CSE 105: Computation

by Liu Tan

Contents

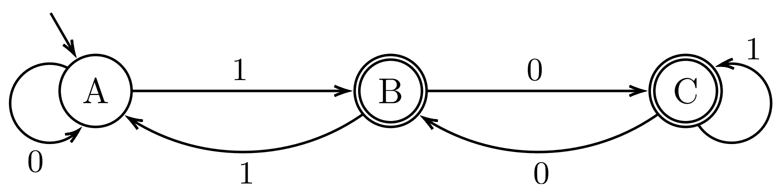
1 Deterministic Finite Automaton (DFA)

A machine consists different 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 read 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?

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

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 difit '0' makse the 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-turple*

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

where

Q *is a finite set, for states*

Σ *is a finite set, for input alphabet*

$s \in Q$, for start states

$F \subseteq Q$, for accepting states

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

Example 1.2: DFA table

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 δ can be described by the table below.

δ	0	1
A	A	B
B	C	A
C	B	C

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

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

where Σ^* is a set of string over Σ.

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

Definition 1.3 δ^*.

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

which is an inductive function defined as

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

where $(q \in Q, a \in \Sigma, w \in \Sigma^*)$

1.2 Configurations of DFAs

Definition 1.4 Configuration.

$$Conf = Q \times \Sigma^*$$

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

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

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

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

Definition 1.7 Machine Output. *The output of a machine is a function that either "True" or "False."*

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

defined as

$$O_M(q, \varepsilon) = \begin{cases} \textit{True}, & if \quad q \in F \\ \textit{False}, & if \quad q \notin F \end{cases}$$

In summary :

- F $\subseteq$ Q
- s $\in$ Q
- $\varepsilon : Q \times \Sigma \mapsto Q$

Example 1.3: Example 1.1 as configurations

With input "10010" write in mathematical language, the confiuration of machine in Example 1.1:

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

And thus the out put

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

.

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

Definition 1.8 Transition Relation.

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

Definition 1.9 n's State's Configuration.

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

e.g.

$$I_M(w) \to_M C_1 \to_M C_2 \to_M \cdots \to_M \in H_M$$

for example

$$L(M) = \{w \in \Sigma^* \mid f_M(w) = \text{True}\}$$
$$L(M_1) \neq \Sigma^*$$
$$1001 \notin 3 \times \mathbb{Z}$$

1.3 Languages

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

In other word, If A is the set of all strings that machine M accepts, we say that A is the ***language of machine*** M and write $L(M) = A$. (M ***recognizes*** A)

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

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

Which means, a DFA could ***recognize*** L. In short, given a regular language, there always exist a DFA could be draw.

Notice that

- ε(small epsilon) = ***empty string***
- Σ(big Sigma) = ***alphabet set***
- $\varepsilon^* = \{\,\varepsilon\,\}$

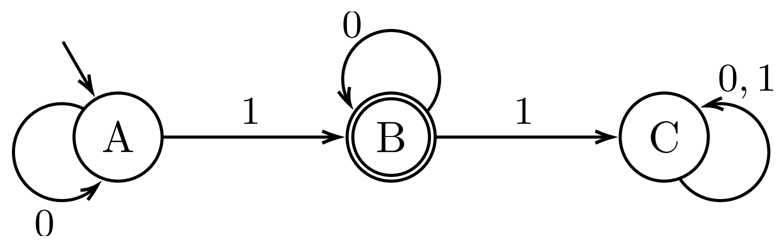
- $\Sigma^* = \{ \varepsilon, 1, 0, 10, 101, \cdots \} = \{ 0, 1 \}^*$,

Example 1.4: Which of the follwing languages are regular?

Given that and which of the following languages are regular?

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

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

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

Theorem 1.1 . *$\mathbb{R}$ is closed under complement.*

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

Yes.

