ICT600 Computational Mathematics

Finite Automata and Regular Expression

Composed by Nirattaya Khamsemanan, Ph.D

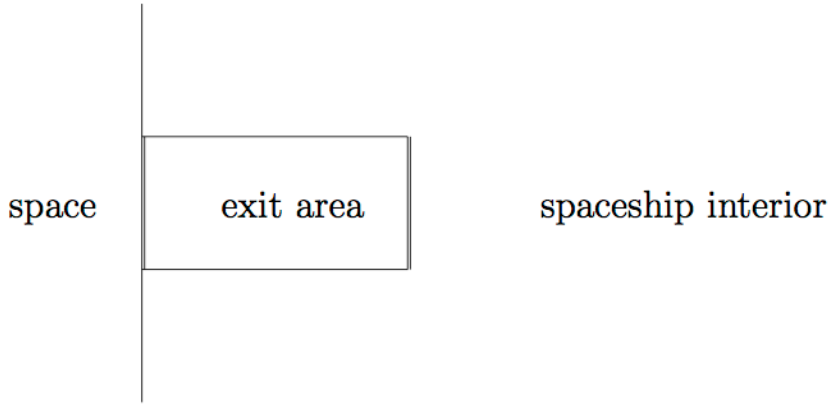
Disclaimer: this partial note is my attempt to help you learn the material in this course. Things in here might not be in the real exam and vice versa. Don't use this as your main study. There might be some typos and/or mistakes.

.-**-..-**-..-**-..-**-..-**-.~ ♡ Have Fun!♡ ∽.-**-..-**-..-**-..-**-..-**-.

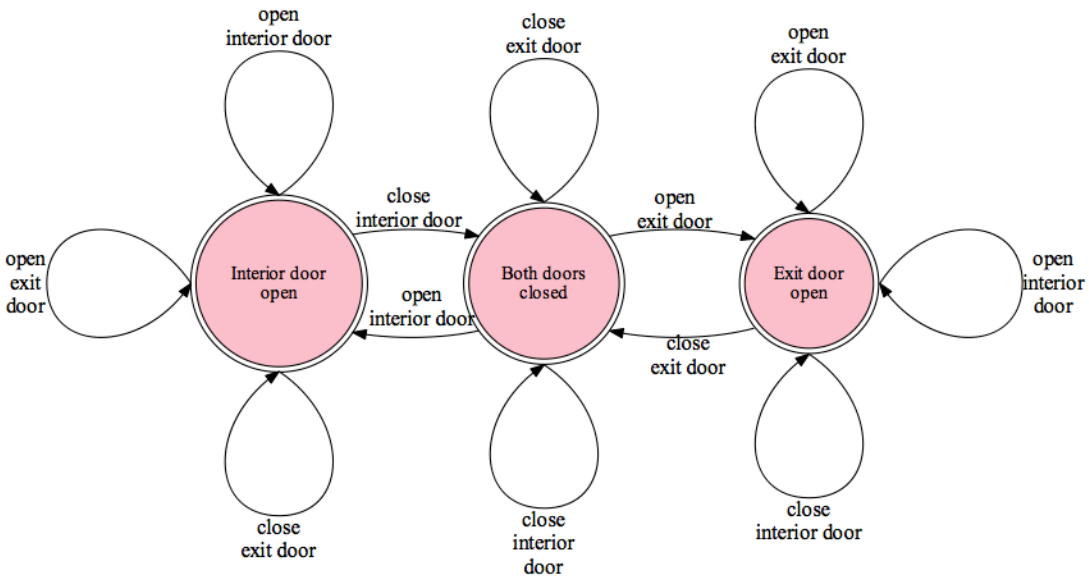
Finite automata (FA) are, in a snap short, very simple computational devices or processes with limited amount of memory. The input is read a symbol at a time. The device then performs a constant amount of processing as a symbol(alphabet) is read. There are two types of FA; Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA).

Now you may wonder whether such simple devices could be of any use. In fact, finite automata are everywhere, from coined washing machines in a laundromat to soft drink dispenser vending machines on the street to the automatic door in the spaceship.

