Seminar : Automata Theory

Lec 1 : Equivalency of Finite Automata and Regular expression

Jinhee Paeng

Aug 22, 2023

Seoul National University

Table of Contents

Table of Contents

Language

Definition (Alphabet)

Alphabet is a finite set of characters. We denote Σ as a alphabet set.

Example. Following sets are examples of the alphabet.

- $\{0, 1\}$
- $\{a, b, c, \cdots, z\}$

Definition (String)

String generated by Σ is a finite length word that uses characters in Σ. We denote the length of the string w as $|w|$. We say a string is an empty string if it is of length zero. We denote the empty string as ϵ.

Language

Definition (Concatenation)

When the two strings x, y are given, we denote a concatenation of as xy. When the two sets of strings X, Y are given, we define a concatenation of sets X, Y as

$$XY = \{xy : x \in X, y \in Y\}.$$

Definition (Kleene product)

For a set of strings S, we denote a set S^* as a Kleene product of the set S :

$$S^* = \bigcup_{k=0}^{\infty} S^k, \quad S^{k+1} = S^k S, \quad S^0 = \{\epsilon\}$$

Remark. Note that the set of all strings generated by the set Σ is Σ^*. For example, when $\Sigma = \{0, 1\}$, then

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \cdots\}.$$

Language

Definition (Language)

- We say a set L is a *Language* if $L \subset \Sigma^*$.
- We call a set of Languages as *Language Class*.

Remark. Since Language is a set of strings, we can define concatenation and Kleene product identically on Languages.

Remark. Definition of *Language* coincides with the general notion of languages such as English, Korean, C, Python.

Table of Contents

Regular Expression RE

Goal. In this section we define the first useful Language class, called *RE*.

Definition (Regular Expression)

We denote a language class *Regular Expression* as RE. Language class RE is defined inductively using following rules.

1. $\emptyset \in RE$: An empty set $\emptyset$ is regular expression.
2. $\epsilon \in RE$: A singleton set $\epsilon = \{\epsilon\}$ is regular expression.
3. $a \in \Sigma \Rightarrow a \in RE$: A singleton set $a = \{a\}$ is regular expression for every alphabet $a \in \Sigma$.
4. $r, s \in RE \Rightarrow (r + s), (rs), (r^*) \in RE$: When r, s are regular expression and equivalent to the set of strings R, S, $(r + s), (rs), (r^*)$ respectively denote sets of strings $R \cup S, RS, R^*$ and are all regular expressions.

Remark. We denote a set of strings expressed by $r \in RE$ as $L(r)$.

Regular Expression RE

Exmaples. Here's some examples of the class RE.

- $((0+1)(0+1)(0+1))^*$: Set of strings with length is a multiple of 3.
- $(1+\epsilon)(0^*01)^*0^*$: Set of string without 11 in the string.
- $(00)^*(11)^* + 0(00)^*1(11)^*$: Set of string of form 0^n1^m, $2|n+m$.

Definition (Equivalent)

We say some form of the expression A, B is equivalent if its corresponding sets satisfy $L(A) = L(B)$.

Table of Contents

Deterministic Finite Automata DFA

Goal. In this section we define another Language class, called *DFA*.

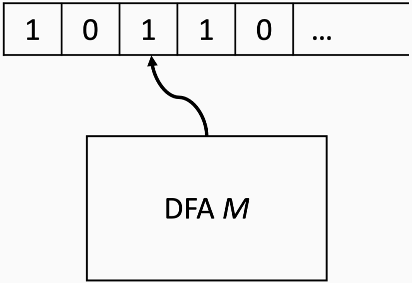
Definition (Deterministic Finite Automata)

We denote a language class *Deterministic Finite Automata* as *DFA*.
Language $M = (Q, \Sigma, \delta, q_0, F)$ is said to be in *DFA* if it is composed with

1. Q : Finite set of *States*.
2. Σ : The *Alphabet* set.
3. $\delta : Q \times \Sigma \to Q$: *Transfer function* that chooses the next state.
4. $q_0 \in Q$: The *Initial state*.
5. $F \subseteq Q$: The *Final states*.