Proof of **??**. 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 $M' = \left(Q, \Sigma, \delta, s, F^C \right)$,
then $L(M') = L(M)^C = L^C$.
$L^C \in \mathbb{R}$ because $L^C = L(M')$. □

Example 1.6: $\forall L_1, L_2$
$L_1 \in \mathbb{R} \vee L_2 \in \mathbb{R} \implies L_1 \cup L_2 \in \mathbb{R}$

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

2 Nondeterministic Finite Automaton (NFA)

In a ***nondeterministic*** machine, several choices may exist for the next state at any point.

2.1 What is NFA

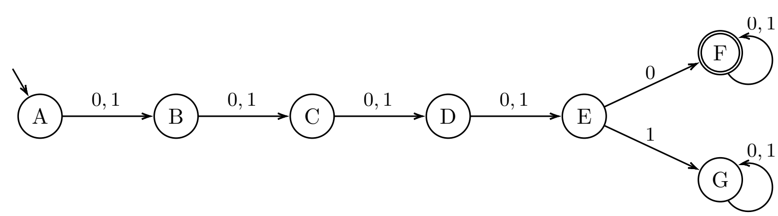
Nondeterminism is a generalization of determinism, so every deterministic finite automaton(DFA) is automatically a nondeterministic finit automaton(NFA). Notice the Difference between DFA figures and NFA's:

1. NFA may has more than one exiting arrow for symbols in the alphabet.
2. NFA may only only have arrows labeled with members of the alphabet but also ε. $(0, 1, \cdots, n)$ arrows may exit from each state with the label ε.

Example 2.1: How many states needed?

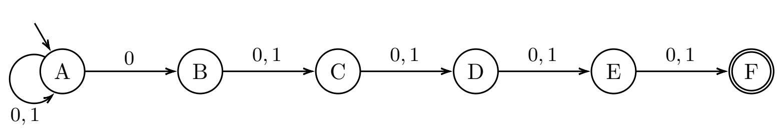
$$L = \{\, w \in \{\, 0, 1\,\}^* \mid \text{ the } 5^{th} \text{ (from the left) of } w \text{ is } 0 \,\}$$

We will need 7 states



What if

$$L = \{\, w \in \{\, 0, 1\,\}^* \mid \text{ the } 5^{th} \text{ from the } \boldsymbol{r}\text{ight) of } w \text{ is } 0 \,\}$$



Notice this is a **DFA**, state A has two exit arrows for 0.

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

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

where

- Q *and* Σ *are finite sets*
- $s \in Q$
- $F \subseteq Q$
- $\delta \colon Q \times \Sigma_\varepsilon$[1] $\mapsto \mathcal{P}(Q)$ [2]
  $\delta(A, \varepsilon) = \{ H \}$ $\delta(D, \varepsilon) = \emptyset$

As said, since a DFA ***is*** an NFA, the definition of NFA is simply a generalized version of DFA's. The difference between NFA and DFA regard as transition function δ is in NFA, δ maps to a set of states instead of exactly one state as of a DFA

2.2 Configurations of NFAs

Definition 2.2 Configurations of NFA.

$$\begin{aligned}
\text{Conf} &= Q \times \Sigma^* \\
I_M(w) &= (s, w) \\
H_M(w) &= Q \times \{ \varepsilon \} \\
O_M(q, \varepsilon) &= \begin{cases} \text{True}, & if \quad q \in F \\ \text{False}, & \textit{otherwise} \end{cases} \\
R &= \{ (q, aw) \mapsto (q', w) \mid \forall q \in Q, a \in \Sigma_\varepsilon, w \in \Sigma^* \}
\end{aligned}$$

After reading the symbol, an NFA splits into multiple copies of itself and follows ***all*** the possibilities in parallel. If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy of the machine dies, along with the branch of the computation associated with if.Finally, if ***any one*** of these copies of the machine is in an accept state at the end of the input, the NFA accepts the input string. That is to say, if at least one of these processes accepts, then the entire computation accepts.

