

Languages and Machines (Module 7 TCS+IAM)  
L&M 1: Languages, Regular Expressions,  
Automata  
Ch 1 (preliminaries); Ch 2:1-4; Ch 5:1-3

Alexander Skopalik

Discrete Mathematics and Mathematical Programming, Applied Mathematics, University of Twente

Lecture 1

Contents

- 1 Introduction
  - Logistics
  - Contents of “Languages and Machines”
- 2 Language and Words
  - What is a language?
  - Operations on Words
  - Operations on Languages
- 3 Regular Languages and Expressions
  - Kleene Star
  - Regular Expressions
  - Regular Languages
- 4 Deterministic Finite Automata
  - DFA and corresponding language
  - Application: specification of systems
  - Incomplete DFA; complement and intersection

Contents

- 1 Introduction
  - Logistics
  - Contents of “Languages and Machines”
- 2 Language and Words
  - What is a language?
  - Operations on Words
  - Operations on Languages
- 3 Regular Languages and Expressions
  - Kleene Star
  - Regular Expressions
  - Regular Languages
- 4 Deterministic Finite Automata
  - DFA and corresponding language
  - Application: specification of systems
  - Incomplete DFA; complement and intersection

Logistics (1)

Teachers

- Lectures: Alexander Skopalik
- Exercises: Marcus Gerhold, Sebastiaan Joosten, Alexander Skopalik, Carlos Budde, Mohsen Safari

Mandatory materials

- *Languages and Machines*, Thomas A. Sudkamp, 3rd edition difficult to get, alternative:
- *Introduction to Automata Theory, Languages, and Computation*, Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. , 3rd edition. Main difference:  $\lambda \rightarrow \epsilon$ .
- Canvas: exercises + handout (Lecture 4)

Optional tool

- JFLAP can be used to demonstrate most of our constructions.

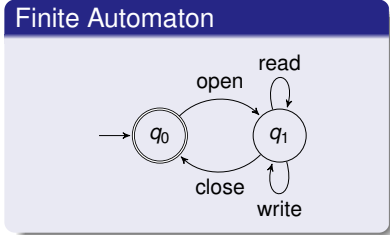
# Logistics (2)

## Setup of theme Languages and Machines:

- **Lecture:** discussion of the course subjects, **incomplete!**
- Make sure to read **all material from the book!**
- Goal of the exercises: **proofs and constructions** – exam training
- Bring your **book** to the exercises!
- **Bonus points:** 14/3 and 28/3
- Use JFLAP on bigger examples, **increase your understanding**
- Link to project: **Partition Refinement** = minimization of automata
- Exam: March 29 (13:45), **closed book**

# Concrete Example: a correct file system

**Regular Expression**  
 $(open (read + write)^* close)^*$



**Some correct sequences:**  
 open, read, read, write, close;  
 open, close, open, close;  
 open, read, close, open, read,  
 read, close;

**Some incorrect sequences:**  
 open, open;  
 open, read, write;  
 open, read, write, close, write;  
 read, write;

We are interested in the **Language** of **words** that are accepted by this **Regular Expression** / **Finite Automaton**.

# Goal of “Languages and Machines”

## Mathematical studies of “computation”

- What is computing?
  - Transformation from input to output
  - **input:** sequence of symbols = **word**
  - **output:** sequence of symbols, or one decision (1 bit yes/no)
- What is a computer?
  - control + data = automaton + tape
  - finite? infinite?

## What do we wish to accomplish?

- Finite descriptions of infinite sets of words = **language**
  - Automata, Expressions, Grammars, Turing Machines
- Algorithms to manipulate those descriptions = **compilers**
  - transformations, analyzing, compilers
- **Prove limits** to what is **computable** in principle

# Typical theoretical questions

- 1 Can we define the same languages by using Regular Expressions and by using Finite Automata?
- 2 If so, how can we transform any Regular Expression into a Finite Automaton? (and vice versa)
- 3 How can we check (by an algorithm) that two given Finite Automata accept the same language?
- 4 Can we compute the smallest Finite Automaton that accepts some language?
- 5 Are there languages that cannot be expressed by Finite Automata at all? Can this be proved rigorously?
- 6 Are there stronger mechanisms than RE / FA to define languages?
- 7 Is there a universal mechanism that can define all languages?