Deterministic Finite Automata DFA

Remark. Deterministic Finite Automata runs by two extra devices, called *Head* and *Tape*. *Tape* contains the input string, while *Head* moves far left to far right reading the input string.



When the current state is $q \in Q$, and the *Head* reads the alphabet $a \in \Sigma$, the next state is $\delta(q, a)$ and the head moves right and reads the next character in the *Tape*.

Deterministic Finite Automata DFA

Notation. We describe state transfer using $\vdash_M$. When our current configuration (i.e. current state and unread part of string) is $(q, 110)$ and if $\delta(q, 1) = q'$ then

$$(q, 110) \vdash_M (q', 10).$$

If clear, omit M and use $\vdash$. 0 or more steps of transfer is denoted as $\vdash^*$.

Definition

We say a string $\omega \in \Sigma^*$ is *accepted* by DFA M if DFA M with input ω ends in the *Final state* F. We define the language $L(M)$ as the set of strings that are accepted by M :

$$L(M) = \{\omega \in \Sigma^* : (q_0, w) \vdash^* (f, \epsilon), f \in F\}.$$

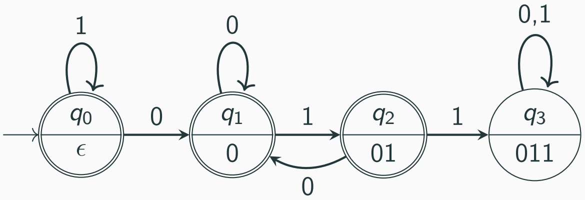
Define a map $\delta^* : Q \times \Sigma^* \to Q$ inductively using $\delta^*(q, \epsilon) = q$ and $\delta^*(q, \omega a) = \delta(\delta^*(q, \omega), a)$ for all $\omega \in \Sigma^*, a \in \Sigma, q \in Q$. Then,

$$L(M) = \{\omega \in \Sigma^* : \delta^*(q_0, \omega) \in F\}.$$

Deterministic Finite Automata DFA

Exmaples. Here's some examples of the class *DFA*.

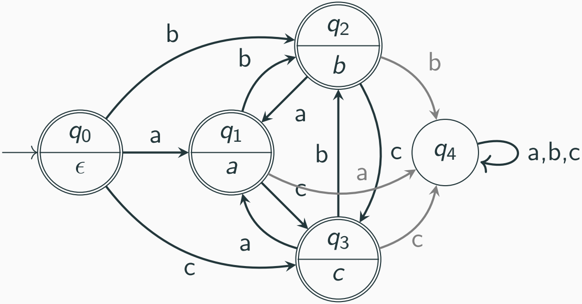
- Set of string without 011 in the string.



Deterministic Finite Automata DFA

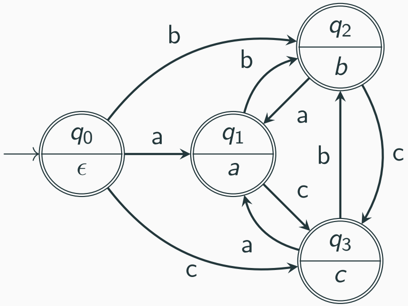
Exmaples. Here's some examples of the class *DFA*.

- Set of string without identical consecutive characters. $\Sigma = \{a, b, c\}$.



Remark. For the simplicity, we allow δ to be a *partial function*. In such case, we append a new state $q' \notin F$ so we can extend δ as

$$\delta(q, a) = \begin{cases} \delta(q, a) & \text{if } \delta(q, a) \text{ is defined} \\ q' & \text{if not defined or } q = q' \end{cases}$$



We call q', which was q_4 in the previous slide as *dead state*.

Table of Contents

Nondeterministic Finite Automata NFA

Goal. In this section we define another Language class, called *NFA*.