---

[1] $\Sigma_\varepsilon = \Sigma \cup \{ \varepsilon \}$
[2] $\mathcal{P}(Q)$ = powerset of Q

Definition 2.3 Computation.

$$C_0, C_1, \ldots, C_n$$

Computation is a sequence of Configurations, such that

$$C_0 = I(w) \quad \forall i, (C_i, C_{i+1} \in \mathbb{R}) \quad [C_i \mapsto C_{i+1}], C_n \in H$$

an NFA is Accepting if $O(C_n) = \text{ True }$, Rejecting if $O(C_n) = \text{ False}$

Definition 2.4 Language of NFA.

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

3 NFA & DFA

According to Theorem 3.1, and 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 3.

Theorem 3.1 .

$$\forall N = (Q, \Sigma, \delta, s, F), \exists \text{DFA} \quad M = (Q', \Sigma', \delta', s', F') s.t. L(N) = L(M)$$

Proof of Theorem 3.1. Let $N = (Q, \Sigma, \delta, s, F)$ be the NFA recognizing some language A, we construct a DFA $M = (Q', \Sigma, \delta', s', F')$ recognizing A.

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

□

3.1 Equivalence of NFAs and DFAs

A finite automaton(DFA $\iff$ NFA)

1. Defining Models of computation.
2. testifying equivalence between models.

Definition 3.1 Reverse.

- *reverse of a string*

$$\text{rev}\big((\,Q_1, Q_2, \cdots, Q_n\,)\big) = \big(\,Q_n, Q_(n-1), \cdots, Q_1\,\big).$$

- *reverse of a language*

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

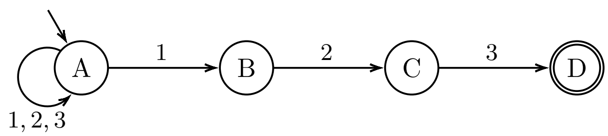


Figure 1: Example of NFA

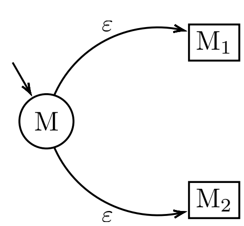
It is easy to find out that

$$\forall L \in \mathbb{R}, \text{rev}(L) \in \mathbb{R}.$$ [3]

Example 3.1: Regular language is closed under union

Recall Example 1.6, with NFA, we could proof Theorem 1.1 much more easier now.

Proof of Theorem 1.1. Let M_1, M_2 become NFA for L_1 and L_2. We build a NFA for $L_1 \cup L_2$ by simply adding a new initial state that transit to s_1 and s_2 with ε arrows:



□

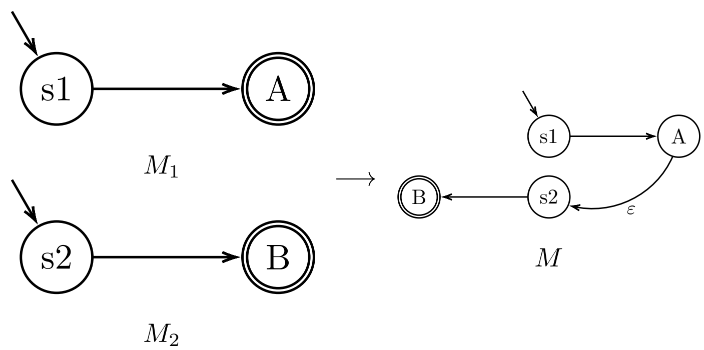
[3] $\mathbb{R}$ = Regular language

Example 3.2: Regular language is closed under concatenation

Proof. Let M_1, M_2 become NFA for L_1 and L_2. We build a NFA for $L_1 \circ L_2$:

1. Assign M's start to be the start state of M_1 which is s_1 and
2. change the final states in M_1 to regular states and connect them to s_2 with additional ε arrows that nondeterministically allow branching to M_2 whenevr M_1 is in an accept state, signifying that it has found an initial piece of the input that constitues a string in L_1
3. The accept states of M are the accept states of M_2 only. only.

