

CSE 105:  
Computation

---

*by* Jy

# Contents

<b>1</b>	<b>Deterministic Finite Automaton (DFA)</b>	<b>1</b>
1.1	Expressions of DFA's . . . . .	1
1.2	Configurations of DFAs . . . . .	3
1.3	Languages . . . . .	4
<b>2</b>	<b>Nondeterministic Finite Automaton (NFA)</b>	<b>5</b>
2.1	Configurations . . . . .	6
2.2	NFA & DFA . . . . .	8
<b>3</b>	<b>Applications of Finite Automata</b>	<b>10</b>
<b>4</b>	<b>Regular Expression</b>	<b>11</b>
4.1	Regular Language . . . . .	11
4.2	Regular Expression . . . . .	11
4.3	GNFA . . . . .	13
4.4	Non-Regular Languages . . . . .	14
4.5	Pumping Lemma . . . . .	15
<b>5</b>	<b>Finite State Transducers (FST)</b>	<b>16</b>
5.1	FST and DFA . . . . .	17
5.2	Composing FSTs . . . . .	18
5.3	FST-Reductions . . . . .	18

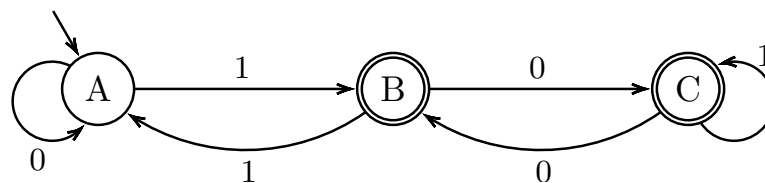
# 1 Deterministic Finite Automaton (DFA)

A machine consists of different states drawn in circles with names. Often a state drawn as a double circle is an “acceptive state,” and a plain circle indicates a “rejective state.” A machine receives a string consisted of ‘1’s and ‘0’s as input and the states change as the machine reads through input digits. An arrow is used to indicate which state is to start with. See [Example 1.1](#) for detailed information.

## 1.1 Expressions of DFA’s

Example 1.1: A DFA

Let’s first look at the DFA below which starts at state *A*.



If the string “010110” is input to the machine, will it result in true or false? Will the state be acceptive or rejective?

010110 → M  $\xrightarrow{1/0 \text{ (True / False, Accept / Reject)}}$

There are two arrows leaving state *A*: one with a label reading ‘1’ which points to state *B* and one reading ‘0’ which goes back to state *A* itself. That means, if an input digit reads ‘1,’ the state changes to *B*, and if ‘0’ the state stays in *A*.

Now step through the procedure:

1. The machine starts off at state *A* with input ‘0,’ which, as explained above, changes the state to *A* itself.
2. Next, the second digit ‘1’ is read so the state is changed to *B*.
3. The next digit ‘0’ makes state *B* to switch to state *C*.
4. Then state *C* reads ‘1’ so no state change occurs.
5. The next digit is ‘1’ again so the state remains still on *C*.
6. Last, the digit ‘0’ switches the state from *C* to *B*.

Thus the input string “010110” changes the machine to state *B*, which is an acceptive state.

**Definition 1.1 DFA.** A DFA is a 5-tuple

$$M = (Q, \Sigma, \delta, s, F)$$

where

$Q$  is a finite set, for states

$\Sigma$  is a finite set, for input alphabet

$s \in Q$ , for start states

$F \subseteq Q$ , for accepting states

$\delta : Q \times \Sigma \mapsto Q$ , a function that specifies the transition between states

### Example 1.2: Denoting machine in Example 1.1

According to definition 1.1, the machine in Example 1.1 can be denoted by

$$M = (Q, \Sigma, \delta, s, F)$$

where

- $Q = \{A, B, C\}$
- $\Sigma = \{0, 1\}$
- $s = \{A\}$
- $F = \{B, C\}$

And function  $\delta$  can be described by the table below.