Definition (Nondeterministic Finite Automata)

We denote a language class *Nondeterministic Finite Automata* as *NFA*. Language $M = (Q, \Sigma, \Delta, q_0, F)$ is said to be in *NFA* if it is composed with

1. Q : Finite set of *States*.
2. Σ : The *Alphabet* set.
3. $\Delta : Q \times (\Sigma \cup \epsilon) \rightrightarrows Q$: *Transfer relation*, collection of next state(s).
4. $q_0 \in Q$: The *Initial state*.
5. $F \subseteq Q$: The *Final states*.

Nondeterministic Finite Automata NFA

Remark. Nondeterministic Finite Automata is a generalization of *DFA* :

- it can have multiple next states or no next state.
- it can transfer with the empty string ϵ.

Remark. We may view Δ as a function of $Q \times (\Sigma \cup \epsilon) \to 2^Q$. In this sense, extend Δ as $\Delta(P, a) = \cup_{q \in P} \Delta(q, a)$ for $P \in 2^Q$.

Remark. We now define a set of states $E(q)$ for each state q. $E(q)$ stands for the states reachable from q without reading any string :

$$E(q) = \bigcup_{i \in \mathbb{Z}_{\geq 0}} E^i(q), \quad E^0(q) = q, \quad E^{k+1}(q) = \Delta(E^k(q), \epsilon).$$

Remark. While we can define $\vdash$ similarly, we should note that

transferred result need not be unique.

Nondeterministic Finite Automata NFA

Definition

We say a string $\omega \in \Sigma^*$ is *accepted* by NFA M if NFA M with input ω ends in the *Final state* F. We define the language $L(M)$ as the set of strings that are accepted by M :

$$L(M) = \{\omega \in \Sigma^* : (q_0, w) \vdash^* (f, \epsilon), f \in F\}.$$

Define a map $\Delta^* : Q \times \Sigma^* \to 2^Q$ inductively using $\Delta^*(q, \epsilon) = E(q)$ and $\Delta^*(q, \omega a) = E(\Delta(\Delta^*(q, \omega), a))$ for all $\omega \in \Sigma^*, a \in \Sigma, q \in Q$. Then,

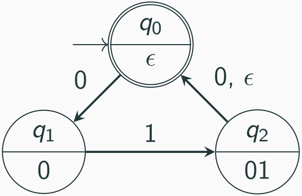
$$L(M) = \{\omega \in \Sigma^* : \Delta^*(q_0, \omega) \cap F \neq \emptyset\}.$$

Remark. We may assume that $|F| = 1$. If $|F| > 1$, then append new state f, use $\{f\}$ as a *Final state*, and append $\Delta(q, \epsilon) = f$ for all $q \in F$.

Nondeterministic Finite Automata NFA

Exmaples. Here's some examples of the class *NFA*.

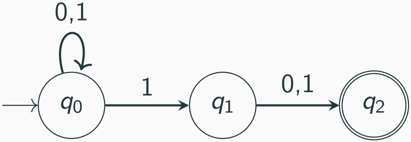
- Regular expression $(010 + 01)^*$.



Nondeterministic Finite Automata NFA

Exmaples. Here's some examples of the class *DFA*.

- Set of string with 1 as the second last character.



Remark. It is trivial that *DFA* is an instance of *NFA*. Thus, if a language L is of class *DFA*, then it is also *NFA*.

Remark. Generally, it is much simpler to build *NFA* than *DFA*.

- Set of string with 1 as the second last character (*DFA*).

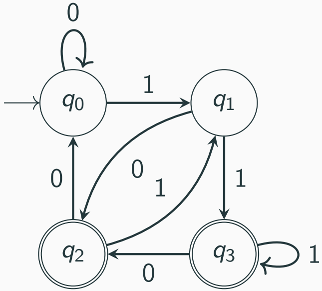


Table of Contents

$DFA = NFA$

Goal. In this section we prove $DFA = NFA$.

Remark. We already know $DFA \subseteq NFA$.

Idea. The idea of proving $NFA \subseteq DFA$ is to build an equivalent DFA for a given NFA with each subset of 2^Q as a state of DFA.

Theorem

$$DFA = NFA$$

To be precise, for a language L, there exists a DFA that accepts strings in L and denies others, if and if only when there exists a NFA that accepts strings in L and denies others.