Let's take a look at the automatic doors in a spaceship. To exit from the interior of the spaceship to the space, an astronaut must pass through two doors, interior door; one from interior door to exit area and exit door; one from exit area read to the space



The critical rule is that at most one of the doors may be open at any given time, otherwise all the air in the spaceship could vent out into the vacuum of space. From this rule, there are only 3 possible situations or "states"; both doors are closed, exit door is open but interior is closed and interior is open when the exit door is closed. Let's assume that the controller can receive one request at the time. There are four possible requests that an astronaut can have; open or close exit door and open or close interior door. The following diagram shows how the controller works.



1 Deterministic Finite Automata (DFA)

Definition 1. A deterministic finite automaton is a 5-tuple $M = (Q, \sum, \delta, q_0, F)$, where

1. Q is a finite set called the states,
2. $\sum$ is a finite set called the alphabets,
3. $\delta : Q \times \sum \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subset Q$ is the set of final states.

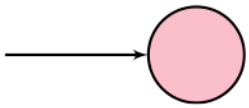
The DFA machine can exist in only one state at any give time. The input of DFA is a string $w \in \sum^*$. Once w is received, the device will start at the start state q_0. Then the device will perform according to the input alphabet in the string w. If after all alphabets in w are consumed, the current state is one of the final states (F) then the device accept w. Otherwise w is rejected.

Definition 2. If A is the set of all strings that machine M accepts, we say that A is the language of machine M and write $L(M) = A$. We say that M recognizes A or M accepts A.

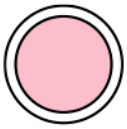
Definition 3. If there exists a DFA M that recognizes A, then A is a DFA-recognizable.

Conventional symbols

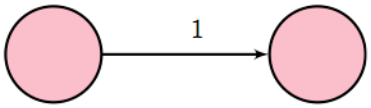
Start state The diagram below shows the start state of a DFA.



Final state The diagram below shows the final state of a DFA.



Transition The diagram below shows that 1 is a transition from a state to another state.



Example 1. *Let M_1 be a DFA defined formally by $M_1 = (Q, \sum, \delta, q_1, F)$, where*

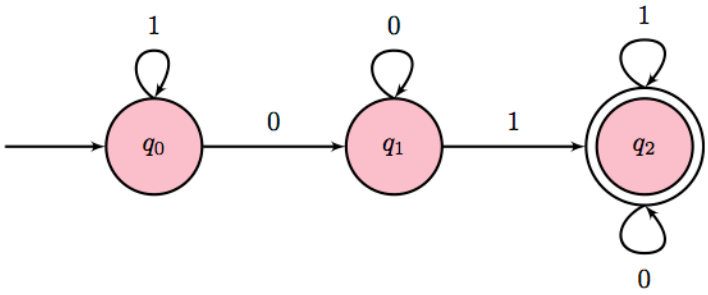
1. $Q = \{q_1, q_2, q_3\}$
2. $\sum = \{0, 1\}$
3. $\delta : Q \times \sum \to Q$ *is described as*

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

4. $q_1 \in Q$ *is the start state, and*
5. $F = \{q_2\}$ *is the set of final states.*

Draw a diagram representing M_1

Example 2. *Define a DFA M_2 by the following diagram.*



1. Let $w_1 = 11001$ and $w_2 = 11010$. Does M_2 accept w_1 or w_2?

2. Write M_2 in term of the 5-tuple formal definition as in Definition 1

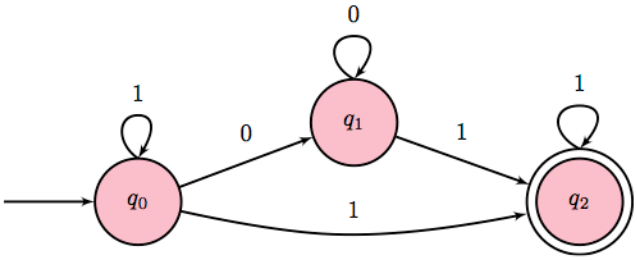
2 Non-deterministic Finite Automata (NFA)