$\delta$	0	1
A	A	B
B	C	A
C	B	C

**Definition 1.2**  $f_M$ . For any DFA  $M = (Q, \Sigma, \delta, s, F)$ , let

$$f_M : \Sigma^* \mapsto \{\text{True}, \text{False}\}$$

where  $\Sigma^*$  is a set of strings over  $\Sigma$ .

$$f_M(w) = \begin{cases} \text{True}, & \delta^*(s, w) \in F \\ \text{False}, & \text{else} \end{cases}$$

**Definition 1.3**  $\delta^*$ .

$$\delta^* : Q \times \Sigma^* \mapsto Q$$

which is an inductive function defined as

$$\begin{cases} \delta^*(q, \varepsilon) = q \\ \delta^*(q, aw) = \delta^*(\delta(q, a), w) \end{cases}$$

where  $\varepsilon$  is an empty string and  $q \in Q, a \in \Sigma, w \in \Sigma^*$ .

## 1.2 Configurations of DFAs

**Definition 1.4 Configurations.**

$$\text{Conf} = Q \times \Sigma^*$$

**Definition 1.5 Initial Configurations.** *The initial configuration of a machine  $I_M(w) \in \text{Conf}$*

$$I_M(w) = (s, w)$$

**Definition 1.6 Final Configurations.** *The final configuration of a machine  $H_M(w) \subseteq \text{Conf}$*

$$H_M(w) = \{ (q, u) \mid q \in Q, u = \varepsilon \}$$

**Definition 1.7 Output.** *The output of a machine is a function that returns either “True” or “False.”*

$$O_M: H_M \mapsto \{ \text{True}, \text{False} \}$$

defined as

$$O_M(q, \varepsilon) = \begin{cases} \text{True}, & q \in F \\ \text{False}, & \text{else} \end{cases}$$

**Definition 1.8 Transition Relations.**  $R_M \subseteq \text{Conf}$

$$R_M = \rightarrow_M = \{ (q, aw) \rightarrow (\delta(q, a), w) \mid q \in Q, a \in \Sigma, w \in \Sigma^* \}$$

### Example 1.3: Configurations of machine in [Example 1.1](#)

With input “10010” write in mathematical language the configurations of machine in [Example 1.1](#):

$$I_M(10010) = (A, 10010) \rightarrow (B, 0010) \rightarrow (C, 010) \rightarrow (B, 10) \rightarrow (A, 0) \rightarrow (A, \varepsilon) \in H_M$$

And thus the output

$$O_F(A, \varepsilon) = \text{False}$$

.

The machine in fact will only accept integers that are *not* multiples of 3.

**Definition 1.9 .**

$$f'_n(w) = O_F(C_n)$$

### 1.3 Languages

A subset of  $\Sigma^*$  of a DFA that contains all inputs to which the output of the machine is True is called the *language* of the machine.

**Definition 1.10 Regularity of Language.**  $L \subseteq \Sigma^*$  is regular if

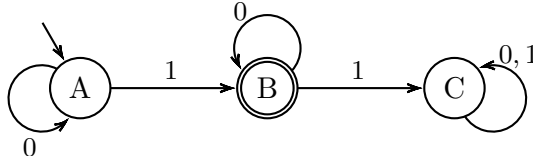
$$\exists \text{DFAM} \mid L(M) = L$$

#### Example 1.4

Given that  $\varepsilon^* = \{\varepsilon\}$  and  $\Sigma^* = \{\varepsilon, 1, 0, 10, 101, \dots\} = \{0, 1\}^*$ , which of the following languages are regular?

- $L_1 = \{w \in \{0, 1\}^* \mid w \text{ is a power of } 2\}$ , and
- $L_2 = \{w \in \{0, 1\}^* \mid w \text{ is a power of } 3\}$ .

$L_1$  is regular while  $L_2$  is not. A binary number that is a power of 2 consists of only one 1 and all other digits should be 0s. A DFA that recognizes the language would be



**Definition 1.11 Operations on Languages.**

**Complement**  $L^C = \{w \in \Sigma^* \mid w \notin L\}$

**Union**  $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$

**Intersection**  $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

**Concatenation**  $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$

Example 1.5: If  $L$  is regular, is  $L^C$  also regular?

Yes.

