Deterministic finite automata

An **automaton** is a very useful and pervasive concept in software and telecommunication engineering.

Let us present first the simplest of the automata, called the **deterministic finite automaton**, or **DFA**. The most intuitive presentation of automata is by means of a graphic:

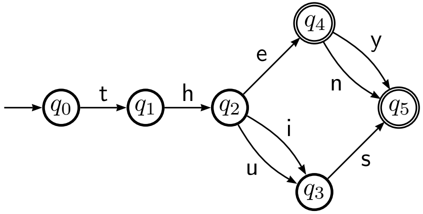
$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "\text{h}"] & q_1 \arrow[r, "\text{e}"] & q_2 \arrow[r, "\text{l}"] & q_3 \arrow[r, "\text{l}"] & q_4 \arrow[r, "\text{o}"] & q_5
\end{tikzcd}$$

One component is a circle or a double-circle, called **node**, with a name inside, called **state**. The other component is an arrow which connect two circles with a letter: these are **edges** and the letters are **labels**. The arrow which has no originating circle is a marker to distinguish a state, just as the double-circle denotes another special state: the former is an **initial state** and the latter is a **final state**.

DFA (cont)

It is possible to have several edges between the same pair of states but they must carry different labels. There may be several final states.

For instance, consider



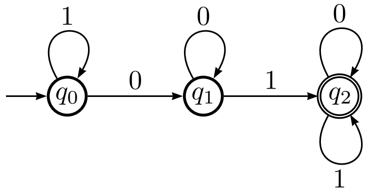
Note that here

1. the **alphabet** of labels is the English alphabet,
2. there are nodes without out-going edges carrying all possible letters: this DFA is not **complete**.

DFA (cont)

The graphic representation of an automaton is called a **transition diagram**.

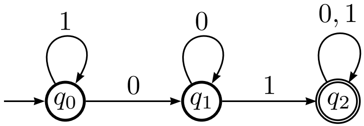
Here is another transition diagram over the alphabet $\{0, 1\}$, called binary alphabet.



Note that there are here some edges that connect a node to itself: these edges are called **loops**.

DFA (cont)

It is possible to simplify the transition diagram by merging edges which have the same pair of nodes and listing the labels, separated by commas:



DFA/Trie

Before giving a formal definition of DFAs, we can explain on transition diagrams how they are used on an example.

Let a short list of English words be composed of then, they, thus and this. Imagine we are given a word and asked to check if this word belongs to our list.

We can of course, compare the given word to each word in the list, one after the other, until a match is found or the end of the list is reached. But this is not efficient.

DFA/Trie (cont)

A better approach is to sort the words in the list in alphabetic order. This way, if we check the words in order and the current word is “greater” than the given word, we know it is not in the list.

This is the way we search a word in a dictionary (well, not exactly, but the analogy is close enough).

Our dictionary is: this, then, they and thus (order matters). Let us search the word they.

DFA/Trie (cont)

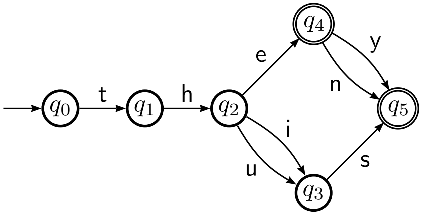
First, they is compared to this. The failure happens at the third letter (i.e., after three letter comparisons) and we must try the next word, then.

Second, they is compared to then. But then, we compare in turn the first letters t and h, i.e., we start again from the beginning of the searched word.

There is a way to avoid this inefficiency by considering a model of the dictionary that is not one-dimensional, like a list, but two-dimensional, like a DFA — called in this context a **trie**.

DFA/Trie (cont)

The trie corresponding to our example dictionary is exactly the transition diagram given page 80. Let us assume here that the word is followed by a special marker $.


$$\begin{tikzcd}
 & & & q_4 \arrow[dr, bend left, "y"] \arrow[dr, bend right, "n"'] & \\
\arrow[r] & q_0 \arrow[r, "t"] & q_1 \arrow[r, "h"] & q_2 \arrow[u, "e"] \arrow[dr, bend left, "i"] \arrow[dr, bend right, "u"'] & & q_5 \\
 & & & & q_3 \arrow[ur, "s"'] &
\end{tikzcd}$$

At the beginning of the search, the current state is the initial state, q_0, and the current letter in the searched word is t.

Then if there is an out-going edge from the current state that matches the current letter, we change the current state to the state pointed to by this edge and the current letter becomes the following one in the word or is $.

DFA/Trie (cont)

If the current letter is \$, i.e., all letters of the word have been successfully matched, and the current state is a final state (double-circled node), then the words belongs to the dictionary (in other words, the trie accepts the word or recognises it).

Otherwise, the word is not in the dictionary.

In the worst case, a word of length *n* is in the dictionary and it is found in exactly *n* letter comparisons.

DFA/Formal definition

Formally, a DFA $\mathcal{D}$ is a 5-tuple $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ where

1. a finite set of *states*, often noted Q;
2. an *initial state* $q_0 \in Q$;
3. a set of *final (or accepting) states* $F \subseteq Q$;
4. a finite set of *input symbols*, called *alphabet*, often noted Σ;
5. a *transition function* δ that takes a state and an input symbol and returns a state: if q is a state with an edge labeled a, the edge leads to state $\delta(q, a)$.

