n-1} p \qquad ②\ \forall a \in \Sigma\ \delta(q,a) \equiv_{n-1} \delta(p,a)}$$

$[q] = \{p :$ ① and ② TRUE $\}$

EXAMPLE COMMING!

State Minimalization



Algorithm $\quad \equiv_n \subset K \times K \quad$ "

$\boxed{\text{TERMINATION:}}$

at each step when $\equiv_n \neq \equiv_{n-1}$

we have $\equiv_{n-1}$ & $\equiv_n$ i.e $\equiv_n$ has at least one more equivalence class than $\equiv_{n-1}$. But K is finite, hence number of equivalence classes (elements of partitions) of K is Finite and $\leq |K|$, so $\boxed{\text{algorithm terminates after AT MOST } |K|-1 \text{ iterations.}}$

$p \equiv_M q \equiv \forall z \in \Sigma^* ((p,z) \in A_n \equiv (q,z) \in A_n)$

$\equiv_n \quad |z| \leq n$

$\boxed{\text{OUTPUT} = \equiv_M}$

when algorithm terminates

$\boxed{\equiv_n = \equiv_{n-1}}$, then by our lemma

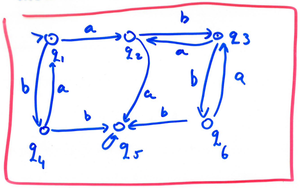
$\equiv_n = \equiv_{n+1} = \equiv_{n+2} = \equiv_{n+3} = \dots$

means z can be ANY length!

i.e $\boxed{\equiv_M = \equiv_n}$

State Minimalization

BACK TO OUR EXAMPLE



$\equiv_0$: $K/_{\equiv_0} = \{ \{q_1, q_3\}, \{q_2, q_4, q_5, q_6\} \}$

$\uparrow$ or $\boxed{\text{SPLIT}}$

$\equiv_1$: $\boxed{q \equiv_1 p \text{ iff } ① \; q \equiv_0 p \quad ② \; \forall a \in \Sigma \;\; \delta(q,a) \equiv_0 \delta(p,a)}$

$\overset{?}{\boxed{q_1 \equiv_1 q_3}}$ iff ① $q_1 \equiv_0 q_3$ yes

yes

② $\forall a \in \Sigma \;\; \delta(q_1, a) \equiv_0 \delta(q_3, a)$

$\delta(q_1, a) = q_2 \overset{!}{\equiv_0} \overset{''}{q_2}$ yes

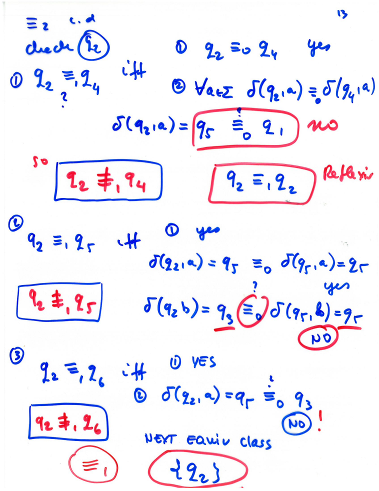
$\boxed{\{q_1, q_3\}}$

is in $K/_{\equiv_1}$

$\delta(q_1, b) = q_4 \overset{!}{\equiv_0} \delta(q_3, b) = q_2$

$\boxed{\text{yes}}$

State Minimalization



$\equiv_2$ c.d 13

check $\boxed{q_2}$

① $q_2 \overset{?}{\equiv_1} q_4$ iff ① $q_2 \equiv_0 q_4$ yes

② $\forall a \in \Sigma \;\; \delta(q_2,a) \overset{?}{\equiv_0} \delta(q_4,a)$

$\delta(q_2,a) = \boxed{q_5 \overset{?}{\equiv_0} q_1}$ no

so $\boxed{q_2 \not\equiv_1 q_4}$ $\boxed{q_2 \equiv_1 q_2}$ Reflexive

② $q_2 \equiv_1 q_5$ iff ① yes

$\delta(q_2,a) = q_5 \equiv_0 \delta(q_5,a) = q_5$

yes

$\boxed{q_2 \not\equiv_1 q_5}$ $\delta(q_2,b) = q_3 \overset{?}{\equiv_0} \delta(q_5,b) = q_5$ NO

