Theory of Computation

Notes by Arjun Maneesh Agarwal

based on course by C Aiswarya

This is the first coutse about computablity course, we will see what can be computed and what can't be computed? We will try to define a computer (through models of computation) and try to answer these questions.

Table of Contents

1. Random Quotes

"This course is a difficult one, but if you work hard, it is easy."

2. Introduction

While some people can compute squares, a toddler can't. This would make the core question extreamly philosphical, but as this is a computation course, we will focus on what computers can compute.

Definition: Descision Problems

A problem which given a **valid input** should have an boolean output (yes or no) is called a descision problem.

This definition poses the problem: how do we define a valid input?

Given the problem of determining if the input is prime, then "chennai" or "CMI" is not a valid input. But to restrict our outputs to boolean, we will put a restriction on the inputs that can be given to us; by constraining the language of communication itself!

3. Formal Language Theory

Definition: Languages

A language is a set of words.

Definition: Words

A word is a sequence (of finite length) of letters from a finite alphabet. We tend to use Σ or A to denote alphabets.

Definition: Empty word

A word with no letters is ε.

Definition: Empty Language

A language with no words is $\emptyset$.

Definition: Cannonical Language

Given a descision problem, a cannonical language is the set of all words which have the answer 'yes' or 'true' when given as input to descision problem.

Example

Given the alphabet $\{0, 1, ..., 9\}$, we can define languages such as:

- even numbers
- numbers with at least 5 length
- numbers with 5 different letters.
- number whoose last digit is 5 or 0
- -First and last digit are the same.

Definition

The set of all words given language Σ is Σ^*. Thus, all languages $L \subseteq \Sigma^*$.

Theorem 3.1. *Number of words is $\aleph_0$.*

Proof. We can enumerate the words, provided an ordering on the alphabets.

For example, in the case of $\Sigma = \{0, 1, ..., 9\}$, our ordering would be:

$$\varepsilon, 0, 1, ..., 9, 00, 01, ..., 99, 000, ...$$

■

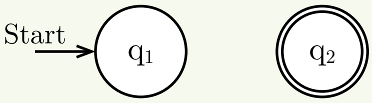
This tells us that we have $2^{\aleph_0}$ or uncountably infinite languages. Thus, by the cannonical language bijection, we can have uncountably infinite languages.

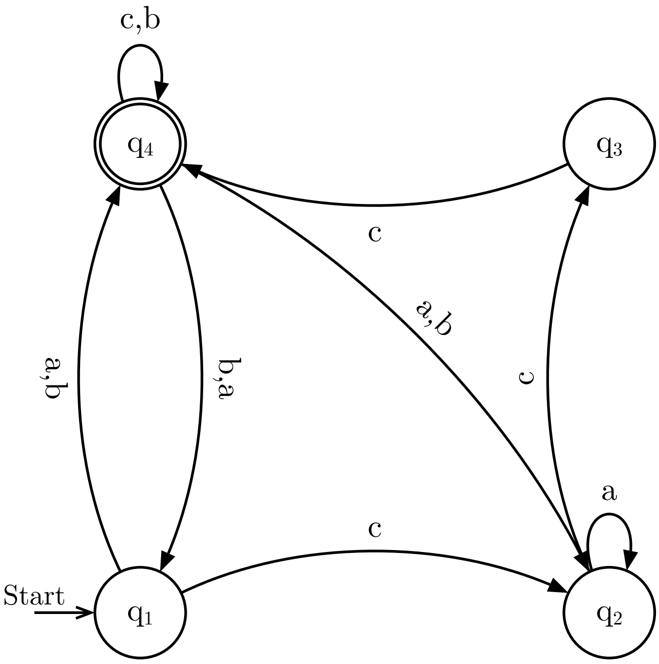
We can thus, not enumerate the languages and need some other way.

4. Automata Theory

Definition: Automata

Automata is a directed graph which can have self loops.

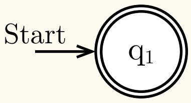




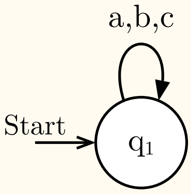
This graph accepts the word "abaccabbbab".

Note that the graph need not have a single path for an letter in the alphabet, it can have none or multiple. If it is none, the word is rejected. If there are more than one paths from some letter, we have a non-deterministic finite automata which means, we have to try all the valid paths and see if it reaches a final state.

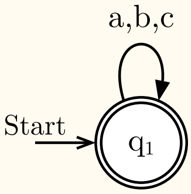
Example



This automaton only accepts the accepts only the empty word.



This automaton accepts only the empty langauage.



This automaton accepts all possible words.

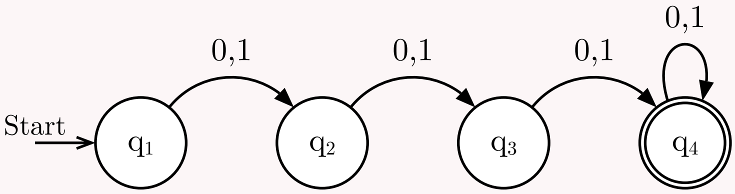
An possible exam problem could be, given a language, make an automaton to check for them.