DFA/Recognised words

Independently of the interpretation of the states, we can define how a given word is accepted (or recognised) or rejected by a given DFA.

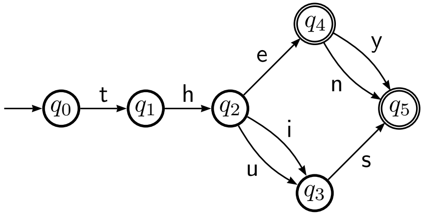
The word $a_1 a_2 \cdots a_n$, where $a_i \in \Sigma$, is recognised by the DFA $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ if

- for all $0 \leqslant i \leqslant n - 1$
- there is a sequence of states $q_i \in Q$ such that
- $\delta(q_i, a_{i+1}) = q_{i+1}$
- and $q_n \in F$.

The language recognised by $\mathcal{D}$, noted $L(\mathcal{D})$ is the set of words recognised by $\mathcal{D}$.

DFA/Recognised words/Example

For example, consider again the DFA page 80:



The word then is recognised because there is a sequence of states $(q_0, q_1, q_2, q_4, q_5)$ connected by edges which satisfies

$$\begin{aligned}\delta(q_0, \mathrm{t}) &= q_1\\ \delta(q_1, \mathrm{h}) &= q_2\\ \delta(q_2, \mathrm{e}) &= q_4\\ \delta(q_4, \mathrm{n}) &= q_5\end{aligned}$$

and $q_5 \in F$, i.e., q_5 is a final state.

DFA/Recognised language

It is easy to define formally $L(\mathcal{D})$.

Let $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$.

First, let us extend δ to words and let us call this extension $\hat{\delta}$:

- for all state $q \in Q$, let $\hat{\delta}(q, \varepsilon) = q$, where ε is the empty string;
- for all state $q \in Q$, all word $w \in \Sigma^*$, all input $a \in \Sigma$, let $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$.

Then the word w is recognised by $\mathcal{D}$ if $\hat{\delta}(q_0, w) \in F$. The language $L(\mathcal{D})$ recognised by $\mathcal{D}$ is defined as

$$L(\mathcal{D}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$$

DFA/Recognised language (cont)

For example, in our last example:

$$\begin{aligned}
\hat{\delta}(q_0, \epsilon) &= q_0 \\
\hat{\delta}(q_0, \mathrm{t}) &= \delta(\hat{\delta}(q_0, \epsilon), \mathrm{t}) = \delta(q_0, \mathrm{t}) = q_1 \\
\hat{\delta}(q_0, \mathrm{th}) &= \delta(\hat{\delta}(q_0, \mathrm{t}), \mathrm{h}) = \delta(q_1, \mathrm{h}) = q_2 \\
\hat{\delta}(q_0, \mathrm{the}) &= \delta(\hat{\delta}(q_0, \mathrm{th}), \mathrm{e}) = \delta(q_2, \mathrm{e}) = q_4 \\
\hat{\delta}(q_0, \mathrm{then}) &= \delta(\hat{\delta}(q_0, \mathrm{the}), \mathrm{n}) = \delta(q_4, \mathrm{n}) = q_5 \in F
\end{aligned}$$

DFA/Recognised language (cont)

What is the language recognised by the previous DFA? The definition states that this language is composed of all the words that are recognised.

Each recognised word corresponds to a **path**, i.e., a sequence of states connected by edges, from the initial state to a final state.

In our example, the language recognised is $\{\text{then}, \text{they}, \text{this}, \text{thus}\}$.

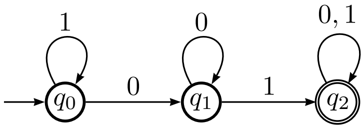
In other words:

$$\hat{\delta}(q_0, \text{then}) \in F$$
$$\hat{\delta}(q_0, \text{they}) \in F$$
$$\hat{\delta}(q_0, \text{this}) \in F$$
$$\hat{\delta}(q_0, \text{thus}) \in F$$

It is a finite language.

DFA/Recognised language (cont)

What is the language recognised by the DFA page 82?


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "1"] \arrow[r, "0"] & q_1 \arrow[loop above, "0"] \arrow[r, "1"] & \boxed{q_2} \arrow[loop above, "0,1"]
\end{tikzcd}$$

In English, this language can be defined as the set of strings of 0 and 1 containing 01.

Note that this an *infinite* language.

DFA/Transition diagrams

We can also redefine transition diagrams in terms of the concept of DFA.

A transition diagram for a DFA $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows:

1. for each state q in Q there is a **node**, i.e., a single circle with q inside;
2. for each state $q \in Q$ and each input symbol $a \in \Sigma$, if $\delta(q, a)$ exists, then there is an **edge**, i.e., an arrow, from the node denoting q to the node denoting $\delta(q, a)$ labeled by a; multiple edges can be merged into one and the labels are then separated by commas;
3. there is an edge coming to the node denoting q_0 without origin;
4. nodes corresponding to final states (i.e., in F) are double-circled.