# Contents “Languages and Machines”

## Applications on language ..... hero: **Noam Chomsky (1928)**

- Translation projects natural language (e.g.: Russian-English)
- Search and index texts

## Applications in Computer Science:

- Specification and verification of (safe) computer systems
- Compiler construction: definition/analysis of progr. languages
- Understand and improve complexity of algorithms

## Mathematical mechanisms to define a language

- Regular expressions ..... (starting today)
- Grammars ..... (2nd half of lectures)

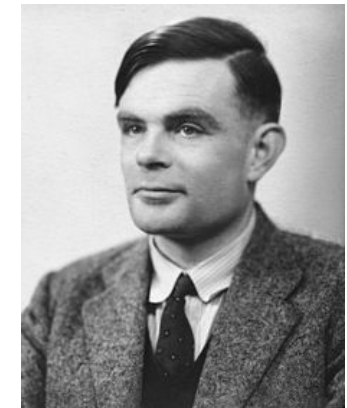
## Calculators with increasing computing power ..... hero: **Alan Turing**

- Automata with finite memory ..... (starting today)
- Automata with infinite memory ..... (2nd half of lectures)

# Heroes and pioneers



Noam Chomsky (1928)



Alan Turing (1912-1954)

# Contents

- 1 Introduction
  - Logistics
  - Contents of “Languages and Machines”
- 2 Language and Words
  - What is a language?
  - Operations on Words
  - Operations on Languages
- 3 Regular Languages and Expressions
  - Kleene Star
  - Regular Expressions
  - Regular Languages
- 4 Deterministic Finite Automata
  - DFA and corresponding language
  - Application: specification of systems
  - Incomplete DFA; complement and intersection

# What is a language?

## We use the following technical definition of “language”:

- 1 **Symbols:** an alphabet of “atomic” letters
- 2 **Words:** a word is a sequence of symbols
- 3 **Language:** a language is a set of words

## Somewhat more formal:

- 1 Given an alphabet  $\Sigma$ .
- 2  $\Sigma^*$  is the set of all words  $a_1 \dots a_n$ , such that  $a_i \in \Sigma$  for every  $1 \leq i \leq n$  ( $n = 0$  is possible).
- 3 A language  $L$  is a set of words; so:  $L \subseteq \Sigma^*$ .

## Questions: let $\Sigma = \{a, b, c\}$ .

Is  $\Sigma^*$  a language? Is  $a$  a language? Is  $\{a, ab\}$  a language? Is  $\emptyset$  a language?

## Examples from natural language

### The language of English words: (in principle finite)

- 1 **Alphabet:** letters a-z,
- 2 **Words:** day, dog, asxxbr, taal, ...
- 3 **Language:** The set of English words
  - **yes:** day, dog, book, ...
  - **no:** asxxbr, taal

### The language of English sentences: (in principle infinite)

- 1 **Alphabet:** words (day, bites, dog, the, ...)
- 2 **Words:** sequences of words, e.g.: "The The The", "The dog bites"
- 3 **Language:** The set of grammatically correct sentences
  - **yes:** "the dog bites the cat", "the day bites the day", "the dog bites the cat, that catches the mouse"
  - **no:** "the dog cat bites day"

## Technical examples

### Programming languages

- **Alphabet:** keywords, identifiers, etc.
- **Language:** all correct expressions in Matlab
- **Language:** alle correct Java programs

### System specifications

- **Alphabet:** events/actions (input/output)
- e.g. elevator: {up,down,open,closed}
- e.g. rails: barriers open/closed, switch left/right, signal green/yellow/red
- **Language:** safe behavior = set of specific sequences of events

## Abstract examples

### $\Sigma = \{ ( , ) \}$ , language consists of "correctly spelled" words

- **yes:**  $(( ))$ ,  $()(())$ ,  $((())(()))$ , ...
- **no:**  $)()()$ ,  $((()$ , ...