DFA $=$ *NFA*

proof of $NFA \subseteq DFA$**.**

Suppose *NFA* $M(Q, \Sigma, \Delta, q_0, F)$ is given. We aim to build an equivalent *DFA*. Consider *DFA* machine $M_D(Q_D, \Sigma, \delta, q_D, F_D)$ as :

1. $Q_D = 2^Q$.
2. $\delta : 2^Q \times \Sigma \to 2^Q$, such that $\delta(P, a) = E(\Delta(P, a))$ for $P \subseteq Q$.
3. $q_D = E(q_0)$.
4. $F_D = \{P : P \cap F \neq \emptyset\}$.

Then we can easily conclude the proof by showing that a string is accepted by M if and only when it is accepted by M_D. The proof is done via induction on the length of the input string. $\square$

Table of Contents

Goal. In this section we prove that RE is equivalent to finite automata.

Idea. We first prove that $RE \subseteq NFA$ and then prove $DFA \subseteq RE$.

Theorem

$$RE = DFA = NFA$$

To be precise, for a language L, it is describable as Regular expression if and only if when there exists a DFA (of NFA) that accepts strings in L and denies others.

Idea. For $RE \subseteq NFA$, we will build NFA.

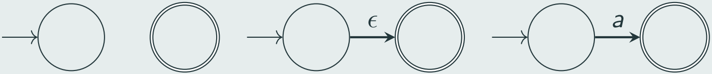
Idea. For $DFA \subseteq RE$, we will construct RE.

$RE = (DFA = NFA)$

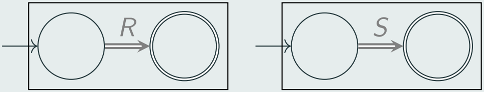
proof of $RE \subseteq NFA$.

Recall that RE is defined inductively. We construct NFA with *singleton Final state* corresponding to RE also inductively.

1. First, $\emptyset, \epsilon, a \in RE$ for $a \in \Sigma$. Corresponding NFA are :

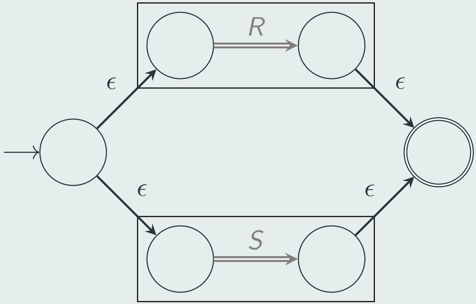


2. $r, s \in RE \Rightarrow (r + s), (rs), (r^*) \in RE$:
   Assume r, s have corresponding NFA : R, S :



proof of $RE \subseteq NFA$, continued.

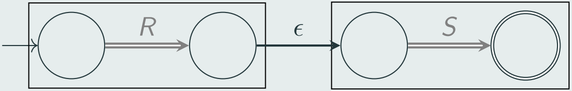
Then NFA for $(r + s)$ is :



NFA for $(r + s)$

proof of $RE \subseteq NFA$, continued.

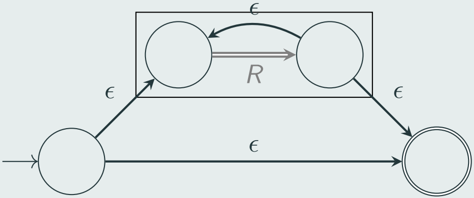
NFA for (rs) is :



NFA for (rs)

proof of $RE \subseteq NFA$, continued.

NFA for (r^*) is :



NFA for (r^*)

$\square$

Now let's prove $DFA \subseteq RE$. The idea is to categorize the strings via the states it passes.

proof of $DFA \subseteq RE$.

Let's write the given DFA M as :

$$M = (Q, \Sigma, \delta, q_1, F), \quad Q = \{q_1, q_2, \cdots, q_n\}.$$

Let's define a set of strings R_{ij}^k as

$$R_{ij}^k = \left\{ \omega \in \Sigma^* : \text{when } (q_i, \omega) \vdash^* (q_m, \omega'), \begin{cases} m \le k & \text{if } |\omega'| > 0 \\ m = j & \text{if } \omega' = \epsilon \end{cases} \right\},$$

strings which transfers q_i to q_j without passing any states with index number larger than k.

proof of $DFA \subseteq RE$, continued.

