Finite-state Automata and Regular Languages

1 Formal Definition

A Deterministic Finite-State Automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma \rightarrow$ is the transition function
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states

Let $M = (Q, \Sigma, \delta, q_0, F)$ to be a DFA and let $w = w_1 w_2 ... w_n$ be a word over Σ. M accepts w if there is a sequence of states $r_0, r_1, r_2, ..., r_n$ satisfying the following conditions

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

2 Regular Languages

The DFA M recognises the language L if L = $\{w|M \text{ accepts } w\}$

Definition: Regular language

A language is called a regular language if some DFA recognises it

3 Regular Operations

Boolean (set-theoretic):

Union	$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
Intersection	$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
Difference	$A \backslash B = \{x \mid x \in A \text{ or } x \notin B\}$
Complement	$\overline{A} = \Sigma^* \backslash A$

Language theory specific

Concatenation	$A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
Star	$A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and } x_i \in A \text{ for every } i, 1 \leq i \leq k\}$

4 Regular expression

A Regular Expression (RE) R defines a regular language L(R). We shall eventually prove that $RE \equiv DFA$ (i.e. REs define exactly class of the regular languages)

The definition is inductive (recursive), i.e. there are initial RE, and new REs can be obtained from old ones by means of Regular Operations

Definition: R is a Regular expression over the alphabet Σ if R is

1. a for some $a \in \Sigma$
2. ϵ
3. $\varnothing$
4. $(R_1 \cup R_2)$, where R_1 and R_2 are REs

5. $(R_1 \circ R_2)$, where R_1 and R_2 are REs, or
6. (R_1^*), where R_1 is an RE

Note that by convention the concatenation symbol may be omitted, i.e. R_1R_2 means $R_1 \circ R_2$. Parentheses may also be omitted, bearing in mind the precedence order

5 Combining Automata

Given L_1 recognised by $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and L_2 recognised by $M_1 = (Q_2, \Sigma, \delta_2, q_2, F_2)$; want to combine M_1 and M_2 into a new automaton M that would recognise $L_1 \cup L_2$

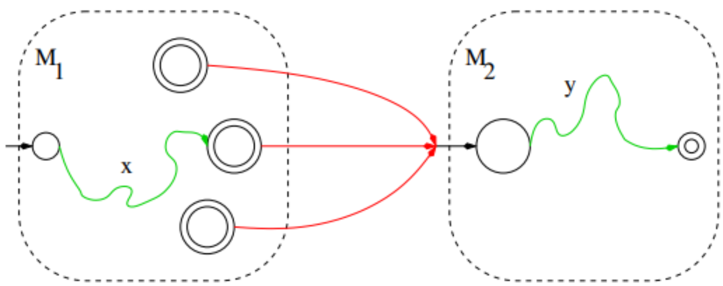
Naive idea: Simulate first M_1 on the input and then simulate M_2 on the same input; accept if either M_1 or M_2 or both accept. This does not work as after M_1 has run on the input, the input is exhausted and there is no way to "rewind" it in order to run M_2

The solution is to run M_1 and M_2 on the input in parallel

6 Concatenation

$w \in L_1 \circ L_2$ only if there are words x and y such that $w = xy, x \in L_1$ and $y \in L_2$

We need to run M_1 on a prefix of w and then to run M_2 on the rest. To find the break point we guess it (non deterministically)



The result is not a DFA because of the red transitions

7 Non-deterministic Finite Automaton (NFA)

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

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathscr{P}(Q)$ is the transition function
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states

7.1 Computation of a NFA

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA and $w = w_1w_2...w_n$ be a word over $\Sigma \cup \{\epsilon\}$
Important remark: ϵ's can be freely added inside the actual word $\in \Sigma^*$

M accepts w if there exists a sequence of states $r_0, r_1, r_2, ..., r_n$ satisfying the following conditions:

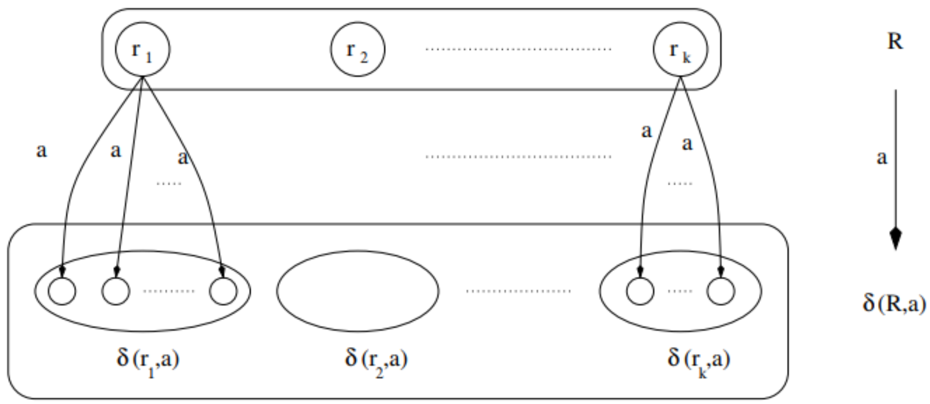
1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for every $0 \leqslant i \leqslant n-1$ and

3. $r_n \in F$

DFA - There is a unique computation (path). It either accepts or rejects
NFA - There are, in general, many computational paths. The NFA accepts if at least one computation accepts. The NFA rejects only if all the computations reject.

