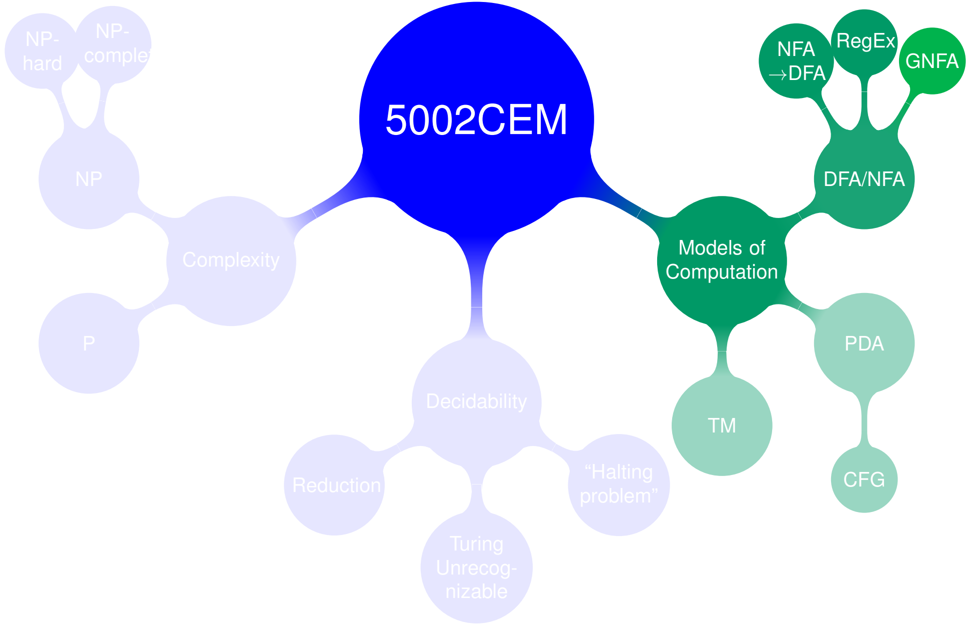
Models of Computation: DFAs & NFAs

Deterministic/Non-deterministic Finite Automata

Dr Kamal Bentahar

School of Computing, Electronics and Mathematics
Coventry University

Lecture 2


5002CEM
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM
Decidability
Reduction
Turing Unrecog-nizable
“Halting problem”
Complexity
NP
P
NP-hard
NP-comple

Last time. . .

Search problems

Given any (finite) search space, decide whether it contains a given item or not.

- Tedious but doable: **exhaustive search.**
- $\rightarrow$ **decision problem**: given data, decide if it has a certain property.
- Can divide all possible instances of the problem into **yes instances** and **no instances.**
- Simplify the way we describe the problems that machines will solve.
  - Turn *search* problems into *decision* problems

"Models of Computation"

- Want to think more precisely about **problems** and **computation.**
- $\rightarrow$ categorise them by the **type of computation** which resolves them.
- $\rightarrow$ idea of **models** of computation
- We introduce simple, theoretical machines and study their limits.
  - Far simpler than Von Neumann Machines, ...
  - ...but some have greater power than Von Neumann machines, ...
  - ......but cannot be created in reality!

Concept of “**language**”

Think about the English language:

- Alphabet: $a, b, c, \ldots, x, y, z$ (plus spaces, punctuation, etc.)
- However, not all strings over this alphabet are members of the language.
- $\rightarrow$ English is a **subset** of “all possible strings over its alphabet.”

In general:

- A problem **instance** can be represented as a **string of symbols**.
- Instances which yield **yes** are said to *belong* to the corresponding **language** for the problem.
- Instances which yield **no** (including invalid strings) do not belong to the language.

Concept of “**language**”

Decision problems can be encoded as problems of **language recognition.**

Problem: is a given number even?

Instance: An integer n (represented in binary).
Question: Is n even? (i.e. is it divisible by 2?)

Example

- Given $n = 12_{10} = 1100_2$, the answer is **yes** because $12 = 2 \times 6$.
- Given $n = 13_{10} = 1101_2$, the answer is **no** because $13 = 2 \times 6 + 1$.

Here:

$$\begin{aligned} \mathit{Integers} &= \{0, 1, 10, 11, 100, 101, 110, 111, 1000, \ldots\} \\ \mathit{Even} &= \{0, 10, 100, 110, 1000, \ldots\} \end{aligned}$$

and

$$\mathit{Even} \subset \mathit{Integers}$$

Language recognition

Problem: is a given number **even**?