Exercise

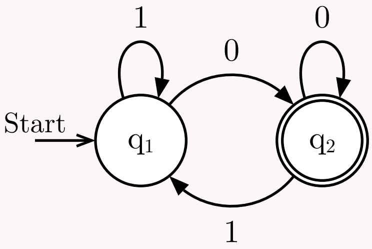
Given: $\Sigma = \{0, 1\}$

1. Language: Set of all words of length ≥ 3
2. Language: Set of all even binary numbers.

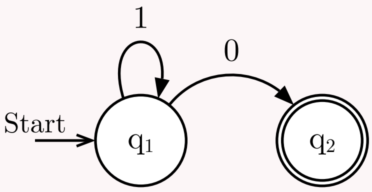
Solution



A Deterministic solution is:



But we can also make a non-deterministic solution:

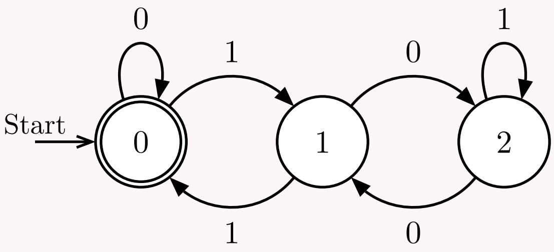


Here the non-determisim turns up as we don't know if we are on the last letter of the word or not.

Exercise

Language such that it contains binary numbers that are divisible by 3.

Solution



Idea : Functional Automata

Instead of denoting auatomata using the diagram, we can use a Functional approach. An automata is $(Q, \Sigma, \Delta, s, F)$ where Q is the set of states, Σ the alphabet, Δ a function of type $Q \times \Sigma \to 2^{\Sigma}$[1], $S \subseteq Q$ is the set of start states and $F \subseteq Q$ is set of final states.

We can think of Q, Σ, s, F as broiler plate and δ as where the real magic happens.

If one looks at the type of δ, we can clearly see some equivalence to `foldr` or `foldl`. Indeed, an automata is can be denoted as function: `automata ::` $[\Sigma] \to \delta \to$

TODO.

Definition: Run

A run of $\mathcal{A}$ on w is:

$$q_0 \overset{a_1}{\to} q_1 \overset{a_2}{\to} q_2 \overset{a_3}{\to} \ldots \overset{a_n}{\to} q_n$$

where $q_0 \in S$, $(q_{i-1}, a_i, q_i) \in \delta$.

[1] In deterministic case, we take $Q \times \Sigma \to \Sigma$.

Definition: Accepting Runs

A run is called accepting if $q_n \in F$.

Definition: Language of Automata

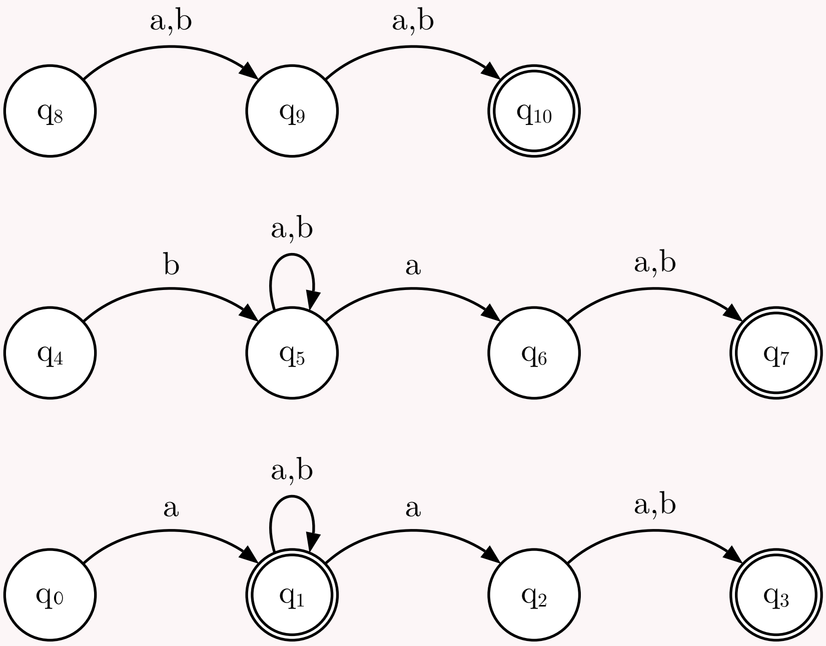
$$L(\mathcal{A}) = \{w \mid \mathcal{A} \text{ has an accpeting run on } w\}$$