See the graph for visualization:



thus

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

□

4 Regular Expression

4.1 Regular Language

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

- union, [4]
- intersection, [5]
- concatenation, and [6]
- star. [7]

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. (via WikiPedia)

[4] if $L_1, L_2 \in \mathbb{R}$, then $L_1 \cup L_2 = \{ w \mid w \in L_1 \vee w \in L_2 \} \in \mathbb{R}$.
[5] if $L_1, L_2 \in \mathbb{R}$, then $L_1 \cap L_2 = \{ w \mid w \in L_1 \wedge w \in L_2 \} \in \mathbb{R}$.
[6] if $L_1, L_2 \in \mathbb{R}$, then $L_1 \cdot L_2 = \{ w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2 \} \in \mathbb{R}$.
[7] 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 using * operation

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.

$$L = \{ 1, \cdots, 9 \} \cdot \left(\{ 0, 1, \cdots, 9 \}^* \cdot \{ , \} \right)^* \cdot \{ 0, 1, \cdots, 9 \}$$

Consider

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

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:

$$a\ digit \text{ and } a\ number\ of (a\ number\ of\ digits \text{ and } a\ comma) \text{ and } a\ digit$$

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).

A regex E can be transformed to language of DFA as follow

$$\begin{aligned} &E = a && L(a) = \{ a \} \\ &E = \varepsilon && L(\varepsilon) = \{ \varepsilon \} \\ &E = E_1 \cdot E_2 && L(E) = L(E_1) \cdot L(E_2) \\ &E = E_1 + E_2 && L(E) = L(E_1) + L(E_2) \\ &E = (E_1)^* && L(E) = L\left((E_1)^*\right) = L(E_1)^* \\ &E = \varnothing && L(\varnothing^*) \end{aligned}$$

In fact, the rule

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

can be replaced with

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

Example 4.2: $\mathbb{R}$ closed under star (see footnote[7])

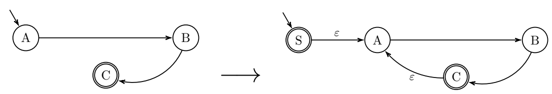
We have a regular language A_1 and want to prove that ${A_1}^*$ also is regular.

Proof.
We take an NFA N_1 for A_1 and modify if to N to recognize ${A_1}^*$.

We can construct N like N_1 with additional ε arrows returning to the start state from the accept states. This way, when processing gets to th end of a piece that N_1 accepts, the machine N has the option of jumping back to the start state to try to read in another piece that N_1 accepts.

We also need to add a new sart state and make it an accept state, and which has an ε arrow to the old start state. So it accept empty string $\varepsilon(n = 0)$.

See the graph for visualization:



N_1 N

One bad idea is simply to make original start state to final state. This approach certainly adds ε to the recognized language, but it may also add other, undesired stings.

□

Theorem 4.1 Equivalence of Regex and Regular Language.

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

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 ($\varnothing$ 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.
- there are no arrows entering the start state.

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

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

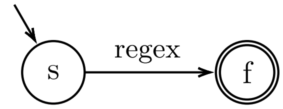
where

$$\delta \colon (Q - \{\, q_{accept} \,\}) \times (Q - \{\, q_{start} \,\}) \mapsto \mathbb{R}(\Sigma)$$

A GNFA, since it can use transitions with regexes, can help develop a regex of the language of the GNFA

To translate a NFA to a GNFA:

1. add a new start state with ε arrow connect to its odd start state,
2. removing states one at a time until we have the form



3. make $\{ q \in f \}$ to non-accept states, connect those original final states to a new and unique final state.