### $\Sigma = \{ a, b \}$ , the language of "palindromes"

- **yes:**  $a$ ,  $abba$ ,  $ababa$ ,  $baaaab$ , ...

### $\Sigma = \{ ( , ) , + , * , 1 , \dots , 9 , 0 \}$ , mathematical expressions.

- **yes:**  $1 + 3$ ,  $(13 * 24) + 18$ , ...
- **no:**  $13+$ ,  $13 + *5$ ,  $13 - 5$

### How to exactly define these languages?

- above languages contain **infinitely** many words.
- every separate word has **finite** length.

## Operations on words (induction, recursion)

### Recursive definition $\Sigma^*$ (set of words, eq. 2.1.1)

- 1  $\lambda$  is the **empty word**
- 2 If  $a \in \Sigma$  and  $w$  is already a word, then  $aw$  is also a word
- 3 All words can be obtained by applying (1) and (2) finitely often.

### Question:

- What are:  $\lambda$ ,  $\{\lambda\}$ ,  $\emptyset$ ,  $\{\emptyset\}$ ?  
 ..... (empty language  $\neq$  language with empty word)

### Recursive definition $u \cdot v$ (concatenation, eq. 2.1.3)

Given  $u$  and  $v$ , define word  $u \cdot v$  **recursive** (over  $u$ ):

- 1 if  $u = \lambda$ :  $\lambda \cdot v = v$
- 2 if  $u = aw$ :  $(aw) \cdot v = a(w \cdot v)$

### Proof with induction over $u$ that $(u \cdot v) \cdot w = u \cdot (v \cdot w)$

## More operations on words:

- Empty word:  $\lambda$
- Concatenation:  $uv$  (from here on we omit  $\cdot$ )

### Reverse (the reverse word)

- $u^R$  (the reverse word) (define recursive!)
- Theorems:  $(uv)^R = v^R u^R$  (proof with induction over  $u$ !)
- Definition of the language of palindromes:  $\{w \mid w = w^R\}$

### Recap. Example: $(aab)^3 = aabaabaab$

If  $u \in \Sigma^*$  and  $n \in \mathbb{N}$ , define  $u^n$  recursive over  $n$ :

- $u^0 = \lambda$
- $u^{n+1} = u(u^n)$

**Proof by yourself:**  $u^m u^n = u^{m+n}$  and  $(u^m)^n = u^{mn}$

## Operations on languages

- Empty:  $\emptyset$ ; singleton:  $\{w\}$  (for every  $w \in \Sigma^*$ )
- Assume below, that  $L, L_1, L_2 \subseteq \Sigma^*$ .

### Set operations on languages

- Union:  $L_1 \cup L_2 = \{u \mid u \in L_1 \vee u \in L_2\}$
- Intersection:  $L_1 \cap L_2 = \{u \mid u \in L_1 \wedge u \in L_2\}$
- Complement:  $\bar{L} = \{u \mid u \in \Sigma^* \wedge u \notin L\}$

### Operations on words transfer to languages

- Reverse:  $L^R = \{u^R \mid u \in L\}$
- Concatenation:  $L_1 L_2 = \{uv \mid u \in L_1 \wedge v \in L_2\}$
- Example:  $\{a, ab\}\{a, ba\} = \{aa, aba, abba\}$