Exercise

Given $\Sigma = \{a, b\}$

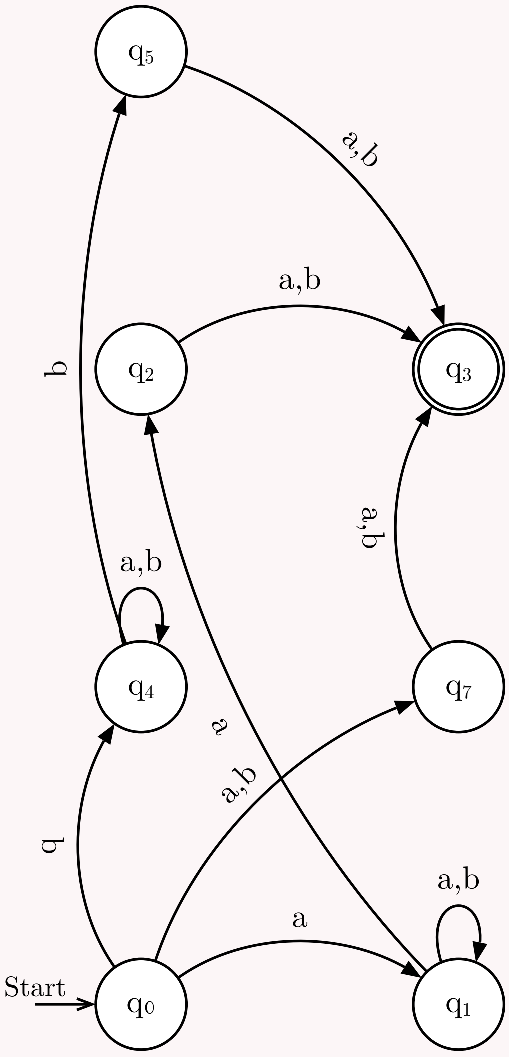
1. First and second last letter are the same. The lower sub-questions are hints.
2. First letter is an a
3. Second last letter is an a
4. 2 and 3 first and second last letters are a.

Solution



[2]

[2] Aish's solution.



[3]

This implies if we have an automata for L_1 and L_2 then we have an automata for $L_1 \cup L_2$ by just keeping both together.

Note that the number of finite state automata are countable. Hence, there are only a countable number of languages we can recognize.

Definition: Recognizable Languages

A language which can be recognized using a finite state non-deterministic automata is called Recognizable Language.

We can see that the set of recognizable langauages is closed under union. What about intersection?

[3] Neel's solution

Definition: Deterministic Finite State Automata

A DFA is an automata is $(Q, \Sigma, \delta, s, F)$ where Q is the set of states, Σ the alphabet, δ a function of type $Q \times \Sigma \to \Sigma$, $S \subseteq Q$ is the set of start states and $F \subseteq Q$ is set of final states.

We can think of Q, Σ, s, F as broiler plate and δ as where the real magic happens.

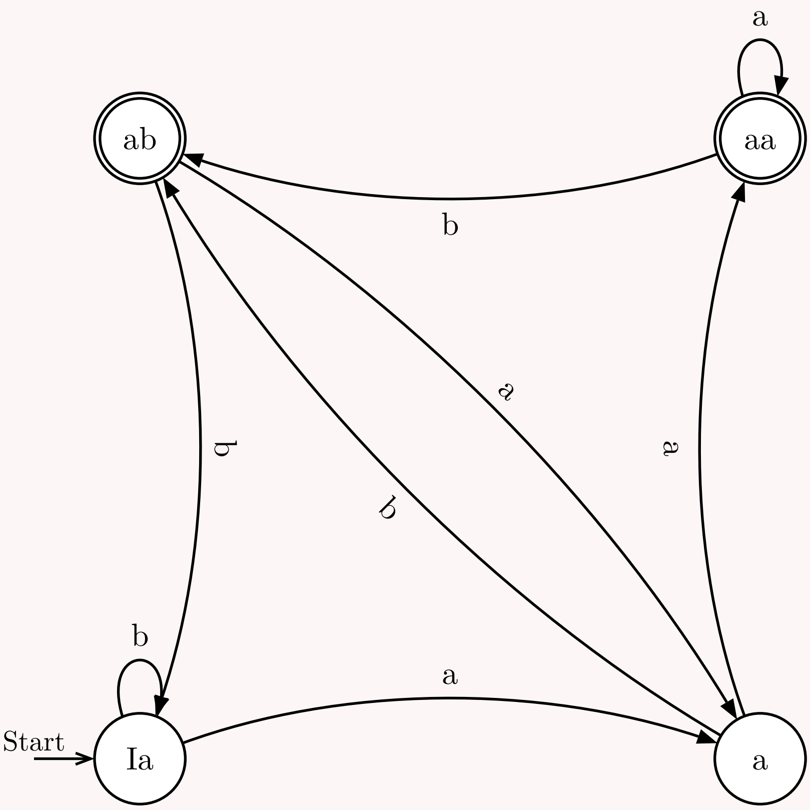
Exercise

Now make a DFA solving the above excercise.