*Proof of statement  $\mathbb{R}$  is closed under complement.* Let  $L \in \mathbb{R}$ , prove  $L^C \in \mathbb{R}$  :

By definition,

$$\exists M = (Q, \Sigma, \delta, s, F) \text{ s.t. } L(M) = L.$$

Let  $M' = (Q, \Sigma, \delta, s, F^C)$ ,  
then  $L(M') = L(M)^C = L^C$ .  
 $L^C \in \mathbb{R}$  because  $L^C = L(M')$ . □

Example 1.6:  $\forall L_1, L_2$

If  $L_1$  and  $L_2 \in \mathbb{R}$ , then  $L_1 \cup L_2 \in \mathbb{R}$ .

Yes,  $\mathbb{R}$  is closed under union.

## 2 Nondeterministic Finite Automaton (NFA)

In automata theory, a finite state machine is called a deterministic finite automaton (DFA), if

- each of its transitions is uniquely determined by its source state and input symbol, and
- reading an input symbol is required for each state transition.

A nondeterministic finite automaton (NFA), or nondeterministic finite state machine, needn't obey these restrictions. In particular, every DFA is also an NFA. (via [Wikipedia](#))

**Definition 2.1 NFA.** An NFA is a 5-tuple

$$N = (Q, \Sigma, \delta, s, F)$$

where

- $Q$  and  $\Sigma$  are finite sets
- $s \in Q, F \subseteq Q$
- $\delta: Q \times \Sigma_\epsilon^1 \mapsto \mathcal{P}(Q)$

As said, since a DFA is an NFA, the definition of NFA is simply a generalized version of DFA's. The difference that in an NFA a state can transit to multiple states shows a different transition function  $\delta$ —the transition function of an NFA maps to a set of states instead of exactly one state as of a DFA.

---

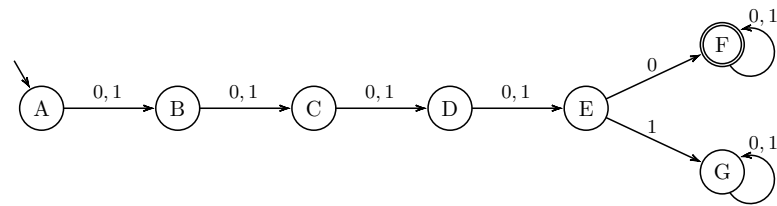
<sup>1</sup> $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

## Example 2.1: How many states do you need?

Given

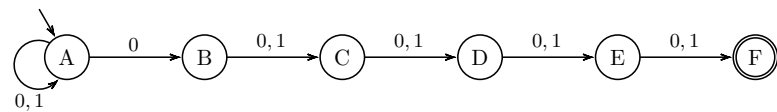
$$L = \{ w \in \{0, 1\}^* \mid \text{the 5th digit of } w \text{ is } 0 \}$$

the DFA is easy to draw:



What if the digits are counted from the right, i.e. the last 5th digit?

This is where NFAs could be useful. Unlike the earlier language which only takes seven states to create a DFA that recognize it, this one could use numbers of states since we don't really know how many digits to expect before reaching the one that needs to be 0. An NFA on the other hand, could finish the job with merely six states:



## 2.1 Configurations

Again the configurations of an NFA are very similar to that of a DFA—except that an NFA has an  $\varepsilon$  transition, a transition that is performed without requiring any input.

### Definition 2.2 Configurations of NFA.

$$\text{Conf} = Q \times \Sigma^*$$

$$I(w) = (s, w)$$

$$H = Q \times \{\varepsilon\}$$

$$O(q, \varepsilon) = \begin{cases} \text{True}, & q \in F \\ \text{False}, & \text{else} \end{cases}$$

$$R = \{ (q, aw) \mapsto (q', w) \mid \forall q \in Q, a \in \Sigma_\varepsilon, w \in \Sigma^* \}$$

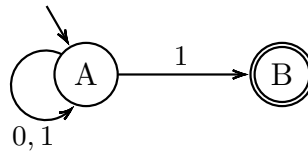
### Definition 2.3 Language of NFA.

$$L(N) = \{ w \mid \exists \text{ accepting computation on input } w \}$$



### Example 2.2: A simple NFA

The following graph shows an automaton  $M$  with a binary alphabet that determines if the input ends with a 1.