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

$$\begin{tikzcd}
\text{DFA} \arrow[r, leftrightarrow] & \text{NFA} \arrow[d] \\
\text{RegEx} \arrow[ur] & \text{GNFA} \arrow[l]
\end{tikzcd}$$

The general translation will be as follow: RegEx $\Rightarrow$ NFA $\Rightarrow$ DFA $\Rightarrow$ NFA $\Rightarrow$ GNFA $\Rightarrow$ RegEx

4.4 Non-Regular Languages

Are there languages which are not regular? The answer is obviously "Yes," but what are they? Take binary strings as example, we want to find $L \subseteq \{ 0, 1 \}^*$ s.t. $\forall$ 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 $0 = 000, 1 = 001, \cdot = 010, \cdots \emptyset = 111$ and use function φ to rewrite the regex above, [8]

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

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).

[8] φ stands for string.

Definition 4.2 Diagonalization:.

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

Proof of $\varphi(''\varnothing'') ='' \ 111'' \notin L(\varnothing) = \varnothing$.
Assume for controdiction L is regular,
Let E a regular expresstion s.t. $L(E) = L$.

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

which is a contradiction to our assumption. □

4.5 Pumping Lemma

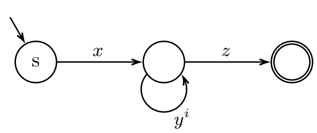
All regular language satisfies property P which is known as Pumping Lemma(P.L.), if a language L does not satisfy P then L is non-regular. To prove a non-regular language, we will first assume L is regular, then L must satisfies P ,but it doesn't, thus lead to controdiction.

Definition 4.3 Pumping Lemma Property P**.**

$$\exists P \geq 1 \qquad \textit{(pumping length)}$$
$$\forall w \in L \textit{ of length } |w| \geq P$$
$$\exists x, y, z \in \Sigma^*, w = x \cdot y \cdot z$$

and

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



DFA with a finite number of state, if read p symbols with p+1 states, there must be one state that is repeat.

Let's try to prove Non-regular language using the pumping lemma

Example 4.3: a regular language satisfy Pumping Lemma

$$L = \{ w \in \{0,1\}^* \mid w \text{ is a multiple of } 5 \}$$

Given a string 11110000 with $P = 5$, which means if a string is of length at least 5, then it can be pumped.

We can transform the string to xyz with $x = 1111, y = 0, z = 000$, so $|xy| = 5 \leq P$ satisfies the pumping lemma.

$$\begin{array}{c|c|c}
x & y^i & z \\
\hline
\overbrace{1111}^{=15} & \overbrace{0}^{2^i} & \overbrace{000}^{2^3}
\end{array} \Rightarrow 15 \cdot 2^1 \cdot 2^3,$$

Example 4.4: Example 1.75 on book P_{97}

let $L = \{ w \in \{0,1\}^* \mid w^{\text{reverse}} = w \}$ which is the same as $L = \{ ww \mid w \in \{0,1\}^* \}$, means $w \in L$ could be 00100 or 1001. $0011 \notin L$.

We show that L is nonregular, using the pumping lemma.

Proof.

Assume to the contrary that L is regular. Let p be the pumping length given by the pumping lemma. Let s be the string 0^p10^p1. Because $s \in L$ and s has length greater than p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, satisfying the three conditions of the lemma.We show that this outcome is impossible.

Condition 3 is crucial, because without it we could pump s if we let x and z be the empty string. With condition 3:

$$\underbrace{0 \cdots 0}_{p} 1 \underbrace{0 \cdots 0}_{p} 1$$

$|xy|$ must $\leq p$, makes y must consist only of 0s, so $xyyz \notin L$. □

Observe that it is critical to the proof that we chose the propriate string, as opposed to, say, the string 0^p0^p could not lead to a contradiction because it is a member of L and it can be pumped.

Example 4.5: Prove $P \notin$ Reg Using Pumping Lemma

let

$$P = \left\{ w \in \{ 1 \}^* \mid |w| \text{ is a prime number} \right\} = \{ 11, 111, 11111, \cdots \}$$

Claim: P is not regular.