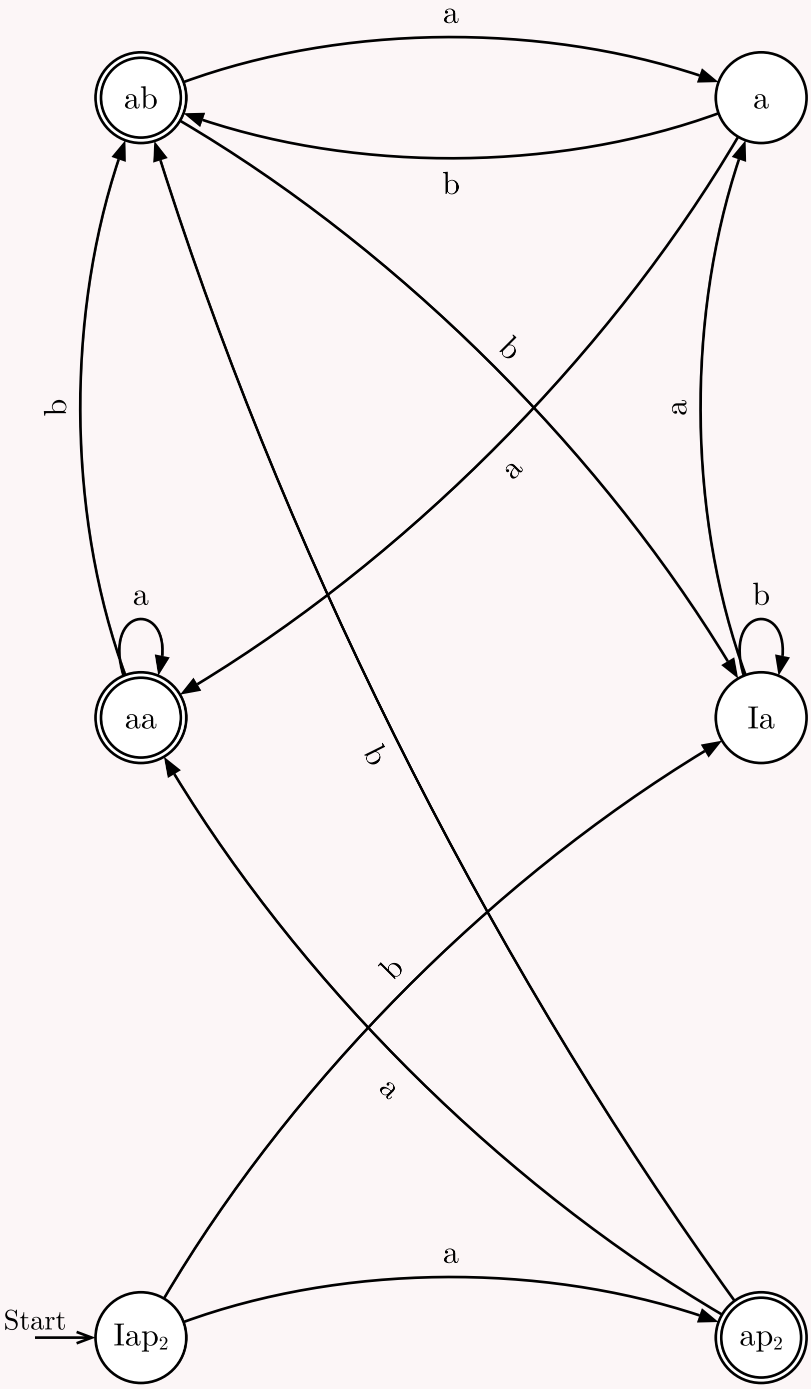
Solution

Once we read the first character, we will branch into two sub-automata. We will show the automata for a (that is the string starts with a).

We will see the side case for the length 2 later. If we assume that the string is of the form `[a:xs]` and `length xs >= 2`, we can make a autmata by simply storing care a two character state.

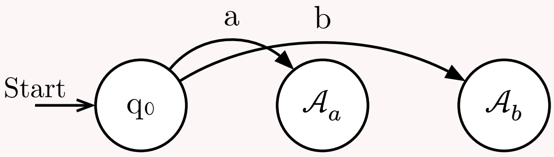


We can accomadate the edge case by simply adding a special `Iap2` denoting the initial case with possiblity of length 2 and move to the full automata once this is ruled out.



We will denote this as $\mathcal{A}_a$ and the analogue for b as $\mathcal{A}_b$.

This makes the final automata:



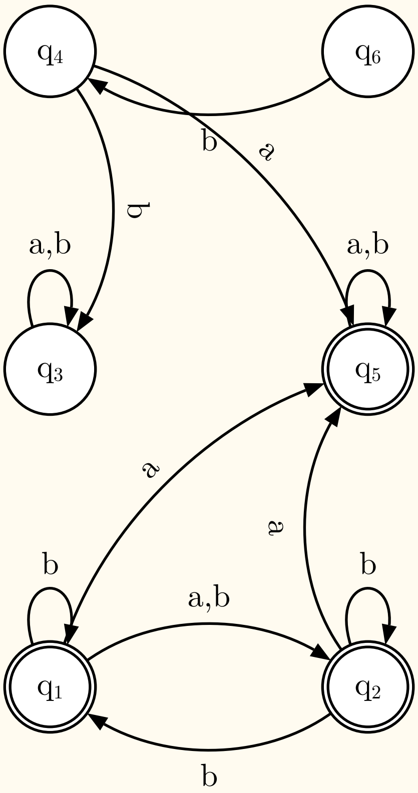
The full code to make the automaton is given below (not rendered for space purposes).

```
  #automaton(
  (
    q0:(Iap2:"a", Ibp2:"b"),

    // A_a branch
    Iap2:(ap2:"a", Ia:"b"),
    ap2:(aa:"a", ab:"b"),
    Ia:(a:"a", Ia:"b"),
    a:(aa:"a", ab:"b"),
    aa:(aa:"a", ab:"b"),
    ab:(a:"a", Ia:"b"),

    // A_b branch (symmetrical to A_a)
    Ibp2:(bp2:"b", Ib:"a"),
    bp2:(bb:"b", ba:"a"),
    Ib:(b:"b", Ib:"a"),
    b:(bb:"b", ba:"a"),
    bb:(bb:"b", ba:"a"),
    ba:(b:"b", Ib:"a")
  ),
  final: ("ap2", "aa", "ab", "bp2", "bb", "ba"),
)
```

4.1. Subset Construction

Example

We will try to convert this to a DFA by keeping track of states we could be in



with initial states q_1 and q_4. We will begin at $\{1, 4\}$ and then move to all states where we could be. That is:

$$\delta(S, a) = \bigcup_{s \in S} \Delta(s, a)$$

and the state space be 2^Q of the NFA. In practice, we don't use all the 2^Q directly and instead build the thing one by one considering letter by letter to avoid talking about subsets which can't ever be created. Remember, in DFA, we have 1 trasition per alphabet.

Algorithm 4.1.

DFA to NFA

```
def delta(a, S, nfaDelta):
ans = []
for s in S:
    k = nfaDelta(a,s)
    if k not in ans:
        push k ans
return ans
def func(alphabet, nfaDelta, S, nfaFinal):
```

DFA to NFA

```
    initial = S
    dfaDelta = []
    final = []
    unVisited = Queue [S]
    checked = []
    while unVisited is not empty:
    state = pop unVisited
    for s in state:
    if state in nfaFinal:
        push s final
        break
        for a in alphabet:
            k = delta(a, S)
            DFA[state][a] = k
            if k in checked: +continue
            else:
                push k unVisited
    return(alphabet, checked, initial, dfaDelta, final)
```

Running this algo, we will get:

TODO. *Will just program this!*

Theorem 4.2. *Our algorithm indeed creates a DFA which accepts the same langauage.*

Proof. We will prove this by induction taking the induction hypothesis to be the stronger statement that if and only if

TODO. *to get from Kozen in func style!*

■

Theorem 4.3. *Given two recognizable languages L_1, L_2 and L_3:*

1. $L_1 \cup L_2$ is recognizable.
2. $L_1 \cap L_2$ is recognizable.
3. $\neg L_1$ and $\neg L_2$ are recognizable.

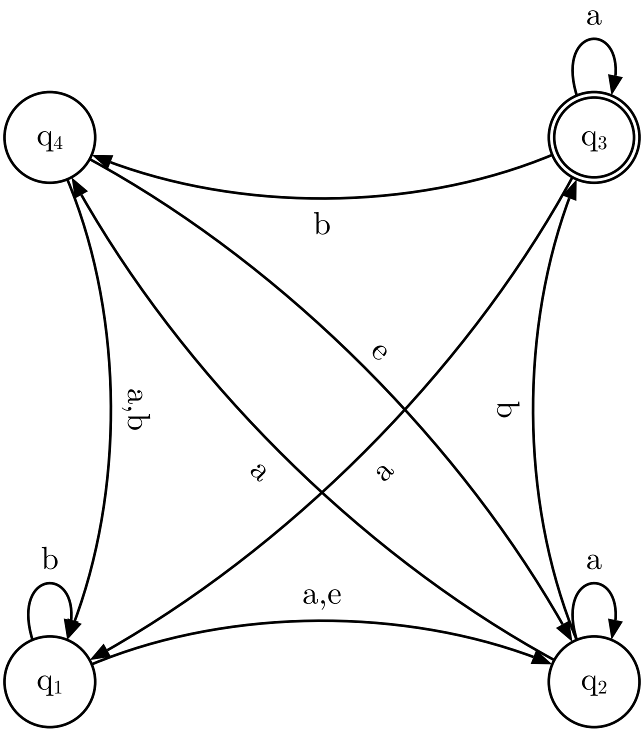
Proof.

TODO.

■

4.2. Epsilon Automata

Consider the automate with ε or e trasitions meaning that they can be taken willy-nilly.



We can make a epsilon free NFA by changing all epsilon paths from source as sources and changing all epsilon paths by scheme $a \xrightarrow{\varepsilon} b \xrightarrow{\#} c \mapsto a \xrightarrow{\#} c$ (forward closure) ...