Instance: an integer n (represented in binary).
Question: is n even? (i.e. is it divisible by 2?)

- n can be represented as a string in binary using only two symbols: 0, 1.
- Can write a **decision procedure** to decide if this string belongs to the language of **yes** instances.

```
b ← least significant bit of n.
if b = 0 then
    return yes
else
    return no
end if
```

Terminology

- Languages are defined over an **alphabet**, denoted by Σ.

  Σ is the set of allowable symbols for the language. ("*Sigma*")
- Σ^*: set of all possible strings over Σ, whose **length is finite.**

  ("*Sigma star*")

  If $\Sigma = \{0, 1\}$ then

  $$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots\}$$

- A language can be regarded as "a subset of Σ^*".

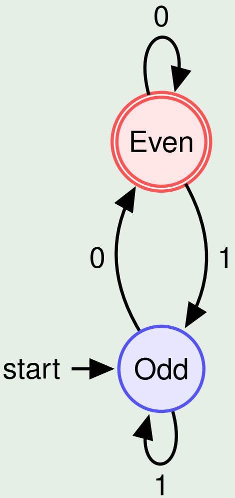
Example

If $\Sigma = \{0, 1\}$ then the language of even numbers $\mathit{Even} \subset \Sigma^*$ is:

$$\mathit{Even} = \{0, 00, 10, 000, 010, 100, \ldots\}$$

The **Deterministic Finite Automaton** (DFA) model

Example (Is a given binary number even?)



Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100

Informal definition of DFAs

The **Deterministic Finite Automaton** (DFA) model

A **directed and labelled graph** which describes how a string of symbols from an alphabet will be processed.

- Each vertex is called a **state**.
- Each directed edge is called a **transition**.
  - The edges are labelled with symbols from the alphabet.
- Each state must have **exactly one** transition defined for **every** symbol.
- <u>One</u> state is designated as the **start state**.
- <u>Some</u> states are designated as **accept states**.
- A string is processed symbol by symbol, following the respective transitions:
  - At the end, if we land on an accept state then the string is accepted,
  - otherwise it is rejected.

Important rules for DFAs

- Each state must have **exactly one transition defined *for each symbol***.
- There must be **exactly one start state**.
- There may be **multiple accept states**.
- There may be more than one symbol defined on a single transition.

JFLAP simulation time!

Example

Let us build DFAs over the alphabet $\{0, 1\}$ to recognize strings that:

- begin with 0;
- end with 1;
- either begin **or** end with 1;
- begin with 1 **and** contain at least one 0.

Formal definition of DFAs

Formal definition of a DFA

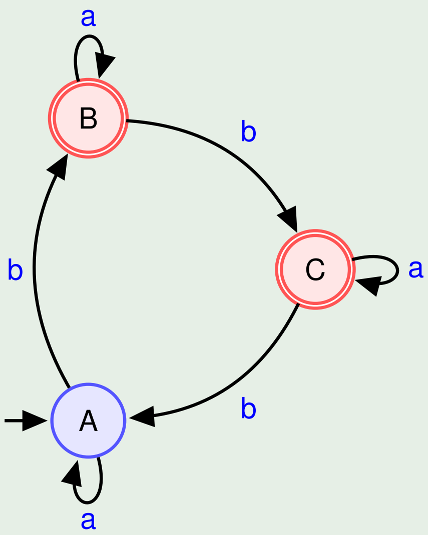
A *Deterministic Finite Automaton* (DFA) is defined by the 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, F)$ where:

- Q is a finite set called the **set of states.**
- Σ is a finite set called the **alphabet.**
- $\delta : Q \times \Sigma \to Q$ is a total function called the **transition function.**
- q_{start} is the unique **start state.** ($q_{\text{start}} \in Q$)
- F is **the set of accepting states.** ($F \subseteq Q$)

Recall:

- **Total function** means it is defined for “all its inputs.”
- Σ, δ: Sigma, delta. (Greek letters)
- $\in, \subseteq$: “**element** of a set”, “**subset** of a set, or equal”. (Set notation)

Example (Formal specification of a DFA)



This DFA is defined by the 5-tuple $(Q, \Sigma, \delta, q_{start}, F)$ where

- $Q = \{A, B, C\}$
- $\Sigma = \{a, b\}$
- $\delta(state, symbol)$ is given by the table:

	a	b
$\rightarrow$ A	A	B
$*$ B	B	C
$*$ C	C	A

  $\rightarrow$ indicates the start state
  $*$ the accept state(s).