8 Converting a NFA into a DFA

At any time there are a number of possible states the NFA computation can be into. As the DFA needs to keep track of all of these, the DFA's states will correspond to subsets of the NFA's-state set. The transition function of the DFA can then be defined as shown below:



Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognising some language L (assume that L has no ϵ transitions). We shall construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognises the same language L as follows

1. $Q' = \mathscr{P}(Q)$
2. For $R \in Q'$ and $a \in \Sigma$ let
$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\}$$
Equivalently
$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' | \exists r \in R r \in F\}$

Let us take care of ϵ-transitions. For $R \subseteq Q$ (same as $R \in Q'$) let

$$E(R) = \{q | q \text{ is reachable from } R \text{ through a number of } \varepsilon \text{ -transitions. }\}$$

Modify the construction as follows

1. $Q' = \mathscr{P}(Q)$
2. For $R \in Q'$ and $a \in \Sigma$ let
$$\delta'(R, a) = \{q \in Q | q \in E(\delta(r, a)) \text{ for some } r \in R\}$$
Equivalently
$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$$
3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q' | \exists r \in R r \in F\}$

9 Closure under regular operations

Union

$$A \cup B = \{x | x \in A \text{ or } x \in B\}$$

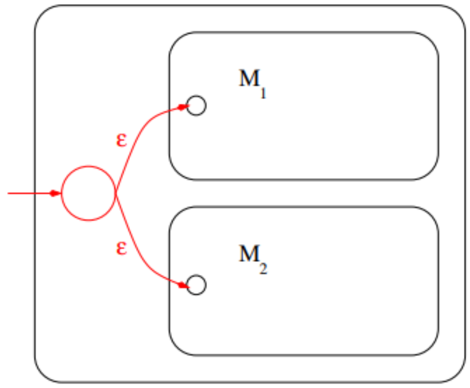


Figure 6: $M_1 \cup M_2$

Concatenation

$$A \circ B = \{xy | x \in A \text{ and } y \in B\}$$

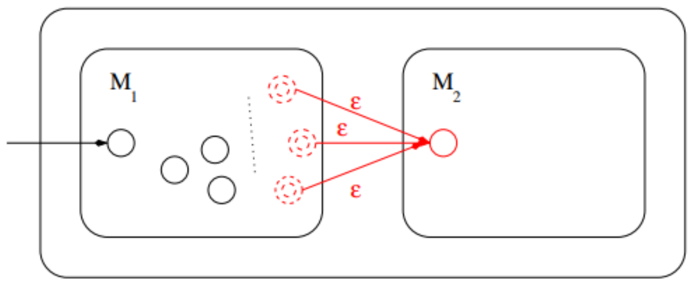


Figure 7: $M_1 \circ M_2$

Star

$$A^* = \{x_1 x_2 \dots x_k | k \geq 0 \text{ and } x_i \in A \text{ for every } i, 1 \leq i \leq k\}$$

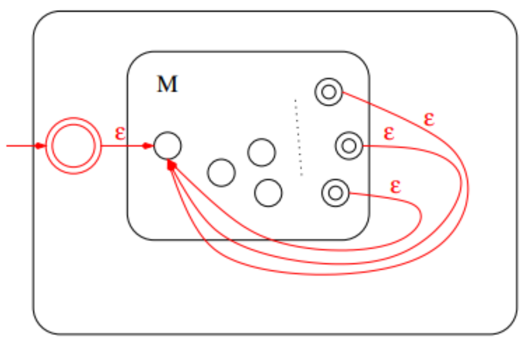


Figure 8: M^*

10 Equivalence of FA and Regular Expressions

Theorem - A language is regular iff some regular expression describes it
Proof
$\Leftarrow$ Given a RE R, construct a NFA that recognises L(R) - this can be done by structural induction

Base case - All initial REs can be implemented by NFAs

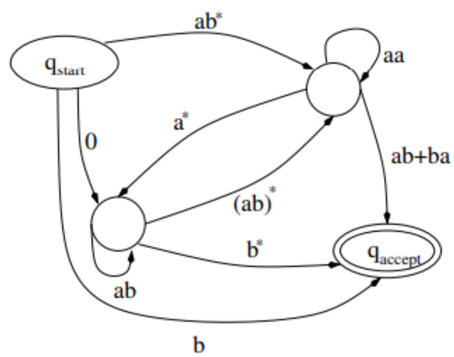
Inductive step - We have already proven that the class of regular languages is closed under regular operations