TODO.

Definition: Concatination

Equavalent to `++` in Haskell.

$\varepsilon \cdot \varepsilon = \varepsilon,$

$u \cdot v = uv$

Theorem 4.4. *DFA = NFA =* ε*-NFA*

Proof. By the above sections! ■

Given:

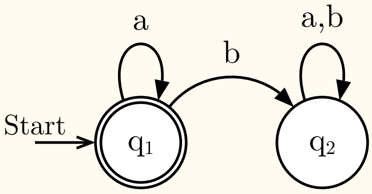
$$\mathcal{A}_1 = (Q^1, \Sigma, \Delta^1, Q_0^1, Q_F^1)$$
$$\mathcal{A}_2 = (Q^2, \Sigma, \Delta^2, Q_0^2, Q_F^2)$$

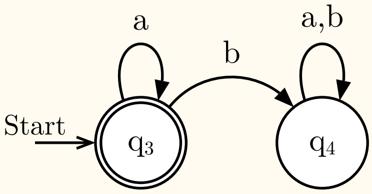
We will try to define $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$

$$\begin{gathered}\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \\ = (Q^1 \times Q^2, \Sigma, \Delta, Q_0^1 \times Q_0^2, Q_F) \\ \text{where } \Delta = \{((q_1, q_2), a, (q_1', q_2')) \mid (q_1, a, q_1') \in \Delta^1, (q_2, a, q_2') \in \Delta^2\} \\ \text{and } Q_F^{\cup} = Q_F^1 \times Q_2 \cup Q^1 \times Q_F^2\end{gathered}$$

This construction fails if the Δ is underdefined, and we will sometimes need to add the trap states.

Example : Using the Construction





We can combine these to get:

Exercise

Concatinate two recognizable langauages.

Solution

Connect the end states of L_1 to all the start states of L_2 using ε. The automata retains the start states of L_1 and end states of L_2.

Formally, if $(Q_1, \Sigma, \Delta_1, S_1, F_1)$ recognizes L_1 and $(Q_2, \Sigma, \Delta_2, S_2, F_2)$ recognizes L_2; then we can recognize $L_1 L_2$ using the automata $(Q, \Sigma, \Delta, S, F)$ where

$$\begin{gathered}Q = Q_1 \uplus Q_2 \\ S = S_1 \\ F = F_2 \\ \Delta = \Delta_1 \uplus \Delta_2 \uplus \uplus \{(q_1, \varepsilon, q_2) \mid q_1 \in F_1, Q_2 \in S_2\}\end{gathered}$$

4.3. Closure Property of Recognizable Languages

We have already seen recognizable langauages are closed under boolean operations.

Definition: Homomorphism

An homomorphism from $\Sigma^* \to \Gamma^*$ is a map with the properties:

- $h(uv) = h(u)h(v)$
- $h(\varepsilon) = \varepsilon$

In practice, to define an homomorphism, we just need to map $\Sigma \to \Gamma^*$ and extend that map. That is, the definition of the homomorphism only needs to be defined on the letters to be complete.

Example

Taking $\Sigma = \{a, b, c\}$ and $\Gamma = \{0, 1\}$; we can have an homomorphism:

- h_1 :

  $a \mapsto 001$
  $b \mapsto 10$
  $c \mapsto 11$

- h_2 :

  $a \mapsto 00$
  $b \mapsto 1$
  $c \mapsto \varepsilon$

- h_3 :

  $a \mapsto 00$
  $b \mapsto 00$
  $c \mapsto 1$

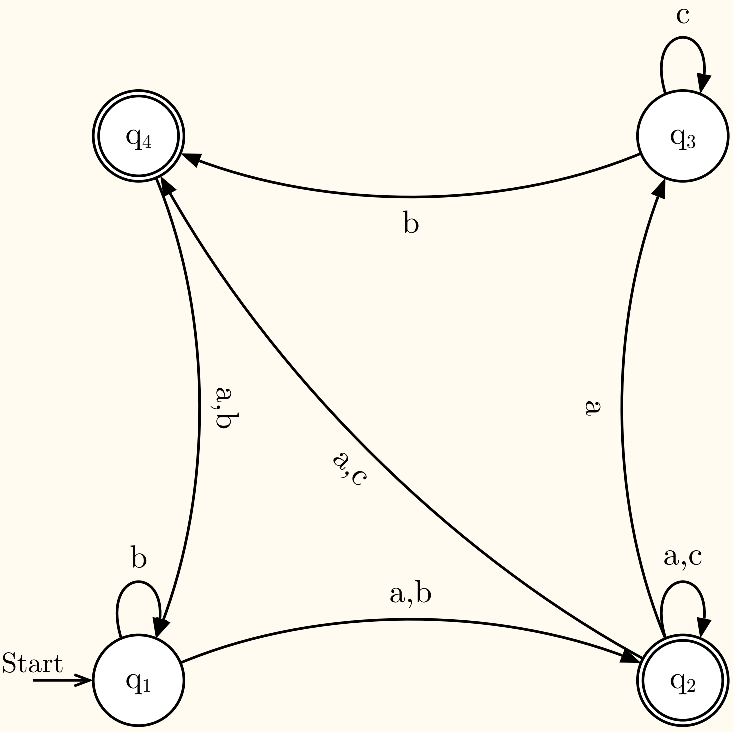
Theorem 4.5. *If $L \subseteq \Sigma^*$ and is recognizable then $h(L) = \{h(w) \mid w \in L\} \in \Gamma^*$ is also recognizable where h is an homomorphism.*