The automaton  $M$  shown above is *not* a DFA since reading a 1 in state  $A$  can lead to  $A$  or to  $B$ .  
 $M$  in formal notation is

$$M = \{ \{ A, B \}, \{ 0, 1 \}, \delta, A, \{ B \} \}$$

where the transition function  $\delta$  can be defined by the state transition table:

	0	1
$A$	$\{ A \}$	$\{ A, B \}$
$B$	$\emptyset$	$\emptyset$

Note that  $\delta(p, 1)$  has more than one state—again therefore  $M$  is nondeterministic.

Some possible state sequences for the input word “1011” are:

input	1	0	1	1
State sequence 1	$A$	$B$	?	
State sequence 2	$A$	$A$	$A$	$B$
State sequence 3	$A$	$A$	$A$	$A$

The word is accepted by  $M$  in sequence 3; it doesn’t matter that both other sequences fail to do so. In contrast, the word “10”, which the state sequences for are shown below, is rejected by  $M$ , since there is no way to reach the only accepting state,  $B$ , by reading the final “0” symbol or by an  $\varepsilon$ -transition.

input	1	0
State sequence 1	$A$	$B$
State sequence 2	$A$	$A$

(Example via [Wikipedia](#))

## 2.2 NFA & DFA

According to [Theorem 2.1](#), any NFA can be translated to a DFA. The method is to fully expand the NFA and draw out every branch of it, which is explained with more details in [Theorem 2.2](#).

**Theorem 2.1 .**

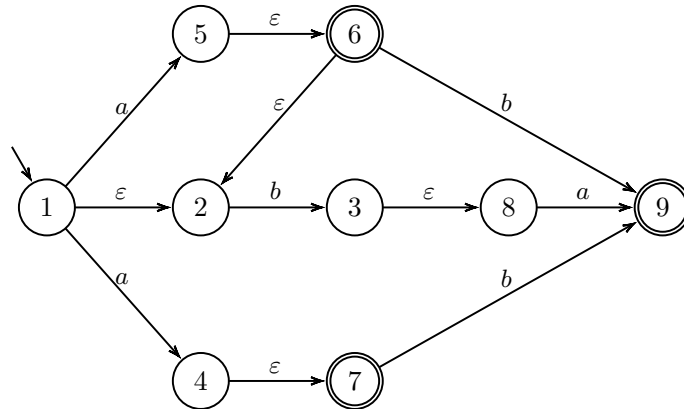
$$\begin{aligned} \forall \text{NFAN} &= (Q, \Sigma, \delta, s, F), \\ \exists \text{DFAM} &= (Q', \Sigma, \delta', s', F') \text{ s.t. } L(N) = L(M) \end{aligned}$$

*Proof of [Theorem 2.1](#).*

$$\begin{aligned} Q' &= \mathcal{P}(Q) \\ F' &= \{ A \subseteq Q \mid A \cap F \neq \emptyset \} \\ s' &= E(\{s\}) = \left\{ q \in Q \mid \exists s \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} q_3 \cdots \xrightarrow{\varepsilon} q \right\} \\ \delta'(A, a) &= E\left(\bigcup_{q \in A} \delta(q, a)\right) \end{aligned}$$

□

### Example 2.3: Convert the following NFA to DFA



Start with the initial state, state 1, form a set of its  $\varepsilon$ -enclosure  $I = E(\{s\}) = \{1, 2\}$ .

For each element of set  $I$ , find  $\delta(q, a)$  where  $q \in I$  and produce a new set  $\{\delta(1, a), \delta(2, a)\} = \{4, 5, \emptyset\} = \{4, 5\}$ , Let  $I_a$  be its  $\varepsilon$ -enclosure  $I_a = E(\{4, 5\}) = \{4, 7, 5, 6, 2\}$ .

Similar for input symbol  $b$ , form  $I_b = E(\{\delta(1, b), \delta(2, b)\}) = E(\{3\}) = \{3, 8\}$ .

Make  $I_a$  and  $I_b$  new states and find  $I_{a_a}, I_{a_b}, I_{b_a}$ , and  $I_{b_b}$  respectively; Repeat the procedure until no new states are formed.