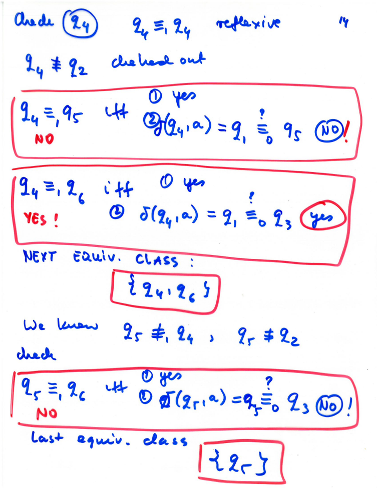
③ $q_2 \equiv_1 q_6$ iff ① YES

② $\delta(q_2,a) = q_5 \overset{?}{\equiv_0} q_3$ NO !

$\boxed{q_2 \not\equiv_1 q_6}$

$\equiv_1$ NEXT Equiv class $\{q_2\}$

State Minimalization



Check $\boxed{q_4}$ $q_4 \equiv_1 q_4$ reflexive 14

$q_4 \not\equiv q_2$ checked out

$q_4 \equiv_1 q_5$ iff ① yes

NO ② $\delta(q_4, a) = q_1 \overset{?}{\equiv}_0 q_5$ $\boxed{\text{NO}}$!

$q_4 \equiv_1 q_6$ iff ① yes

YES ! ② $\delta(q_4, a) = q_1 \overset{?}{\equiv}_0 q_3$ $\boxed{\text{yes}}$

NEXT EQUIV. CLASS :

$\boxed{\{q_4, q_6\}}$

We know $q_5 \not\equiv_1 q_4$, $q_5 \not\equiv q_2$

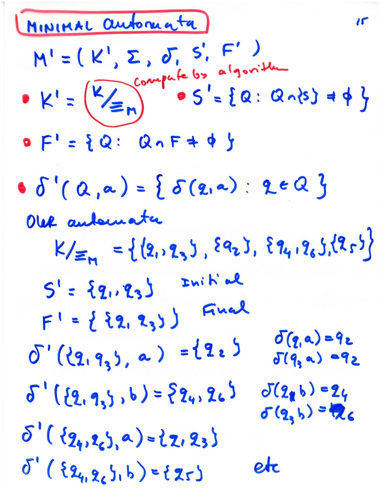
check

$q_5 \equiv_1 q_6$ iff ① yes

NO ② $\delta(q_5, a) = q_5 \overset{?}{\equiv}_0 q_3$ $\boxed{\text{NO}}$!

Last equiv. class $\boxed{\{q_5\}}$

State Minimalization



MINIMAL automata

15

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

compute by algorithm

- $K' = \boxed{K/\equiv_M}$
- $s' = \{Q : Q \cap \{s\} \neq \emptyset\}$
- $F' = \{Q : Q \cap F \neq \emptyset\}$
- $\delta'(Q, a) = \{\delta(q, a) : q \in Q\}$

Our automata

$K/\equiv_M = \{\{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}\}$

$s' = \{q_1, q_3\}$ Initial

$F' = \{\{q_1, q_3\}\}$ Final

$\delta'(\{q_1, q_3\}, a) = \{q_2\}$ $\quad \delta(q_1, a) = q_2$, $\delta(q_3, a) = q_2$

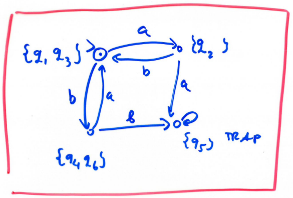
$\delta'(\{q_1, q_3\}, b) = \{q_4, q_6\}$ $\quad \delta(q_1, b) = q_4$, $\delta(q_3, b) = q_6$

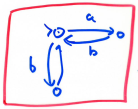
$\delta'(\{q_4, q_6\}, a) = \{q_1, q_3\}$

$\delta'(\{q_4, q_6\}, b) = \{q_5\}$ $\quad$ etc

State Minimalization

MINIMAL





– trap nondeterm

+ trap d.f.a

Nondeterministic

$L = (ab \cup ba)^*$