- $q_{start} = A$
- $F = \{B, C\}$

Notation: functions/maps

$\delta\colon Q \times \Sigma \to Q$ means that:

- the function δ takes a pair (q, s) as input where:
  - q is a *state* from Q
  - s is an *alphabet symbol* from Σ,
- and returns a state from Q as the result.

This is usually given as a table, e.g.

	a	b
$\to q_0$	q_0	q_1
$*q_1$	q_0	q_2
$\vdots$	$\vdots$	$\vdots$

We put $\to$ next to the start state, and $*$ next to the accept states.

This means that:

$$\begin{aligned}\delta(q_0, a) &= q_0\\ \delta(q_0, b) &= q_1\\ \vdots &= \vdots\end{aligned}$$

[break]

Recall: Power set – set of all subsets

2^Q is the **set of all subsets of** Q (called: the **power set of** Q)

Example

If $Q = \{A, B, C\}$ then

$$2^Q = \left\{ \underbrace{\emptyset}_{\text{Empty set}}, \underbrace{\{A\}, \{B\}, \{C\}}_{\text{One element each}}, \underbrace{\{A, B\}, \{A, C\}, \{B, C\}}_{\text{Two elements each}}, \underbrace{\{A, B, C\}}_{Q} \right\}.$$

It has 8 elements $= 2^{\text{size of } Q} = 2^{\#Q} = 2^3 = 8$.

The **Nondeterministic Finite Automaton (NFA)** model

From the design point of view: NFAs are almost the same as DFAs.

DFA: every state has **one and only one outward transition** defined **for each symbol**.

NFA: **zero or more transition(s)** defined **for each symbol**.

Formally:

DFA: $\delta \colon Q \times \Sigma \to Q$ is a **total** function, i.e.

1. δ is defined for *every* pair (q, s) from $Q \times \Sigma$
2. δ sends (q, s) to a **state** from Q. (exactly one state, no more, no less)

NFA: $\delta \colon Q \times \Sigma \to 2^Q$ is a **partial** function, i.e.

1. δ is *not necessarily* defined for every pair (q, s) from $Q \times \Sigma$.
2. δ sends (q, s) to a **subset of** Q. (many, one, or no states)

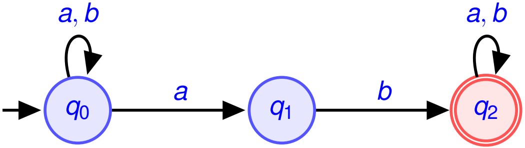
Formal description of NFAs

Definition of an NFA

A *Nondeterministic Finite Automaton* (NFA) is defined by the 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, F)$ where

- Q is a finite set called the **set of states**
- Σ is a finite set called the **alphabet**
- $\delta\colon Q \times \Sigma \to 2^Q$ is a partial function called the **transition function**
- q_{start} is the unique **start state.** ($q_0 \in Q$)
- F is the **set of accepting states.** ($F \subseteq Q$)

NFA example



$$Q = \{q_0, q_1, q_2\}$$

$$\Sigma = \{a, b\}$$

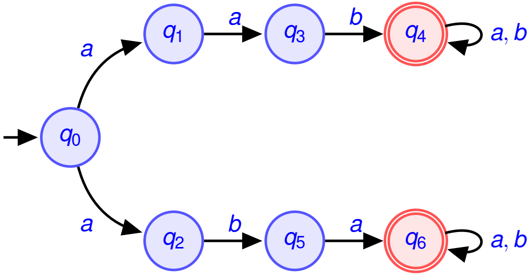
$$q_{\text{start}} = q_0$$

$$F = \{q_2\}$$

δ :

	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\{q_2\}$	$\{q_2\}$

NFA example



δ :

	a	b
$\rightarrow q_0$	$\{q_1, q_2\}$	$\emptyset$
q_1	$\{q_3\}$	$\emptyset$
q_2	$\emptyset$	$\{q_5\}$
q_3	$\emptyset$	$\{q_4\}$
$*q_4$	$\{q_4\}$	$\{q_4\}$
q_5	$\{q_6\}$	$\emptyset$
$*q_6$	$\{q_6\}$	$\{q_6\}$

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$$
$$\Sigma = \{a, b\}$$
$$q_{\text{start}} = q_0$$
$$F = \{q_4, q_6\}$$

JFLAP simulation time!

Example

Let us build DFAs over the alphabet $\{0, 1\}$ to recognize strings that:

- begin with 0;
- end with 1;
- either begin **or** end with 1;
- begin with 1 **and** contain at least one 0.

Next week...

Surprise: NFAs recognize exactly the same languages as DFAs!