COMP30026 Models of Computation

Finite-State Automata

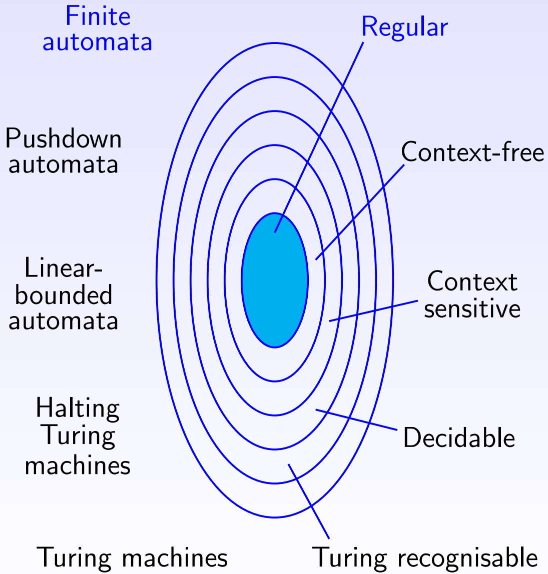
Anna Kalenkova

Lecture Week 7 Part 1

Semester 2, 2020

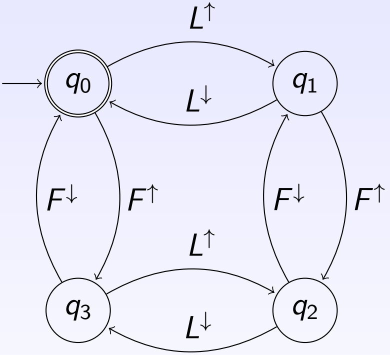
This Lecture is Being Recorded

Machines vs Languages



An Example Automaton

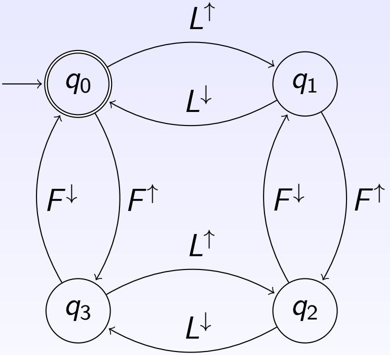
Consider a light/fan system managing lighting and air conditioning.



We start in a state (q_0) with the lights and fan turned off. We then can perform sequences steps, e.g., switch on/off the light ($L^{\uparrow}$, $L^{\downarrow}$), start/stop the fan ($F^{\uparrow}$, $F^{\downarrow}$). These sequences of steps lead us to one of the system's states.

An Example Automaton

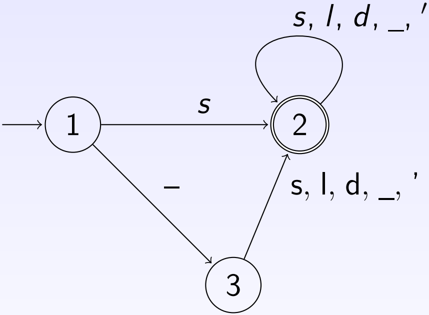
Consider a light/fan system managing lighting and air conditioning.



Sequence $\langle L^\uparrow, F^\uparrow, L^\downarrow, F^\downarrow, F^\uparrow, F^\downarrow \rangle$ is accepted by the system, while $\langle L^\uparrow, L^\downarrow, F^\downarrow, F^\uparrow, L^\uparrow \rangle$ is not, because it leaves us in q_1 state.

Example 2

Here is an automaton for recognising Haskell variable identifier:



s is an abbreviation for $a, \dots, z$ (the small or lower-case letters)
l is an abbreviation for $A, \dots, Z$ (the large or upper-case letters)
d is an abbreviation for $0, \dots, 9$ (the digits)

Formal Definition

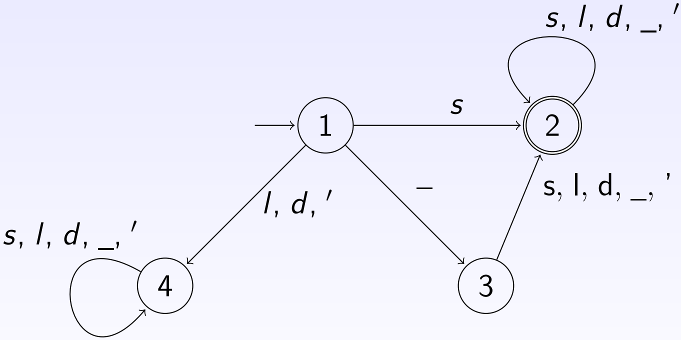
A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ are the accept states.

