Introduction

A finite automaton is a formal machine that considers discrete input and output. The machine can take any configuration among a given finite set. Those configurations are usually referred to as *states*. Thus, the current state of the machine summarizes all the previous information processed and it is necessary to determine the future behaviour of the machine.

Leaving aside the theoretical interest of finite automata (they model a well-known and interesting family of languages in the Chomsky hierarchy, the family of *regular languages*), there are important reasons to study finite automata. Many widely-used algorithms are based on finite automata. For instance, the *lexical analysis* carried out by any compiler is usually based on the simulation of a finite automaton. In this process, the symbols of a source code are analyzed to detect those strings that correspond with identifiers, constants, or reserved words as well. To do so, during the process, it is enough to consider a finite amount of information.

Automata Theory is also used to design efficient string processors. An example of this is the use of finite automata to detect the positions where a certain pattern can be found in a text x. This problem is known as *String Matching* o *Pattern Matching* and will be studied in Practices three and four.

This Practice is focused to study the representation of three models of finite automata as well as to the implementation of some operations over those models.

Representation of finite automata

A finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ will be represented as a list:

$$aut = \{st, alf, trans, ini, fin\}$$

where:

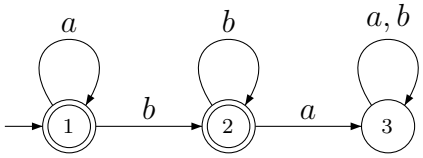
- st is a list that contains the identifiers (integers, symbols, lists, etc.) of the *set of states* (Q)
- alf is a list with the *alphabet* symbols (Σ)
- $trans$ a list with the representation of the *transition function* of the automaton (δ).

  Each transition of the automaton is represented as a list:

  $$\{OriginState, Symbol\ or\ \{\}, DestState\}$$

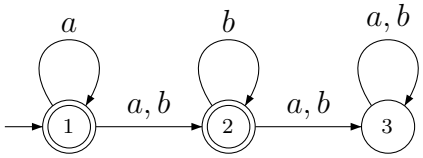
- ini is the identifier of the *initial state* (q_0)
- fin is a list with the identifiers of the *final states* (F)

Examples



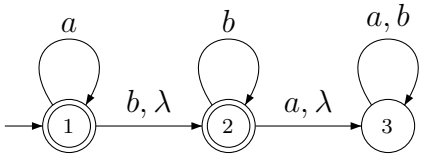
its *Mathematica* representation:

$$A = \{\{1, 2, 3\}, \{a, b\}, \{\{1, a, 1\}, \{1, b, 2\}, \{2, a, 3\}, \{2, b, 2\}, \{3, a, 3\}, \{3, b, 3\}\}, 1, \{1, 2\}\}$$



its *Mathematica* representation:

$$A = \{\{1, 2, 3\}, \{a, b\},$$
$$\{\{1, a, 1\}, \{1, a, 2\}, \{1, b, 2\}, \{2, a, 3\}, \{2, b, 3\}, \{2, b, 2\}, \{3, a, 3\}, \{3, b, 3\}\},$$
$$1, \{1, 2\}\}$$



its *Mathematica* representation:

$$A = \{\{1,2,3\},\{a,b\},$$
$$\{\{1,a,1\},\{1,\{\},2\},\{1,b,2\},\{2,a,3\},\{2,\{\},3\},\{2,b,2\},\{3,a,3\},\{3,b,3\}\},$$
$$1,\{1,2\}\}$$

Exercises

Exercise 1

Design a Mathematica module that, on input of a FA A, returns whether A is deterministic or not.

Exercise 2

Implement a Mathematica module that, on input of a DFA A, returns whether A is complete or not.

Exercise 3

Implement a Mathematica module that, on input of a DFA A, returns an automaton equivalent to A without dead states (those for which there is no path to a terminal state).

Exercise 4

For any given automaton A (deterministic or not), the automaton is *codeterministic* if it has only one final state and there exist no pair of transitions (q, a, p) and (r, a, p) (two transitions that reach the same state with the same symbol). Implement a Mathematica module that, on input of a DFA A, returns whether A is codeterministic or not.

Exercise 5

Implement a Mathematica module that, on input of a DFA A and a string x, returns if x is in $L(A)$ or not

Exercise 6

Design a Mathematica module that, on input of a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and a string x, outputs another automaton $A' = (Q, \Sigma, \delta, q_0, F')$ such that $F' = \{q \in Q : \delta(q, x) \in F\}$.

Exercise 7

Design a Mathematica module that, on input of a complete DFA A and a string x, returns the most-visited state in the the analysis of x.

Exercise 8

Design a Mathematica module that, on input of a complete DFA A and two strings x and y, returns the states that are visited in the analysis of both strings.

Exercise 9

Design a Mathematica module that, on input of a NFA $A = (Q, \Sigma, \delta, q_0, F)$, outputs another automaton $A' = (Q, \Sigma, \delta, q_0, F')$ such that $F' = \{q \in Q : \forall a \in \Sigma,\ |\delta(q, a)| \leq 1\}$ (note that the new set of final states contains the *deterministic* states).

Exercise 10

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a NFA. A state $q \in Q$ *represents its alphabet* if there are transtitions from q with every symbol in Σ. Implement a Mathematica module that, on input of a NFA A, returns $True$ if there is at least one state that represents its alphabet and $False$ otherwise.

Exercise 11

Design a Mathematica module that, on input of a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and two states p and q, outputs another automaton where each reference to p is substituted with a reference to state q.

Exercise 12

Implement a Mathematica module that, on input of a λ-FA A and a state p, returns the $\lambda - closure$ of the state.