**Theorem:**  $|L_1 L_2| \leq |L_1| |L_2|$  and  $|L_1 \cup L_2| \leq |L_1| + |L_2|$

## Repeat-operations on languages

For  $L \subseteq \Sigma^*$ , define  $L^n$  with recursion over  $n$

- $L^0 = \{\lambda\}$ ;  $L^{n+1} = LL^n$  (concatenation of languages)
- **Beware:**  $\{u^n \mid u \in L\} \subset L^n$  (for  $n \geq 2$  and  $|L| > 1$ )

**Example:** let  $\Sigma = \{a, b\}$ , let  $L = \{aa, bb\}$

- $L^2 = \{aa, bb\}\{aa, bb\} = \{aaaa, aabb, bbaa, bbbb\}$
- $\{u^2 \mid u \in L\} = \{aaaa, bbbb\}$  (always use the same word)

Two important sets (Kleene star)

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \dots \quad \text{0 times or more}$$

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup L^3 \cup \dots \quad \text{1 time or more}$$

## Contents

- 1 Introduction
  - Logistics
  - Contents of "Languages and Machines"
- 2 Language and Words
  - What is a language?
  - Operations on Words
  - Operations on Languages
- 3 Regular Languages and Expressions
  - Kleene Star
  - Regular Expressions
  - Regular Languages
- 4 Deterministic Finite Automata
  - DFA and corresponding language
  - Application: specification of systems
  - Incomplete DFA; complement and intersection

## More about Kleene Star after: Stephen Cole Kleene (1909-1994)

### Observations

- $L^+ = LL^*$
- $L^* = \{\lambda\} \cup L^+$
- If  $\lambda \in L$ , then  $L^* = L^+$
- **Questions:** What are  $\emptyset^*$  and  $\emptyset^+$ ? (see also [Sudkamp] table 2.1)

### Examples

- $\{a, b\}^*$  contains  $\lambda, a, b, aaabb, a^{10}b^{12}, a^{199}, ab^{100}a$
- $\{ab, ba\}^+$  contains  $ab, baba, abbaabba$ , but **not**  $abbaaba$  or  $\lambda$
- $(\{a, b\}^* \cup \{cc\}^*)^*$  contains  $aab\ cc\ cc\ aa\ ba\ cccc$
- **Question:** does  $(\{a, b\}^* \cup \{cc\}^*)^* = (\{a, b\} \cup \{cc\})^*$  hold?
- **Question:** does  $(\{a, b\} \cup \{cc\})^* = (\{a, b\} \cup \{c\})^*$  hold?

## Regular expressions (important!)

### Regular expressions over $\Sigma$ are defined inductively:

- Basis:  $\emptyset, \lambda$  and  $a$  (for  $a \in \Sigma$ ) are regular expressions
- If  $E_1$  and  $E_2$  are regular expressions, then  $(E_1E_2), (E_1 \cup E_2)$  and  $(E_1^*)$  are also regular expressions
- There exist no other regular expressions

### A regular expression $E$ defines a languages $\mathcal{L}(E)$ as follows:

Expression $E$	Language $\mathcal{L}(E)$	Alternative notation
$\emptyset$	$\emptyset$	0
$\lambda$	$\{\lambda\}$	1 (or: $\epsilon$ )
$a$	$\{a\}$	$a$
$E_1 \cup E_2$	$\mathcal{L}(E_1) \cup \mathcal{L}(E_2)$	$E_1 + E_2$
$E_1 E_2$	$\mathcal{L}(E_1)\mathcal{L}(E_2)$	$E_1 \cdot E_2$
$E^*$	$\mathcal{L}(E)^*$	$E^*$

## Regular languages

### $L$ is regular, if a regular expression $E$ exists, such that $L = \mathcal{L}(E)$

- A **regular language** over  $\Sigma$  is thus a language that can be described by a **regular expression**.
- Not every language can be defined with a regular expression (but how to prove this in concrete instances?) (discussed later)