Proof.

Assume for contradiction P is regular.
Therefore, by P.L., there is a "Pumping Length" $n \geq 1$ s.t. 1) 2) 3) in Pumping Lemma is true, which means 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.
$m = |x| + |y| + |z|$ is a prime.
By P.L.,

1. $\forall i \geq 0, \quad x \cdot y^i \cdot z = x \overbrace{yy \cdots y}^{i} z \in L,$
2. $|y| \geq 0 (y \neq \varepsilon),$
3. $|x| + |y| \leq n.$

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

.

$|x| + i|y| + |z|$ is also a prime according to the assumption. Let $i = m + 1$,

$$\begin{aligned} &\overbrace{|x| + |y| + |z|}^{m} + (i-1)|y| \\ &= m + (i-1)|y| \\ &= m + m|y| \\ &= m(1 + |y|) \not\subseteq \text{ prime number} \end{aligned}$$

Contradicting $xy^i z \in L$. $\square$

5 Finite State Transducers (FST)

Definition 5.1 Finite State Transducers (FST). *A FST is a* 5*-tuple*

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

where

- Q *is finite set of states,*
- Σ, Γ *are finite sets of symbols (*Σ *for input,* Γ *for output).*
- $s \in Q$ *is the start state, and*
- $\delta \colon Q \times \Sigma \mapsto Q \times \Gamma^*$ *is the transition function.*

Definition 5.2 .

$$\delta^*(q, w) \in Q \times \Gamma^*$$

is the extended transition function defined as such:

$$\begin{cases} \delta^*(q, \varepsilon) = (q, \varepsilon) & \\ \delta^*(q, aw) = (q'', uv) \quad \textit{if} & \\ & = \delta(q, a) = (q', u) \qquad \textit{and} \\ & = \delta^*(q', w) = (q'', v) \\ & \qquad\qquad = (\ \textit{where}\ a \in \Sigma, u, v, w \in \Sigma^* \end{cases}$$

Definition 5.3 f_M of and FST.

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

5.1 Transition between FST and DFA

We can combine an FST with a DFA, obtain a new DFA as result.

The DFA M' recognize regular language $L(M)'$, but the FST M does not recognize a language, instead it produces a transition function. In general, we use first machine FST to compute a input, and then fit the output (a string) into the second machine DFA to get a final state.

Theorem 5.1 Transmition theorem. $\forall$ FST $F = (\, Q_F, \Sigma, \Gamma, \delta_F, s_F \,)$,
$\forall$ DFA $D = (\, Q_D, \Gamma, \delta_D, s_D, F_D \,)$,

$\exists$ DFA $M = (\, Q, \Sigma, \delta, s, F\,)$ *s.t.*

$$\overbrace{L(M)}^{A} = f^{-1}(\overbrace{L(D)}^{B}) \textit{ meaning}$$

$$L(M) = \{\, w \in \Sigma^* \mid f_M(w) \in L(D)\,\}$$

$$f(A) = B \not\Leftrightarrow A = \{\, x \mid f(x) \in B\,\}$$

A is the inverse image of B $\rightarrow A = f^{-1}(B)$.

Proof of Theorem 5.1.
Let $Q = Q_F \times Q_D$,
$s \;= (s_F, s_D)$,
$F = Q_F \times F_D$.
Then the transition function $\delta \colon Q \times \Sigma \mapsto Q$ will be

$$\delta((q_F, q_D), a) = \big(q'_F, \delta^*_D(q_D, w)\big), \text{where } \delta_F(q_F, a) = (q'_F, w)$$

$\square$

Theorem 5.2 .

Definition 5.4 Reduction of languages.

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

is called a reduction from A to B. If such reduction exists, we say A is reducable 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,The contrapositive will be
If $A \leq_{\text{FST}} B$ and A is not regular, then B is not regular.