Definition 4. A deterministic finite automaton is a 5-tuple $N = (Q, \sum, \delta, q_0, F)$, where

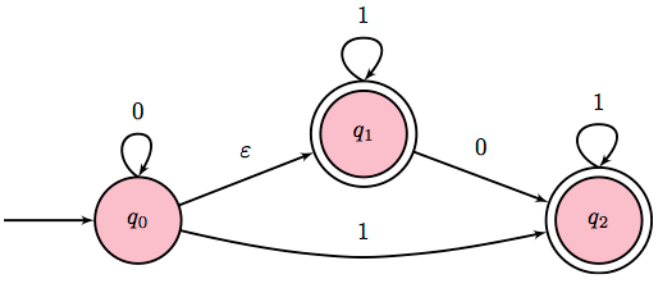
1. Q is a finite set called the states,
2. $\sum$ is a finite set called the alphabets,
3. $\delta : Q \times \sum \cup \{\varepsilon\} \to 2^Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subset Q$ is the set of final states.

An NFA is an automaton that the device can exist in multiple states at the same time. Hence, a DFA is, by definition, an NFA as well.

It allows one or more outgoing arrows under the same alphabet. In an NFA below, with a transition 1 at the start state, we can be either in q_0 or q_1



An NFA might also have ε transition. With a ε, we can move from one state to others without having any alphabets. Without lost of generality, our NFAs will follow the arrow ε *after* each input symbol is read. In an NFA below, with a transition 0 at the start state, we be in q_0 or q_2.



In an NFA, there are more than one choices for the device to choose under the same specific alphabet. This means that given a string w, there might be more than one paths to follow. Some of them may fail. Some may not. In order to accept it suffices to fine one that get to final state.

Definition 5. An Non-Deterministic Finite Automaton accepts an input string w if there exists a path following transitions under the alphabets of w consecutively, that leads to an accepting state

Definition 6. If there exists a NFA N that recognizes a language A, then A is a NFA-recognizable.

Definition 7. A language is called a regular language if some finite automation recognizes it.

Example 3. *Let N_1 be a NFA defined formally by $N_1 = (Q, \sum, \delta, q_1, F)$, where*

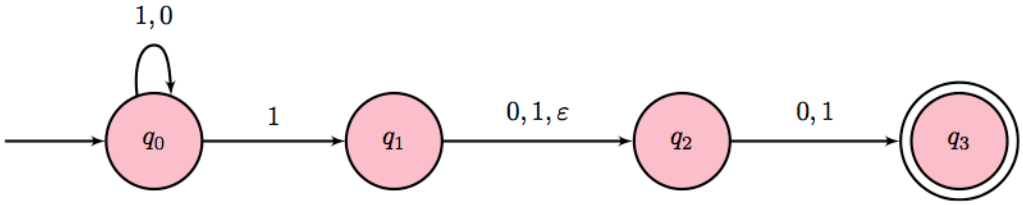
1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\sum = \{0, 1\}$
3. $\delta : Q \times \sum \cup \{\varepsilon\} \rightarrow 2^Q$ *is described as*

δ	*0*	*1*	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. $q_1 \in Q$ *is the start state, and*
5. $F = \{q_4\}$ *is the set of final states.*

Draw a diagram representing N_1

Example 4. *Define a NFA N_2 by the following diagram.*



1. Let $w = 110$. List all possible path for w. Does N_2 accept w?

2. Write N_2 in term of the 5-tuple formal definition as in Definition 4

3 Deterministic finite automata versus Non-deterministic finite automata

DFA	NFA
Each transition leads to one state.	A transition could lead to a subset of states.
For each state, transition on all possible alphabets should be defined.	For each state, not all alphabets necessarily have to be defined in the transition function.
Accepts input if the last state is in F.	Accepts input if one of the last states is in F.
Sometimes harder to construct because of the number of states.	Generally easier than a DFA to construct.
Practical implementation is feasible.	Practical implementation has to be deterministic (must be converted to a DFA).