Example

Let h be the map:

$$a \mapsto 001$$
$$b \mapsto 10$$
$$c \mapsto 11$$

Make the map for the homomorphic language given the automata here is:



Proof.

TODO.

■

Theorem 4.6. *If $L \subseteq \Gamma^*$ is recognizable then $h^{-1}(L) = \{w \in \Sigma^* \mid h(w) \in L\} \subseteq \Sigma^*$ is recognizable.*

Claim 4.7. We claim that if $(Q, \Gamma, \Delta, S, F)$ is an automata recognizing L then $h^{-1}(L)$ is recognized by $(Q, \Sigma, \Delta^*, S, F)$ where $\Delta^*(q, x) = \bigcup \text{fold}(\Delta(q, h(x)))$.

Definition: Kleene Star

The Kleene Star is an uniary operation on a language L which returns:

$$\bigcup_{i=0}^{\infty} L^i$$

Note, this is just arbitarry concatinations of L. Also, even if $\varepsilon \notin L$, still $\varepsilon \in L^*$.

Theorem 4.8. *If L is recognizable, L^* is recognizable.*

The idea is to add a new initial (as well as final) state and then link it to all old initial states via epsilon and from all the old final states from epsilon.

Claim 4.9. If L is recognizable by $(Q, \Sigma, \Delta, S, F)$ then L^* is recognizable from $(Q \cup 1_k, \Sigma, \Delta', 1_k, 1_k)$ where $\Delta' = \Delta \cup \{(1, \varepsilon, q) \mid q \in S\} \cup \{(q, \varepsilon, 1) \mid q \in F\}$.

4.4. Rational Language

Definition: Class of Rational Languages

Rational languages is the smallest class of langauages such that

- Every finite language is in the class or equivalently $\emptyset, \{\varepsilon\} \in R$ and $\{k\} \in R$ for all $k \in \Sigma$ are in the class.
- closed under union
- closed under concatinations
- closed under Kleene star

All languages in this set are called Rational langauages.

Exercise

If $\Sigma = \{a, b, c\}$,

(i) is Σ^* rational?
(ii) is language with all second letters a rational?
(iii) is language with words that contain cc rational?
(iv) is langauage with words not containing consecutive c's rational?

Solution

(i) Obviosuly as $\Sigma^* = (\{a\} \cup \{b\} \cup \{c\})^*$.
(ii) Can be reprasented as $\Sigma\{a\}\Sigma^*$.
(iii) Can be reprasented using $\Sigma^*\{cc\}\Sigma^*$.
(iv) $(a+b)^*(c(a+b)(a+b)^*)^*(c+\varepsilon)$[4].

A better solution by Neel was $(c+\varepsilon)((a+b)^+(c+\varepsilon))^*$

Idea : Regex Notation

We can use a regex like notation where $\cup \leftrightarrow +$ and dropping the brakets and dots and $a^+ = aa^*$.

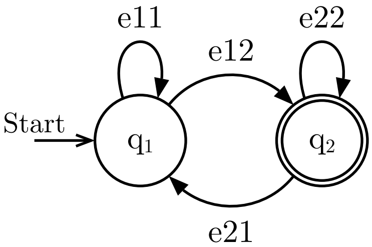
Theorem 4.10. *All Rational Languages are Recognizable*

Proof. We can identify the atoms $a, b, c, \emptyset, \varepsilon$ etc and the operators are also closed under Finite Autmata. ■