For the remark,

$$L(M) = \bigcup_{q_j \in F} R_{1j}^n.$$

The set R_{ij}^k could be calculated recursively as follows :

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma : \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a \in \Sigma : \delta(q_i, a) = q_i\} \cup \epsilon & \text{if } i = j \end{cases}$$

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1}(R_{kk}^{k-1}) * R_{kj}^{k-1}.$$

proof of $DFA \subseteq RE$, continued.

In other words, each R_{ij}^k are a RE language r_{ij}^k defined recursively as :

$$r_{ij}^0 = \begin{cases} \{a \in \Sigma : \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a \in \Sigma : \delta(q_i, a) = q_i\} + \epsilon & \text{if } i = j \end{cases}$$

$$r_{ij}^k = r_{ij}^{k-1} + (r_{ik}^{k-1}(r_{kk}^{k-1}) * r_{kj}^{k-1}).$$

Thus, $L(M)$ is RE language since

$$L(M) = \sum_{q_j \in F} r_{1j}^n.$$

$\square$

Table of Contents

Theorem (Pump theorem)

Suppose L is a RE language. Then, there exist an integer $t > 0$ such that, for any $\omega \in L$ with $|\omega| \geq t$, it is possible to decompose ω as xyz that satisfies :

1. $|xy| \leq t$ *and* $|y| \geq 1$,
2. *For all* $i \geq 0$, $xy^i z \in L$.

Idea. Find equivalent *DFA* and define t as the number of states.
Remark. The inverse does not hold.

proof of Pump theorem.

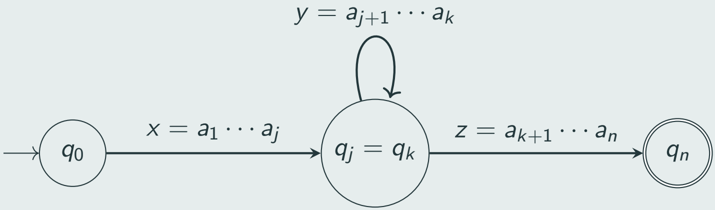
Now we can consider equivalent *DFA* $M = (Q, \Sigma, \delta, q_0, F)$. Choose the number t as $|Q|$, which only depends on the language L. Name the characters of the input string as $\omega = a_1 a_2 \cdots c_n$ where $|\omega| = n \geq t$. Label the state M stays as it reads the input as

$$q_i = \delta^*(q_0, a_1 a_2 \cdots a_i), \quad 1 \leq i \leq n.$$

With the pigeon's hole principle, among $q_0, q_1, \cdots, q_t$ there exists $0 \leq j < k \leq t$ such that $q_j = q_k$.

Further topics

proof of Pump theorem, continued.



Define x, y, z as the figure above. Then, $|xy| = k \leq t$ and $|y| = k - j \geq 1$. Also, from

$$q_j = \delta^*(q_j, y),$$

we can say $q_j = \delta^*(q_j, y^*)$. Hence, $\delta^*(q_0, xy^i z) = q_n \in F$.
$\therefore xy^i z \in L, \quad \forall i \geq 0.$ $\square$

Examples of languages not in *RE*

- $\{0^n 1^n : n \geq 0\}$.
- $\{uu : u \in \Sigma^*\}$. $(|\Sigma| \geq 2)$
- $\left\{a^{n^2} : n \geq 0\right\}$.
- $\{0^m 1^n : m \neq n\}$.

Remark. What is *RE* or *FA* in practical issue? We can see that *RE* are not capable of recording a long term memory, while short term memory of bounded length is possible. In the example, It was not possible to duplicate a sequence after it is once written, while it is possible to check the character written right before.

Remark. Due to such characteristics, *FA*s are commonly refered as the computer with out a memory device. We use them even in real life, such as a door lock.