In fact, DFAs and NFAs are equivalent. But how?

Since DFAs are special cases of NFAs, NFAs recognize at least the DFA-recognizable (regular) languages. Now what is left for us is to convert an NFA to a DFA. Here is a rough sketch on how to do that.

Converting NFAs to DFAs

Given an NFA $N = \{Q_N, \sum, \delta_N, q_N, F_N\}$, we are going to create a DFA $M = \{Q_M, \sum, \delta_M, q_M, F_M\}$ such that if a language is recognized by N, then M also recognize it.

Define $E(q) =$ set of states reachable from q using zero or more ε including q itself.

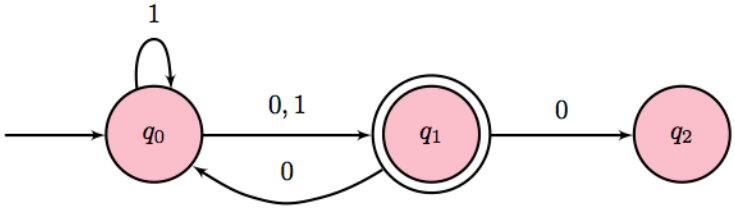
1. $Q_M = 2^{Q_N}$; power set of Q_N
2. $q_M = E(q_N)$
3. $\sum = \sum$; same set of alphabets.
4. $\delta_M(S, a) = \cup_{r \in S} E\big(\delta_N(r, a)\big)$, for $S \subset Q_N$ and $a \in \sum$.

Starting from states in S, $\delta_M(S, a)$ gives all states that could be reached after an alphabet a and possibly ε transitions.

5. $F_M = \{S \subset Q_N | S \cap F_N \neq \emptyset\}$
   Accepting states of M are the sets that contain an accepting state of N

Additionally we need to also modify M by removing all the states that have no arrows pointing at them. The removal will have no effects to the performance of the machine.

Example 5. *Convert the following NFA to an equivalent DFA.*



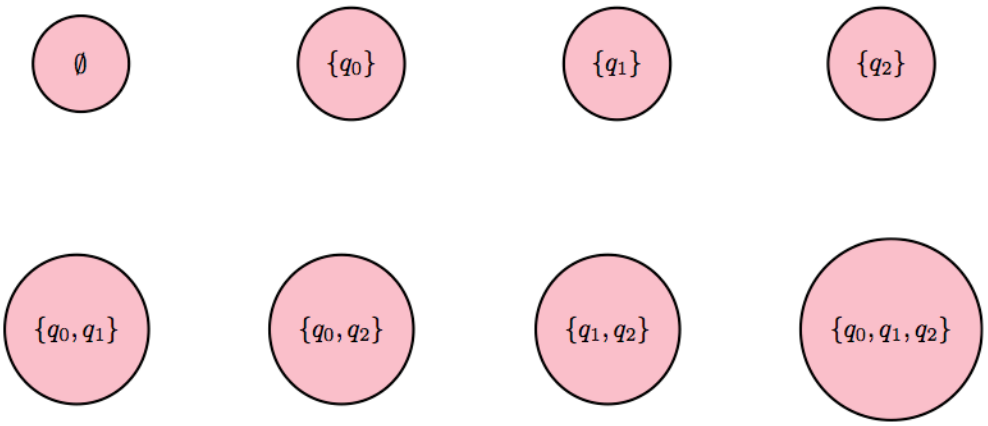
Solution DFA $M = (Q, \sum, \delta, q_0, F)$, where

1. $Q = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$
2. $\sum = \{0, 1\}$
3. δ is described as