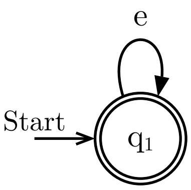
[4] Aish's solution

Kleene's Theorem 4.11. *Every Recognizable language is rational.*

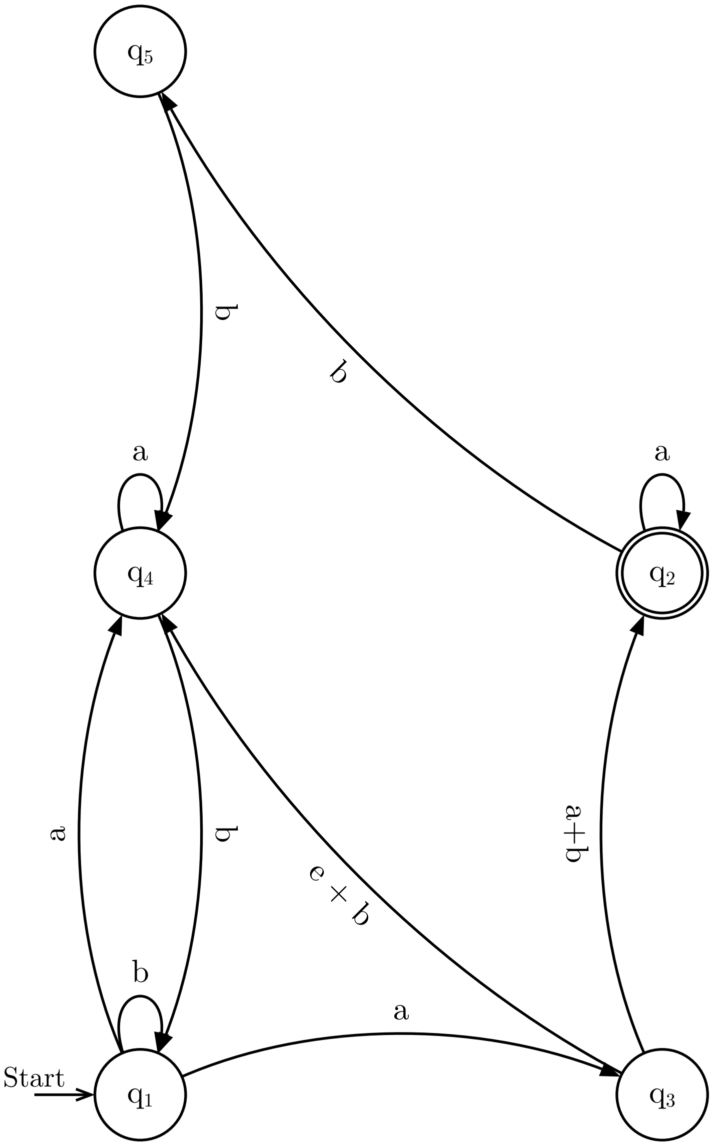
Proof. We claim that:



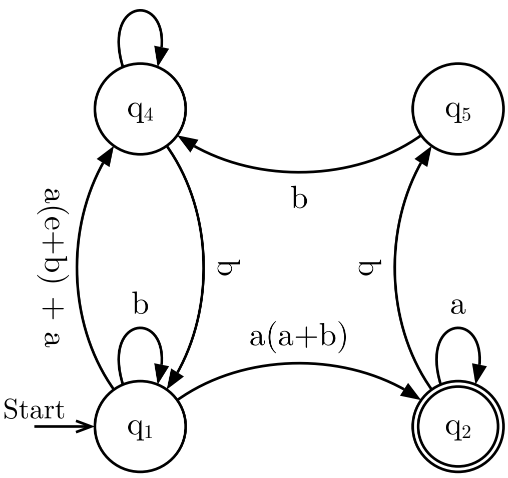
We can see that the rational expression is $(e_{11} + e_{12}e_{22}^*e_{21})^*e_{12}e_{22}^*$.



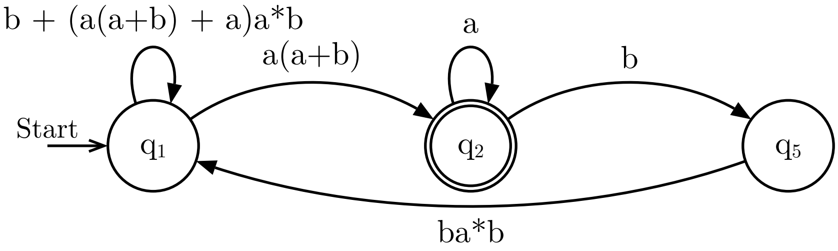
We can get the rational expression as e^*.



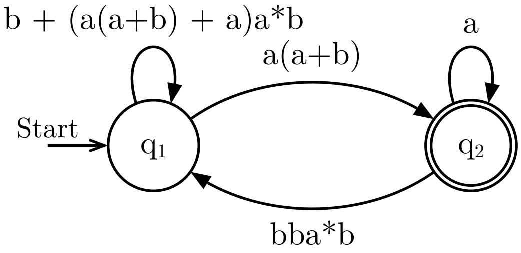
We will now remove a state, say q3.



We will now remove q4.



We can now remove q5.



Using the above case, we can get the rational expression.

■

Theorem 4.12.

$$(e_1 + e_2)^* = e_1^* e_2^*$$

TODO. *2 classes worth of notes*

We have show rational, recognizable and regular langauages are the same things with the former being equal by Kleene's and the latter by Myhill-Nerode.

We also saw closure properties as well as proof techniques for non-recognizablity such as fooling sets and pumping lemma.

We also saw that there is a unique minimal automata.

We can note NFA to DFA conversion takes exponential nodes.

Rational to NFA takes linear nodes.

NFA to Rational takes exponential nodes.

Membership where we are given a word w and a[5]

Does a fiven NFA accept only a finite number of words?

Given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, is $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?

DFA minimization is polynomial in number of nodes and size of alphabets.

[5] I don't trust half these numbers. Aish is dumb dumb.