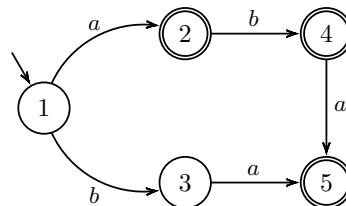
The following table neatly traces the whole process:

States	$I$	$I_a$	$I_b$
1	$\{1, 2\}$	$\{4, 7, 5, 6, 2\}$	$\{3, 8\}$
2	$\{4, 7, 5, 6, 2\}$	$\emptyset$	$\{9, 3, 8\}$
3	$\{3, 8\}$	$\{9\}$	$\emptyset$
4	$\{9, 3, 8\}$	$\{9\}$	$\emptyset$
5	$\{9\}$	$\emptyset$	$\emptyset$

With the table, we use the  $\varepsilon$ -enclosure of the original initial state as the initial state of our DFA and those contain original final states as our new final states. Thus the new initial state is state 1 and final states are 2, 4, 5 (state 2 contains 6 and 7, state 4 and 5 have 9).

The first row shows that state 1 reading the input  $a$  produces state  $\{4, 7, 5, 6, 2\}$ , which in the table is state 2. Therefore in the DFA, we expect state 1 transited to state 2 with an  $a$ -arrow.

With all the information we need, draw out the DFA:



### 3 Applications of Finite Automata

A finite automaton can be applied to

- defining models of computation,
- testifying equivalence between models,
- etc.

**Definition 3.1 Reverse.** For a string,

$$\text{rev}((a_1, a_2, \dots, a_n)) = (a_n, \dots, a_2, a_1);$$

For a language,

$$\text{rev}(L) = \{ \text{rev}(w) \mid w \in L \}.$$

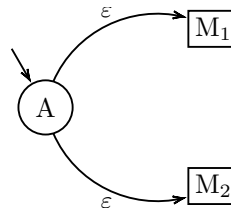
It is easy to find that

$$\forall L \in \mathbb{R}, \text{rev}(L) \in \mathbb{R}.$$

#### Example 3.1: $\mathbb{R}$ is closed under union

Recall [Example 1.6](#), with NFA it is much easier to prove the theorem now.

*Proof of Example 1.6.* Let  $M_1, M_2$  be NFAs for  $L_1$  and  $L_2$ , build an NFA for  $L_1 \cup L_2$  by simply adding a new initial state that transit to  $s_1$  and  $s_2$  with  $\varepsilon$  arrows:



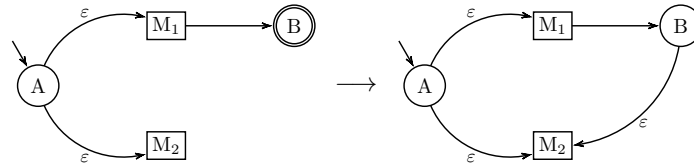
□

### Example 3.2: $\mathbb{R}$ is closed under concatenation

*Proof.* Let  $M_1, M_2$  be NFAs for  $L_1$  and  $L_2$ , build an NFA for  $L_1 \cdot L_2$ :

1. use a new initial state that transit to  $s_1$  and  $s_2$  with  $\varepsilon$  arrows as in [Example 3.1](#); and
2. change the final states in  $M_1$  to regular states and connect them to  $s_2$  with  $\varepsilon$  arrows.

A rough graph that represents the above steps is



so

$$L(M) = L(M_1) \cdot L(M_2) = \{ wv \mid w \in L(M_1), v \in L(M_2) \}.$$

□

## 4 Regular Expression

### 4.1 Regular Language

We've learned that the class of regular languages is closed under

- union, <sup>2</sup>
- intersection, <sup>3</sup>
- concatenation, and <sup>4</sup>
- star. <sup>5</sup>

Many complex languages can be built using these operations; ?? is one practical example.

### 4.2 Regular Expression

A regular expression (abbreviated regex or regexp) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. (via [WikiPedia](#))

<sup>2</sup> if  $L_1, L_2 \in \mathbb{R}$ , then  $L_1 \cup L_2 = \{ w \mid w \in L_1 \vee w \in L_2 \} \in \mathbb{R}$ .