### Open problem:

- Regular languages are by definition closed under **union, concatenation** and **Kleene star**.
- Are regular languages closed under intersection? i.e.: if  $L_1$  and  $L_2$  are regular, then is  $L_1 \cap L_2$  regular?
  - Yes, but how can you see that??
  - Given  $E_1$  and  $E_2$ . **How to systematically construct**  $E_3$ , such that  $\mathcal{L}(E_3) = \mathcal{L}(E_1) \cap \mathcal{L}(E_2)$ ?
  - For example:  $(aaa)^* \cap (aa)^* = (aaaaaa)^*$

## Examples of regular languages (pattern matching)

### Examples, where $\Sigma = \{a, b, c\}$

- All words that start with  $ab$  and end with  $ba$ 
  - mathematically:  $\{abwba \mid w \in \Sigma^*\} \cup \{aba\}$
  - regular expression:  $ab(a \cup b \cup c)^*ba \cup aba$
- All words with an even number of a's: regular expression:  $((b \cup c)^*a(b \cup c)^*a)^*(b \cup c)^*$
- All words that contain  $ab$ :  $(a \cup b \cup c)^*ab(a \cup b \cup c)^*$
- All words that **do not** contain  $ab$  (difficult!):  $(b \cup a^*c)^*a^*$

### More open problems:

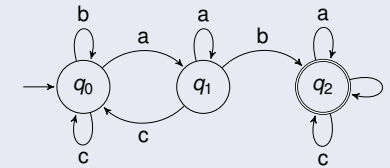
- How to compute the complement of  $E$  systematically?
- How to prove that  $\{a^n b^n \mid n \in \mathbb{N}\}$  is **not** regular?

# Contents

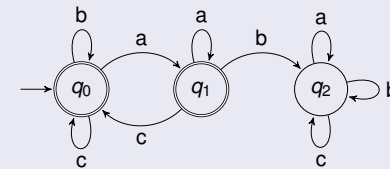
- 1 Introduction
  - Logistics
  - Contents of "Languages and Machines"
- 2 Language and Words
  - What is a language?
  - Operations on Words
  - Operations on Languages
- 3 Regular Languages and Expressions
  - Kleene Star
  - Regular Expressions
  - Regular Languages
- 4 Deterministic Finite Automata
  - DFA and corresponding language
  - Application: specification of systems
  - Incomplete DFA; complement and intersection

# Deterministic finite automata (examples)

Language of words over  $\{a, b, c\}$  that contain  $ab$ :



Language of words over  $\{a, b, c\}$  that **do not** contain  $ab$ :



- apparently the **complement** is simple for a DFA

# Deterministic finite automaton (definition)

A Deterministic Finite Automaton (DFA) is a 5-tuple:

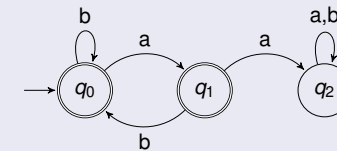
$$M = (Q, \Sigma, \delta, q_0, F)$$

Here, the components are:

- $Q$  is a finite set of states
- $q_0 \in Q$  is the unique initial state
- $\Sigma$  is a finite set of symbols, the alphabet.
- $\delta$  is a total function,  $\delta : Q \times \Sigma \rightarrow Q$  (transition function)
- $F \subseteq Q$  is the set of accepting states.

# Three equivalent notations

State diagram:



Mathematical definition:

$$Q = \{q_0, q_1, q_2\}; \Sigma = \{a, b\}; F = \{q_0, q_1\};$$

$$\delta(q_0, a) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2,$$

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

Transition table for  $\delta$ :

$\delta$	$a$	$b$
$q_0$	$q_1$	$q_0$
$q_1$	$q_2$	$q_0$
$q_2$	$q_2$	$q_2$

## Language accepted by a finite automaton (DFA)

the **language** that  $M$  accepts

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

$$\mathcal{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts the word } w\}$$

Simple intuition (will later be formally defined)