Here δ is a total function, that is, δ must be defined for all possible inputs.

Back to Example 2

To make it clear that the transition function is total, we should add a new state 4 and arcs to state 4 from state 1:



Strings and Languages

An alphabet Σ can be any non-empty finite set.

The elements of Σ are the symbols of the alphabet. Usually we choose symbols such as $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c}$, 1, 2, 3, ...

A string over Σ is a finite sequence of symbols from Σ.

We write the concatenation of a string y to a string x as xy.

The empty string is denoted by ϵ.

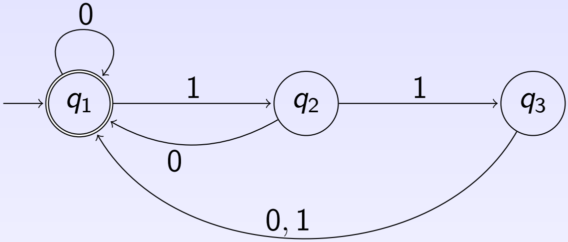
A language (over alphabet Σ) is a (finite or infinite) set of finite strings over Σ.

Σ^* denotes the set of all finite strings over Σ.

Examples of Languages over Alphabet $\Sigma = \{0, 1\}$

- $\emptyset$
- $\{\epsilon\}$
- $\{\epsilon, 0, 1\}$
- $\{00, 01, 10, 11\}$
- $\{\epsilon, 0, 00, 000, \ldots\}$
- $\{\epsilon, 0, 1, 00, 11, 000, 111, \ldots\}$
- $\{\epsilon, 01, 0011, 000111, \ldots\}$
- $\{w | w \text{ contains odd number of } 0\}$
- $\{w | \text{the length of } w \text{ is a multiple of } 3\}$
- $\{w | w \text{ is not empty string}\}$
- $\{w | w \text{ does not contain } 001\}$
- Σ^*

Example 3



The automaton M_1 (above) can be described precisely as

$$M_1 = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_1\}) \qquad \text{with}$$

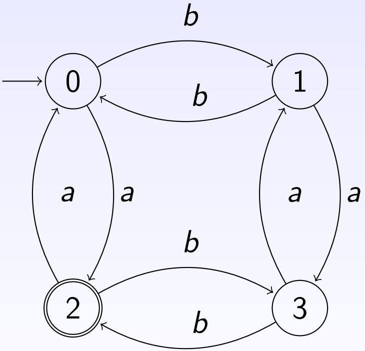
δ	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_1

$$L(M_1) = \left\{ w \;\middle|\; \begin{array}{l} w \text{ is } \epsilon\text{, or ends with 0, or the max legth of the sequence} \\ \text{of 1 symbols ending } w \text{ is a multiple of 3.} \end{array} \right.$$

is the language recognised by M_1.

Example 4

Which language is recognised by this machine?



Exercise

Consider the alphabet $\Sigma = \{0, 1\}$. We can interpret strings in Σ^* as numbers in binary representation.

Construct an automaton over Σ to recognise exactly those numbers that are multiples of 5.

Hint: Consider five states:

Acceptance and Recognition, Formally

What does it mean for an automaton to accept a string?

Let $M = (Q, \Sigma, \delta, q_0, F)$ and let $w = v_1 v_2 \cdots v_n$ be a string from Σ^*.

M accepts w iff there is a sequence of states $r_0, r_1, \ldots, r_n$, with each $r_i \in Q$, such that

1. $r_0 = q_0$
2. $\delta(r_i, v_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$
3. $r_n \in F$

M recognises language A iff $A = \{w \mid M \text{ accepts } w\}$.

Regular Languages

A language is regular iff there is a finite automaton that recognises it.

We shall soon see that there are languages which are not regular.

Regular Operations

Remember that, to us, a language is simply a set of strings.

Let A and B be languages. The regular operations are:

- **Union:** $A \cup B$
- **Concatenation:** $A \circ B = \{xy \mid x \in A, y \in B\}$
- **Kleene star:** $A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, \text{ each } x_i \in A\}$

Note that the empty string, ϵ, is always in A^*.

Regular Operations: Example

Let $A = \{\text{aa}, \text{abba}\}$ and $B = \{\text{a}, \text{ba}, \text{bba}, \text{bbba}, \ldots\}$.