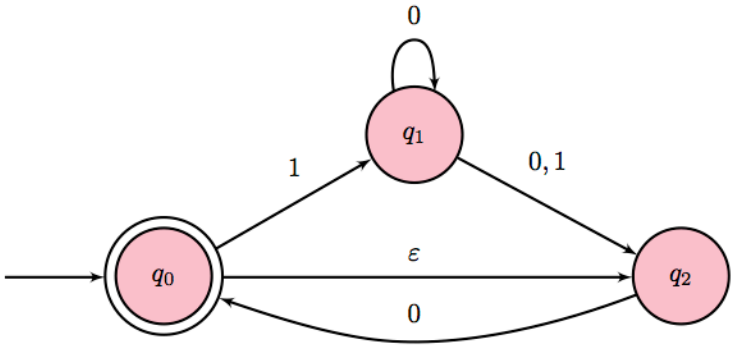
δ	0	1
$\emptyset$		
$\{q_0\}$		
$\{q_1\}$		
$\{q_2\}$		
$\{q_0, q_1\}$		
$\{q_0, q_2\}$		
$\{q_1, q_2\}$		
$\{q_0, q_1, q_2\}$		

4. Start state is

5. Final states are



Example 6. *Convert the following NFA to an equivalent DFA.*



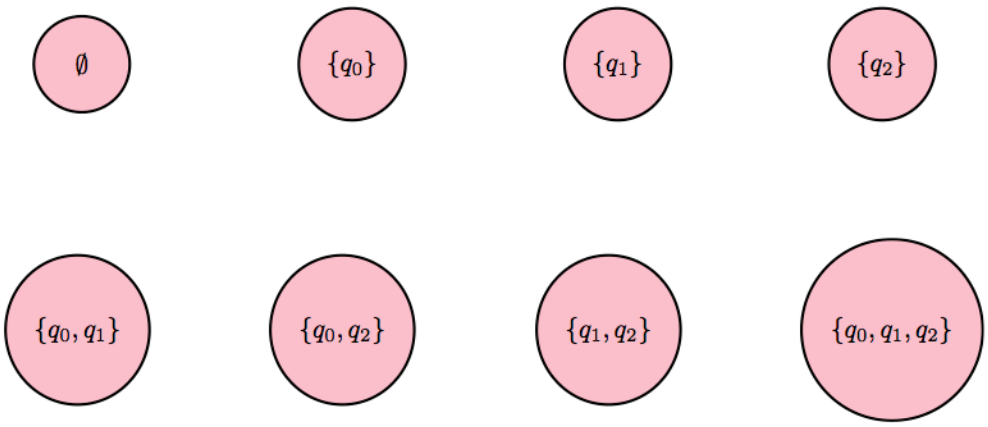
Solution DFA $M = (Q, \sum, \delta, q_0, F)$, where

1. $Q = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$
2. $\sum = \{0, 1\}$
3. δ is described as

δ	0	1
$\emptyset$		
$\{q_0\}$		
$\{q_1\}$		
$\{q_2\}$		
$\{q_0, q_1\}$		
$\{q_0, q_2\}$		
$\{q_1, q_2\}$		
$\{q_0, q_1, q_2\}$		

4. Start state is

5. Final states are



4 Regular Expressions (RE)

Regular expression is an algebraic expression notation for describing some languages, rather than a machine representation. Languages described by regular expression are exactly the regular languages.

Definition 8. R is a regular expression over alphabet $\sum$ if R is one of the following:

1. a for some $a \in \sum$
2. ε
3. $\emptyset$
4. $R_1 \cup R_2$, the union of two regular expressions R_1 and R_2
5. $R_1 \cdot R_2$, the concatenation of two regular expressions R_1 and R_2
6. $(R_1)^*$, the set of all strings over the regular expression R_1.