<sup>3</sup> if  $L_1, L_2 \in \mathbb{R}$ , then  $L_1 \cap L_2 = \{ w \mid w \in L_1 \wedge w \in L_2 \} \in \mathbb{R}$ .

<sup>4</sup> if  $L_1, L_2 \in \mathbb{R}$ , then  $L_1 \cdot L_2 = \{ w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2 \} \in \mathbb{R}$ .

<sup>5</sup> if  $L \in \mathbb{R}$ , then  $L^* = \{ w_1 w_2 \cdots w_n \mid w_1, w_2, \cdots, w_n \in L, n \geq 0 \} \in \mathbb{R}$ .

### Example 4.1: Build a rather complex language

It is common we want to search for all numbers, say, in a file. The following set is a language that matches all numbers greater than 10 and allowing appearance of commas.

$$L = \{1, \dots, 9\} \cdot (\{0, 1, \dots, 9\}^* \cdot \{, \})^* \cdot \{0, 1, \dots, 9\}$$

Consider

$$\begin{aligned} L_1 &= \{1, \dots, 9\} \\ L_2 &= \{0, 1, \dots, 9\}^* \cdot \{, \} \\ L_3 &= \{0, 1, \dots, 9\} \end{aligned}$$

so  $L = L_1 \cdot L_2^* \cdot L_3$ .

What are  $L_1$ ,  $L_2$  and  $L_3$ ?

$L_1$  is a set of all digits from 1 to 9;

$L_3$  is a set of all digits from 0 to 9;

$L_2$  is a little more complicated, it can also be written as  $L_3^* \cdot \{, \}$ , while  $L_3^*$  matches a string of any number of elements in  $L_3$ , that is, a string made of all digits with unknown length. What  $\cdot \{, \}$  does is it appends a comma to the end of this string. In all,  $L_2$  is a number of digits with a comma at the end.

With that, the set  $L_1 \cdot L_2^* \cdot L_3$  can be now (roughly) seen as:

*a digit and a number of (a number of digits and a comma) and a digit*

Now, notice there is a leading digit and an ending one, why should one be in  $L_1$  and the other  $L_3$ ? Because matching from set  $L_1$  rules out the numbers with leading 0s ( $L_1$  doesn't have 0), and the rest of digits should allow 0s. The middle portion ( $L_2^*$ ) allows unknown number of strings from  $L_2$  (even 0) in between the first and last digit. In the case where the number of  $L_2$  is 0, which makes the input string also in set  $L_1 \cdot L_3$ , the input string is a two-digit number (10 to 99).

### Example 4.2: $\mathbb{R}$ closed under star

*Proof.* use  $\varepsilon$  transition from the final states to the initial states (including states transited directly from the initial state with an  $\varepsilon$  arrow) to prove that  $\mathbb{R}$  is closed under star.  $\square$

A regex  $E$  has the following rules

$E = a$	$L(a) = \{ a \}$
$E = \varepsilon$	$L(\varepsilon) = \{ \varepsilon \}$
$E = E_1 \cdot E_2$	$L(E) = L(E_1) \cdot L(E_2)$
$E = E_1 + E_2$	$L(E) = L(E_1) + L(E_2)$
$E = (E_1)^*$	$L(E) = L((E_1)^*) = L(E_1)^*$
$E = \emptyset$	$L(\emptyset^*)$

In fact, the rule

$$L(\varepsilon) = \{ \varepsilon \}$$

can be replaced with

$$L(\emptyset^*).$$

**Theorem 4.1 Equivalence of Regex and Regular Language.**

$$\forall L \in \mathbb{R}, \exists \text{Regex } E \text{ s.t. } L(E) = L.$$

### 4.3 GNFA

A generalized nondeterministic finite automaton (GNFA) is an NFA where

- there are exactly one arrow entering and one leaving a state,
- states can be transited using regexes ( $\emptyset$  arrows, star arrows, etc.),
- there are no arrows entering the initial state,
- there is only one final state, and
- there are no arrows leaving the final state.