⇒ Given a DFA M, construct a RE that recognises L(M)

11 Generalised Non-deterministic Finite Automaton (GNFA)

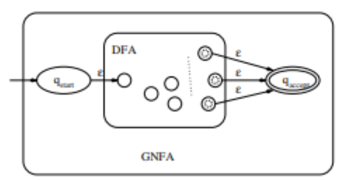
1. A single start state with no incoming edges
2. A single accept state with no outgoing edges
3. Every transition is labelled with a regular expression

-

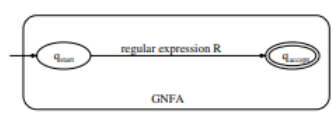


12 DFA into RE

1. Convert the given n-state DFA(NFA) into an equivalent n+2 state GNFA



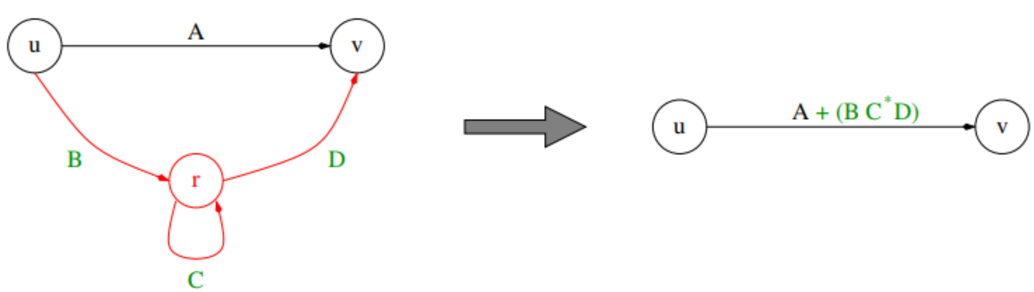
2. Eliminate all internal states of the GNFA one by one, while preserving the language recognised by the automaton



3. We are done, as $L(R) = L(DFA)$

13 Internal state elimination

Pick an internal state r for elimination
For every two states u and v $u \neq r \neq v$, do the following



14 Non-regular languages

How do we prove that a given language N is not regular?

- Can't be recognised by DFA: assume that there is a DFA that recognises N and then use an adversary argument against that DFA in order to get a contradiction

15 Pumping Lemma

For every regular language L, there is a number p (called "pumping length" of L) such that every word $w \in L$ of length at least p can be divided into three parts $w = xyz$, satisfying the following conditions:

1. $xy_iz \in L$ for every $i \in \mathbb{N}$ (in particular $xz \in L$ as one can take $i = 0$)
2. y is a non empty string
3. The length of the string xy is not greater than p

Proof Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognises L, and let us set the pumping length p to be the number of states in M: $p = |Q|$
Consider the computation of M on the word $w = w_1w_2...2_n$ (where $n \leqslant p$)

$$q_0 = r_0 \to^{w_1} \to r_1 \to r_2 \to r_2 \to \cdots\cdots\cdots r_{n-1} \to r_n \in F$$

Let i be the first moment in time where we see a repetition in the states $r_0, r_1, r_2, ..., r_n$ (such an i exists by the pigeon hole principle as $n + 1 > p$) Then we can set $x = w_1w_2...w_j$, $y = w_{j+1}..w_i$ and $z = w_{i+1}...w_n$

It is now straightforward to check that all three conditions are met

16 DFA Minimisation

An algorithm that "minimises" any given DFA, i.e. finds an equivalent DFA with the minimal number of states. It answers at once the following important questions

1. What is the smallest (in terms of number of states) DFA that recognises the same language?
2. Given two DFAs, do they recognise the same language

16.1 Outline of the algorithm

1. Remove all the states that are not reachable from the start state
2. Contract a set of states that are equivalent into a single state

Two states, s and t, are equivalent if any word w, which is accepted when starting from s, is accepted when starting from t and vice versa.

16.2 Equivalent and Distinguishable states

Extended Transition Function: Let a DFA $M = (Q, \Sigma, \delta, q_0, F)$ be given and let $\hat{\delta} : Q \times \Sigma^8 \to Q$ be defined as

$$\hat{\delta}(s, \varepsilon) = s$$
$$\hat{\delta}(s, w_1w_2 \dots w_n) = \hat{\delta}(\delta(s, w_1), w_2 \dots w_n)$$

for every state s and every word $w_1w_2...w_n$
Equivalent states: Two states s and t are equivalent if

$$\left\{w \in \Sigma^* | \hat{\boldsymbol{\delta}}(s, w) \in F\right\} = \left\{w \in \Sigma^* | \hat{\boldsymbol{\delta}}(t, w) \in F\right\}$$

Otherwise, s and t are distinguishable, i.e. there is a witness word u such that either $\hat{\delta}(s, u) \in F, \hat{\delta}(t, u) \notin F$ or $\hat{\delta}(s, u) \notin F, \hat{\delta}(t, u) \in F$