DFA/Transition table

There is a compact textual way to represent the transition function of a DFA: a **transition table**.

The rows of the table correspond to the states and the columns correspond to the inputs (symbols). In other words, the entry for the row corresponding to state q and the column corresponding to input a is the state $\delta(q, a)$:

δ	$\dots$	a	$\dots$
$\vdots$			
q		$\delta(q, a)$	
$\vdots$			

DFA/Transition table/Example

The transition table corresponding to the function δ of our last example is

$\mathcal{D}$	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$\# q_2$	q_2	q_2

Actually, we added some extra information in the table: the initial state is marked with $\rightarrow$ and the final states are marked with $\#$.

Therefore, it is not only δ which is defined by means of the transition table here, but the whole DFA $\mathcal{D}$.

DFA/Example

We want to define formally a DFA which recognises the language L whose words contain an even number of 0's and an even number of 1's (the alphabet is binary).

We should understand that the role of the states here is to **not** to count the exact number of 0's and 1's that have been recognised before but this number **modulo 2**.

Therefore, there are four states because there are four cases:

1. there has been an even number of 0's and 1's (state q_0);
2. there has been an even number of 0's and an odd number of 1's (state q_1);
3. there has been an odd number of 0's and an even number of 1's (state q_2);
4. there has been an odd number of 0's and 1's (state q_3).

DFA/Example (cont)

What about the initial and final states?

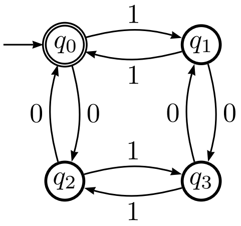
- State q_0 is the initial state because before considering any input, the number of 0's and 1's is zero and zero is even.
- State q_0 is the lone final state because its definition matches exactly the characteristic of L and no other state matches.

We know now almost how to specify the DFA for language L. It is

$$\mathcal{D} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

where the transition function δ is described by the following transition diagram.

DFA/Example (cont)



Notice how each input 0 causes the state to cross the horizontal line.

Thus, after seeing an even number of 0's we are always above the horizontal line, in state q_0 or q_1, and after seeing an odd number of 0's we are always below this line, in state q_2 or q_3.

There is a vertically symmetric situation for transitions on 1.

DFA/Example (cont)

We can also represent this DFA by a transition table:

$\mathcal{D}$	0	1
$\# \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

We can use this table to illustrate the construction of $\hat{\delta}$ from *delta*. Suppose the input is 110101. Since this string has even numbers of 0's and 1's, it belongs to L, i.e., we expect $\hat{\delta}(q_0, 110101) = q_0$, since q_0 is the sole final state.

DFA/Example (cont)

We can check this by computing step by step $\hat{\delta}(q_0, \mathtt{110101})$, from the shortest prefix to the longest (which is the word $\mathtt{110101}$ itself):

$$\begin{aligned}
\hat{\delta}(q_0, \varepsilon) &= q_0 \\
\hat{\delta}(q_0, \mathtt{1}) &= \delta(\hat{\delta}(q_0, \varepsilon), \mathtt{1}) = \delta(q_0, \mathtt{1}) = q_1 \\
\hat{\delta}(q_0, \mathtt{11}) &= \delta(\hat{\delta}(q_0, \mathtt{1}), \mathtt{1}) = \delta(q_1, \mathtt{1}) = q_0 \\
\hat{\delta}(q_0, \mathtt{110}) &= \delta(\hat{\delta}(q_0, \mathtt{11}), \mathtt{0}) = \delta(q_0, \mathtt{0}) = q_2 \\
\hat{\delta}(q_0, \mathtt{1101}) &= \delta(\hat{\delta}(q_0, \mathtt{110}), \mathtt{1}) = \delta(q_2, \mathtt{1}) = q_3 \\
\hat{\delta}(q_0, \mathtt{11010}) &= \delta(\hat{\delta}(q_0, \mathtt{1101}), \mathtt{0}) = \delta(q_3, \mathtt{0}) = q_1 \\
\hat{\delta}(q_0, \mathtt{110101}) &= \delta(\hat{\delta}(q_0, \mathtt{11010}), \mathtt{1}) = \delta(q_1, \mathtt{1}) = q_0 \in F
\end{aligned}$$

DFA/Dictionary

Let us use DFAs for implementing a dictionary, that is, a trie as presented at page 80.

First, here is how to add a word $w \in \Sigma^+$ to a trie $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$. The transition function δ is considered here as an array which is modified in place.

```
   $\textsc{Add}(w, (Q, \Sigma, \delta, q_0, F))$
1  $p \leftarrow q_0$; $i \leftarrow 1$
2  while $\delta[p, w[i]]$ is defined
3        do $p \leftarrow \delta[p, w[i]]$; $i \leftarrow i + 1$
4  for $j \leftarrow i$ to $|w|$
5        do $q \leftarrow \textsc{New-state}(Q)$; $Q \leftarrow Q \cup \{q\}$; $\delta[p, w[j]] \leftarrow q$; $p \leftarrow q$
6  $F \leftarrow F \cup \{p\}$
```