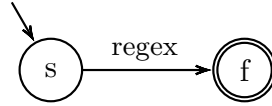
**Definition 4.1 GNFA.** A GNFA is defined as a 5-tuple

$$(Q, \Sigma, \delta, s, f)$$

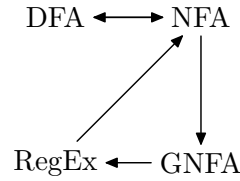
where

$$\delta: (Q \setminus \{f\}) \times (Q \setminus \{s\}) \mapsto \mathbb{R}(\Sigma)$$

A GNFA, since it can use transitions with regexes, can help develop a regex of the language of the GNFA by removing states one at a time until we have the form



The concept of regex finely relates to all automata we've learned so far:



## 4.4 Non-Regular Languages

Are there non-regular languages? The answer is obviously “Yes,” but what are they? Take binary strings as example, find  $L \subseteq \{0, 1\}^*$  s.t.  $\forall \text{ regex } E, L(E) \neq L$ . With symbols  $\{0, 1, \cdot, +(\cup), *, (, ), \emptyset\}$ , any regular language can be expressed,

$$E = (0 + 1)^*.$$

Map each of the symbols to 0 = 000, 1 = 001,  $\cdot$  = 010,  $\dots \emptyset$  = 111 and use function  $\varphi$  to rewrite the regex above,

$$\varphi(E) = 101\,000\,011\,001\,110\,100 \in \{0, 1\}^*$$

so  $L(E) \subseteq \{0, 1\}^*$ . Then the non-regular language is

$$L = \{ \varphi(E) \mid \varphi(E) \notin L(E) \},$$

which is called the diagonal language (see 4.2).

### Definition 4.2 Diagonal Language.

$$D = \{ \varphi(E) \mid E \in \text{RegEx}(\{0, 1\}), \varphi(E) \notin L(E) \}$$

*Proof of  $\varphi(" \emptyset ") = " 111 " \notin L(\emptyset) = \emptyset$ .* Assume for contradiction  $L$  is regular, Let  $E$  s.t.  $L(E) = L$ .

$$\varphi(E) \in L \leftrightarrow \varphi(E) \notin L(E) = L.$$

□



## 4.5 Pumping Lemma

Every regular language satisfies property  $P$ , if a language  $L$  does not satisfy  $P$  then  $L$  is non-regular. In proving, we will first assume  $P(L)$  and lead to contradiction.

**Definition 4.3 Property  $P$ .**

$$\exists p \geq 1 \quad (\text{pumping length})$$

$$\forall w \in L, |w| \geq p$$

$$\exists x, y, z \in \Sigma^*, w = x \cdot y \cdot z$$

and

- $\forall i \geq 0, \quad x \cdot y^i \cdot z = x \overbrace{yy \cdots y}^i z \in L,$
- $|y| \geq 0 (y \neq \varepsilon),$
- $|xy| \leq p.$

### Example 4.3

let

$$P = \{ w \in \{1\}^* \mid |w| \text{ is a prime number} \}.$$

Claim:  $P$  is not regular.

To prove it, the pumping lemma would help. Here is the proof.

*Proof.*

Assume for contradiction  $P$  is regular.

Therefore, by P.L., there is an  $n \geq 1$  s.t. all  $w \in L$  of length  $|w| \geq n$  can be pumped.

Let  $w = 1^m$  s.t.  $m \geq n$  and  $m$  is a prime,

Thus  $w = 1^m$  satisfies all 3 conditions of P.L.

Let  $w = xyz$  s.t.  $w$  remains satisfying the P.L.

Then  $|x| + |y| + |z| = m$  is a prime.

By P.L.,

$$\begin{aligned} xy^iz &= 1^{|x|} \cdot 1^{i \cdot |y|} \cdot 1^{|z|} \\ &= 1^{|x| + i|y| + |z|} \\ &\in L. \end{aligned}$$

That is,  $|x| + i|y| + |z|$  is also a prime.