Remark 1. Note here also that

1. The definition of regular expressions is a recursive definition.
2. The expression ε represents the language containing a single string, the empty string, where $\emptyset$ represent the language that doesn't contain any strings.
3. If parenthesises are omitted, use precedence of operations: * highest, then $\cdot$, then $\cup$.
4. $R^+ = RR^*$
5. $a^*\emptyset = \emptyset$, for any alphabet $a \in \sum$
6. $\emptyset^* = \{\varepsilon\}$

Example 7. *Let* $\sum = \{0, 1\}$. *Describe the following*

1. 0^*10^*

2. $\sum^* 001 \sum^*$

3. $(\sum\sum\sum)^*$

4. $0\sum^* 0 \cup 1\sum^* 1 \cup 0 \cup 1$

5. $(0 \cup \varepsilon)1^*$

5 Regular Expressions and Finite Automata

Theorem 1. *A language is regular if and only if some regular expression describe it.*

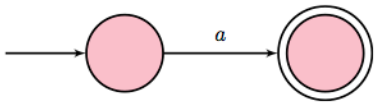
This theorem says that the two statements below are equivalent

- If R is a regular expression, then $L(R)$ is a a regular language which means that there exist some FA that recognizes it. It implies that we can convert an RE to a FA.
- If L is a regular language, then there is a regular expression R with $L = L(R)$. In the other words, we can always convert a NFA to an RE. Unfortunately this direction of conversion is relatively harder and more technical than the other way.

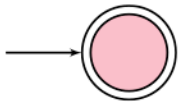
5.1 Converting a regular expression to a finite automaton

In order to convert a regular expression R to an NFA N, we have to consider six cases in definition 8.

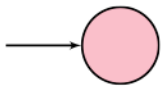
1. If the expression is a symbol a then the language that represents is $\{a\}$ and an automaton that recognizes it is



2. If the expression is a symbol ε then the language that represents is $\{\varepsilon\}$ and an automaton that recognizes it is

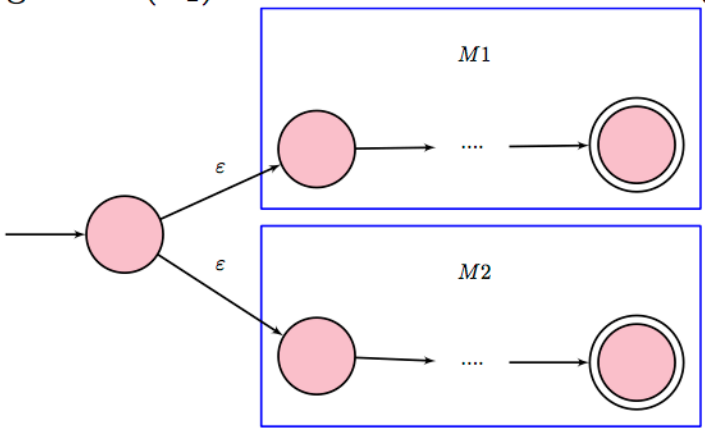


3. If the expression is a symbol $\emptyset$ then the language that represents is $\emptyset$ and an automaton that recognizes it is

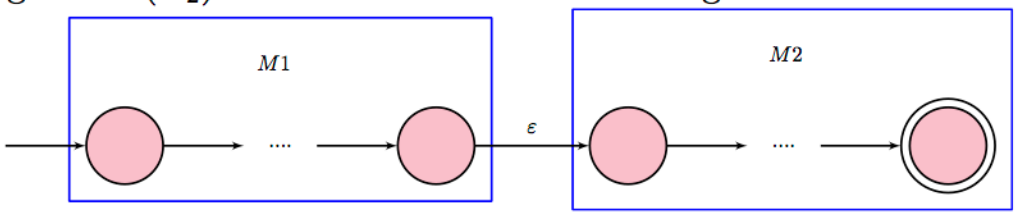


4. If the expression is a symbol $R_1 \cup R_2$ then the language that represents is $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$. If M_1 recognizes $L(R_1)$ and M_2

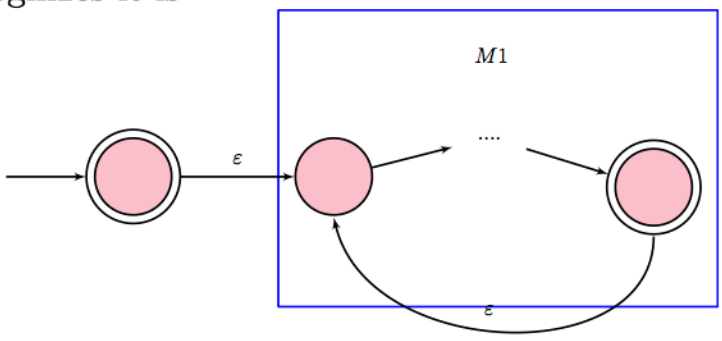
recognizes $L(R_2)$ then an automaton that recognizes it is