- You start in the initial state  $q_0$
- You read the word  $w$  symbol by symbol, and continuously follow the corresponding arrows according to  $\delta$
- At the end, you check whether you are in an accepting state of  $F$

## Acceptation of words (exactly, via $\vdash$ )

A **configuration** of a DFA is a pair  $[q, w] \in Q \times \Sigma^*$

- $q$  is the current state
- $w$  is the word that is left to be read

Let  $M = (Q, \Sigma, \delta, q_0, F)$  and define sequentially

- A **one-step-relation**  $\vdash \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ :  
 $[q_i, aw] \vdash [\delta(q_i, a), w]$  (read: "infers")
- The reflexive-transitive closure  $\vdash^*$ :
  - **Contains**  $\vdash$ : If  $[q, w] \vdash [q', w']$  then  $[q, w] \vdash^* [q', w']$
  - **Reflexive**:  $[q, w] \vdash^* [q, w]$
  - **Transitive**: If  $[q_1, w_1] \vdash^* [q_2, w_2]$  and  $[q_2, w_2] \vdash^* [q_3, w_3]$ , then also  $[q_1, w_1] \vdash^* [q_3, w_3]$ .
- $M$  **accepts**  $w$ , if a  $q \in F$  exists, with  $[q_0, w] \vdash^* [q, \lambda]$ .

## Acceptance of words (exactly, via $\hat{\delta}$ )

The **extended transition function** has type  $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

- Let  $M = (Q, \Sigma, \delta, q_0, F)$ .
- Define  $\hat{\delta}(q, u)$  recursively over  $u$ :
  - Basis:  $\hat{\delta}(q, \lambda) = q$
  - Step:  $\hat{\delta}(q, aw) = \hat{\delta}(\delta(q, a), w)$
- Now define:  $M$  **accepts**  $w$ , if  $\hat{\delta}(q_0, w) \in F$ .

**Theorem**: these two definitions are equivalent.

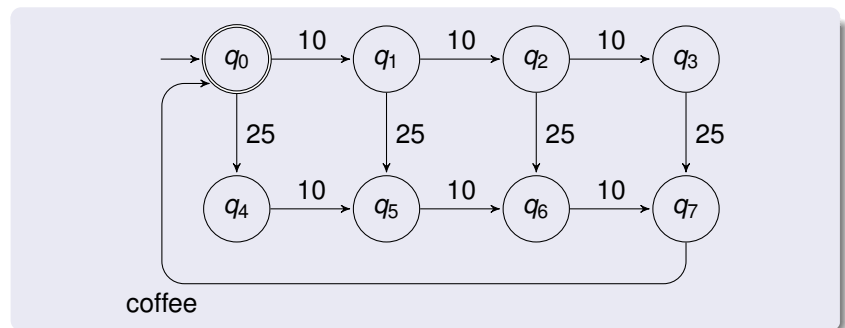
- In particular:  
 $\hat{\delta}(q_0, w) = q$  if and only if  $[q_0, w] \vdash^* [q, \lambda]$ .
- So:

$$\mathcal{L}(M) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

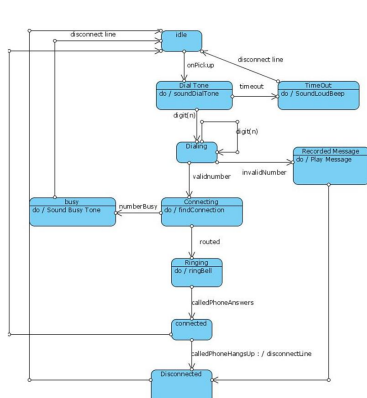
## Other application: specification of system behavior