Let  $i = m + 1$ ,

$$\begin{aligned} &|x| + |y| + |z| + (i - 1)|y| \\ &= m + (i - 1)|y| \\ &= m + m|y| \\ &= m(1 + |y|) \end{aligned}$$

is not a prime,

Contradicting  $xy^iz \in L$ . □

## 5 Finite State Transducers (FST)

**Definition 5.1 Finite State Transducers (FST).** An FST is

$$M = (Q, \Sigma, \Gamma, \delta, s),$$

where

- $Q$  is finite set of states,
- $\Sigma, \Gamma$  are finite sets of i/o symbols,
- $s \in Q$  is the start state, and
- $\delta: Q \times \Sigma \mapsto Q \times \Gamma^*$  is the transition function.

**Definition 5.2 Transition Function of an FST.**  $\delta^*(q, w) \in Q \times \Gamma^*$  is the extended relation function

defined as such:

$$\begin{aligned}
\delta^*(q, \varepsilon) &= (q, \varepsilon) \\
\delta^*(q, ab) &= (s, uv) \quad \text{if} \\
&\quad \delta(q, a) = (r, u) \text{ and} \\
&\quad \delta^*(r, b) = (s, v) \\
&\quad \text{where } a, u \in \Sigma, b, v \in \Sigma^*.
\end{aligned}$$

**Definition 5.3**  $f_M$  of an FST.

$$f_M(w) = u \quad \text{s.t.} \quad \delta^*(s, w) = (q, u).$$

## 5.1 FST and DFA

Combine FST  $M$  with DFA  $M'$  to test the translated string. An FST combined with a DFA is a DFA. Notice that an FST does not have a language; rather than procedures such as DFAs, it instead describes a function.

**Theorem 5.1 .**

$$\begin{aligned}
\forall \text{FST } F &= (Q_F, \Sigma, \Gamma, \delta_F, s_F), \\
\text{DFAD } D &= (Q_D, \Gamma, \delta_D, s_D, F_D), \\
\exists \text{DFAM } M &= (Q, \Sigma, \delta, s, F) \text{ s.t.}
\end{aligned}$$

$$L(M) = f_M^{-1}(L(M_D)) \text{ meaning } L(M) = \{ w \in \Sigma^* \mid f_M(w) \in L(M_D) \}.$$

*Proof of [Theorem 5.1](#).*

$$\begin{aligned}
\text{Let } Q &= Q_F \times Q_D, \\
S &= (s_F, s_D), \\
F &= Q_F \times F_D, \\
\delta: Q \times \Sigma &\mapsto Q \text{ s.t.} \\
\delta((q_F, q_D), a) &= (q'_F, \delta_D^*(q_D, w)) \text{ where } (q'_F, w) = \delta_F(q_F, a).
\end{aligned}$$

□

## 5.2 Composing FSTs

**Theorem 5.2 FSTs are Composable.**

$$\begin{aligned} \forall \text{FST } M_1 &= (Q_1, \Sigma, \Gamma, \delta_1, s_1), M_2 = (Q_2, \Gamma, \Delta, \delta_2, s_2), \\ \exists \text{FST } M &= M_2 \circ M_1 \text{ s.t. } f_M = f_{M_2} \circ f_{M_1}. \end{aligned}$$

*Proof of Theorem 5.2.* Let  $M = (Q, \Sigma, \Delta, \delta, s)$  where  $Q = Q_1 \times Q_2, s = (s_1, s_2) \in Q$  and  $\delta : Q \times \Sigma_1 \mapsto Q \times \Gamma^*$  is the function defined as  $\delta((q_1, q_2), a) = ((q'_1, q'_2), v)$  with  $(q'_1, u) = \delta_1(q_1, a)$  and  $(q'_2, v) = \delta_2^*(q_2, u)$  and It can be easily verified by induction that  $f_M = f_{M_2} \circ f_{M_1}$ .  $\square$

## 5.3 FST-Reductions

**Definition 5.4 Reduction of Languages.**

$$f : \Sigma^* \mapsto \Gamma^* \text{ s.t. } A = f^{-1}(B)$$

is called a reduction from  $A$  to  $B$ . If such reduction exists, we say  $A$  is reducible to  $B$ , denoted by

$$A \leq_{\text{FST}} B.$$

**Theorem 5.3 .**

*If  $A \leq_{\text{FST}} B$  and  $B$  is regular, then  $A$  is regular.*

*If  $A \leq_{\text{FST}} B$  and  $A$  is not regular, then  $B$  is not regular.*