5. If the expression is a symbol $R_1 \cdot R_2$ then the language that represents is $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$. If M_1 recognizes $L(R_1)$ and M_2 recognizes $L(R_2)$ then an automaton that recognizes it is



6. If the expression is a symbol R_1 then the language that represents is $L(R_1^*) = \Big(L(R_1)\Big)^*$. If M_1 recognizes $L(R_1)$ then an automaton that recognizes it is



Example 8. *Convert* $(01 \cup 1)^*$ *to a NFA.*

Example 9. *Convert* $(0 \cup 1)^*010$ *to a NFA.*

5.2 Converting a finite automaton to a regular expression

As mentioned before, a conversion from a FN to a RE is relatively harder with more technical process than vice versa. Here is a rough sketch on now to convert a given NFA to a regular expression.

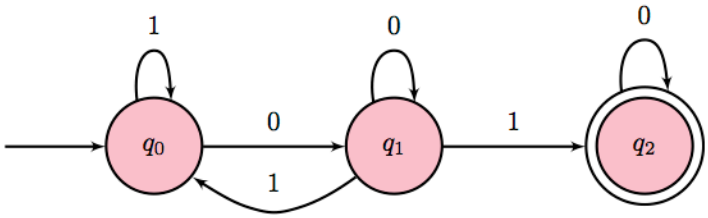
1. Add to new states, a start state s and a unique final state f to the NFA such that

   - A start state s points with ε to the previous start state of NFA.
   - Each accept states of NFA points to a unique final state f with transition ε.
   - All other states are non-accepting.

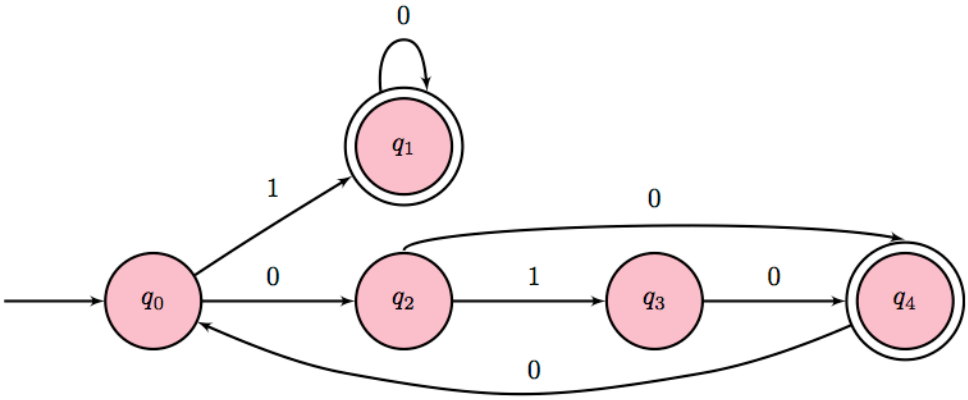
   Note that this new FA is called Generalized NFA or GNFA for short.

2. Repeatedly removed all the other states one by one in any order, replacing labels of arrows with more complicated regular expressions. Beware of all outgoing and incoming arrows of state that we are removing. Consider all possible combinations.

Example 10. *Convert the following NFA to a regular expression*



Example 11. *Convert the following NFA to a regular expression*



References

[1] Michael Sipser, *Introduction to the Theory of Computation, 2nd Edition, Thomson Course Technology*, Thomson Course Technology 2006. ISBN 0-534-95097-3.

[2] Richard Cole, *Theory of Computing. What is Feasible, Infeasible, and Impossible Computationally*, Lecture notes.