Ancient coffee machine

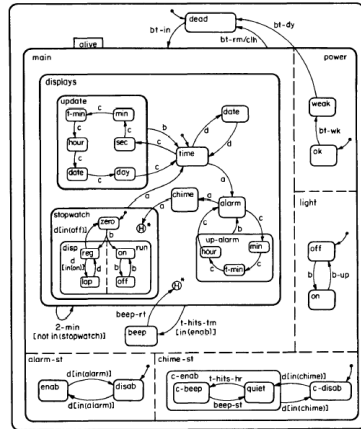
- $\Sigma = \{10, 25, \text{coffee}\}$ ;  $Q = \{q_0, \dots, q_7\}$
- **What is going on here with  $\delta$ ?**



## Other application: specification of system behavior



Telephone operation



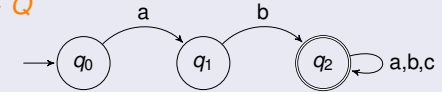
Digital Watch

Also: Communication Protocols, Railway Interlockings, Cell Biology

## Incomplete deterministic finite automaton

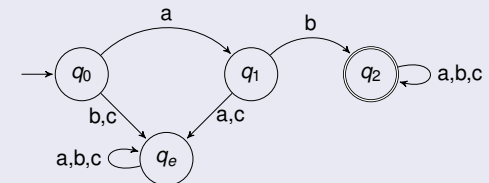
- An incomplete automation has a **partial** function:

$$\delta : Q \times \Sigma \rightarrow Q$$



$\delta(q_0, a) = q_1$ , but  $\delta(q_0, b)$  is undefined.

- Agreement:** If  $w$  can only be read partially, it is **not** accepted by the incomplete DFA
- You can always complete an incomplete automation by adding an error state, that never accepts again:



$$\delta'(q_0, a) = q_1, \text{ en } \delta'(q_0, b) = q_e \text{ en } \delta'(q_e, a) = q_e$$

## Complete an incomplete DFA

Precise construction:

- Given an incomplete DFA

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

- Construct the corresponding DFA  $M'$  as follows:

$$M' = (Q \cup \{q_e\}, \Sigma, \delta', q_0, F), \text{ where}$$

- $\delta'(q, a) = \delta(q, a)$  if  $q \in Q$  and  $\delta(q, a)$  is defined,
- $\delta'(q, a) = q_e$  if  $q \in Q$  but  $\delta(q, a)$  is not defined,
- $\delta'(q_e, a) = q_e$ .

Observations

- $M$  and  $M'$  are **equivalent** in the sense that  $\mathcal{L}(M) = \mathcal{L}(M')$ .
- An incomplete DFA requires less arrows, but the above shows that you cannot define **other** languages with it.

## Complement DFA (construction)

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

- Construct a corresponding DFA  $M'$  as follows:

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

- Theorem (5.3.3):**  $\mathcal{L}(M') = \Sigma^* - \mathcal{L}(M) = \overline{\mathcal{L}(M)}$

State of Affairs:

- Regular expressions are closed under **union**
- DFA languages are closed under **complement**

Open problem:

- Are these classes of languages the same?
- If yes: then also closed under intersection:  $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Solution:

- Non-deterministic Finite Automata (NFA): **Guess!**

## Discussed this lecture:

### Languages:

- Alphabet, Words, Language, Operations

### Regular Expressions:

- Choice, Sequence, Kleene star
- Regular languages  $\mathcal{L}(E)$

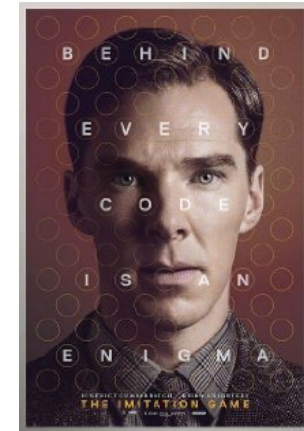
### Deterministic Finite Automata (DFA):

- States, Transitions, Acceptance
- Accepted language  $\mathcal{L}(M)$
- Complete versus incomplete (equally powerful)

## Don't miss out on this: The Imitation Game



Alan Turing



Benedict Cumberbatch