$A \cup B = \{\text{a}, \text{aa}, \text{abba}, \text{ba}, \text{bba}, \text{bbba}, \ldots\}$.

$A \circ B = \{\text{aaa}, \text{abbaa}, \text{aaba}, \text{abbaba}, \text{aabba}, \text{abbabba}, \ldots\}$.

$$A^* = \left\{ \begin{array}{l} \epsilon, \text{aa}, \text{abba}, \text{aaaa}, \text{aaabba}, \text{abbaaa}, \text{abbaabba}, \\ \text{aaaaaa}, \text{aaaaabba}, \text{aaabbaaa}, \text{aaabbaabba}, \ldots \end{array} \right\}.$$

The regular languages are closed under the regular operations.

It will be easier to show this after we have considered non-deterministic automata.

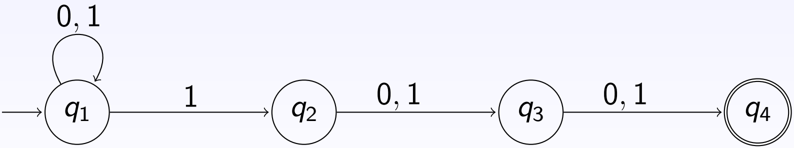
Nondeterminism

The type of machine we have seen so far is called a deterministic finite automaton, or DFA.

We now turn to non-deterministic finite automata, or NFAs.

Here is an NFA that recognises the language

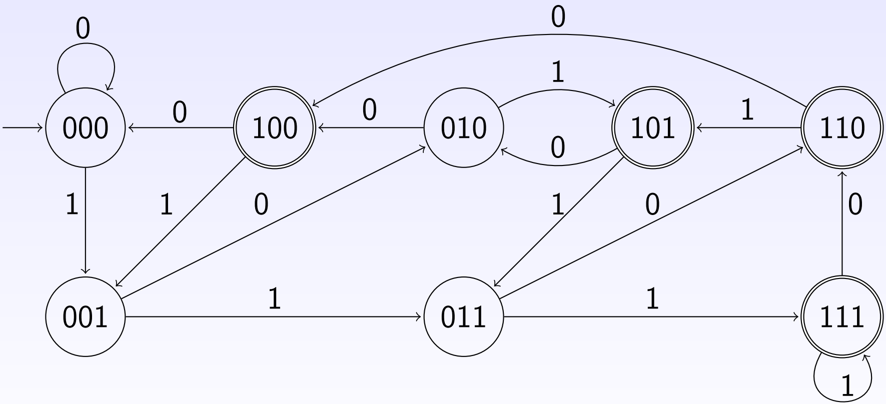
$$\left\{ w \;\middle|\; \begin{array}{l} w \in \{0,1\}^* \text{ has length 3 or more,} \\ \text{and the third last symbol in } w \text{ is } 1 \end{array} \right\}$$



Note: No transitions from q_4, and two possible transitions when we meet a 1 in state q_1.

Nondeterminism

The NFA is more intelligible than a DFA for the same language:



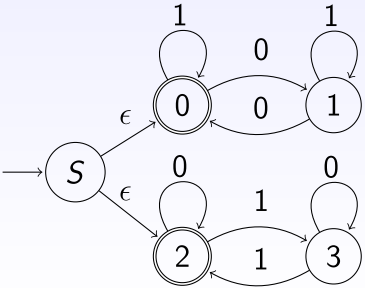
This is the simplest DFA that will do the job!

Epsilon Transitions

NFAs may also be allowed to move from one state to another without consuming input.

Such a transition is an ϵ transition.

Amongst other things, this is useful for constructing a machine to recognise the union of two languages:



Formal Definition

For any alphabet Σ let Σ_ϵ denote $\Sigma \cup \{\epsilon\}$.

An NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

NFA Acceptance and Recognition, Formally

The definition of what it means for an NFA N to accept a string says that it has to be possible to make the necessary transitions.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let $w = v_1 v_2 \cdots v_n$ where each v_i is a member of Σ_ϵ.

N accepts w iff there is a sequence of states $r_0, r_1, \ldots, r_n$, with each $r_i \in Q$, such that

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, v_{i+1})$ for $i = 0, \ldots, n-1$
3. $r_n \in F$

N recognises language A iff $A = \{w \mid N \text{ accepts } w\}$.

Next Lecture: Being Regular

More regular language theory in the next lecture.

In particular we shall see that NFAs are no more powerful than DFAs.