CSE 105: Computation

by Jy

Contents

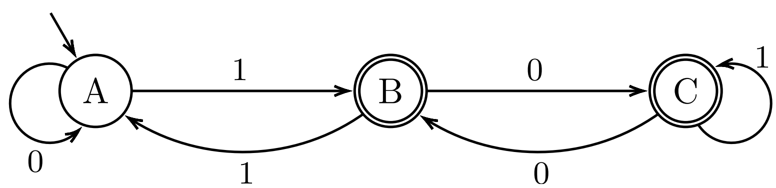
1 Deterministic Finite Automaton (DFA)

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?

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

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 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*

Σ *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 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 any DFA $M = (Q, \Sigma, \delta, s, F)$, let*

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

where Σ^* is a set of strings over Σ.

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

Definition 1.3 δ^*.

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

which is an inductive function defined as

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

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.*"*

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

defined as

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

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 Σ^* 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*

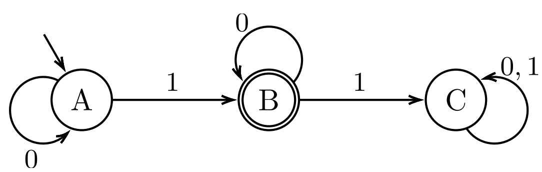
$$\exists \text{DFA} M \mid L(M) = L$$

Example 1.4

Given that $\varepsilon^* = \{\varepsilon\}$ and $\Sigma^* = \{\varepsilon, 1, 0, 10, 101, \cdots\} = \{0, 1\}^*$, 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 $L_1 \cap L_2 = \{ w \mid w \in L_1 \wedge \in L_2 \}$
Concatenation $L_1 \cdot L_2 = \{ w_1 \cdot w_2 \mid w \in L_1, w_2 \in L_2 \}$

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 $M' = \big(\, Q, \Sigma, \delta, s, F^C \,\big)$,
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 Σ are finite sets*
- $s \in Q, F \subseteq Q$
- $\delta \colon Q \times \Sigma_\varepsilon$[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 δ—the transition function of an NFA maps to a set of states instead of exactly one state as of a DFA.

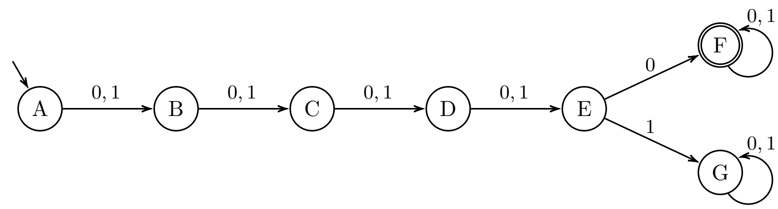
[1] $\Sigma_\varepsilon = \Sigma \cup \{\, \varepsilon \,\}$

Example 2.1: How many states do you need?

Given

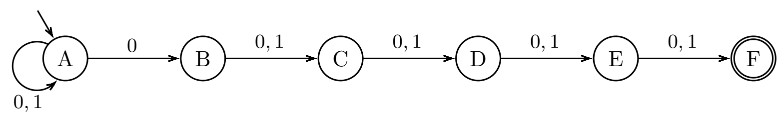
$$L = \{\, w \in \{\, 0, 1\,\}^* \mid \text{the 5th digit of } w \text{ is } 0 \,\}$$

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 ε transition, a transition that is performed without requiring any input.

Definition 2.2 Configurations of NFA.

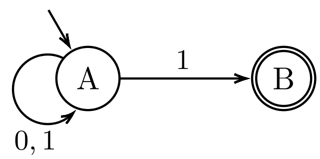
$$\begin{aligned}
\text{Conf} &= Q \times \Sigma^* \\
I(w) &= (s, w) \\
H &= Q \times \{\, \varepsilon \,\} \\
O(q, \varepsilon) &= \begin{cases} \text{True}, & q \in F \\ \text{False}, & \textit{else} \end{cases} \\
R &= \{\, (q, aw) \mapsto (q', w) \mid \forall q \in Q, a \in \Sigma_\varepsilon, w \in \Sigma^* \,\}
\end{aligned}$$

Definition 2.3 Language of NFA.

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

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 δ can be defined by the state transition table:

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

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		B

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 ε-transition.

input		1		0	
State sequence 1	A		B		?
State sequence 2	A		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 .

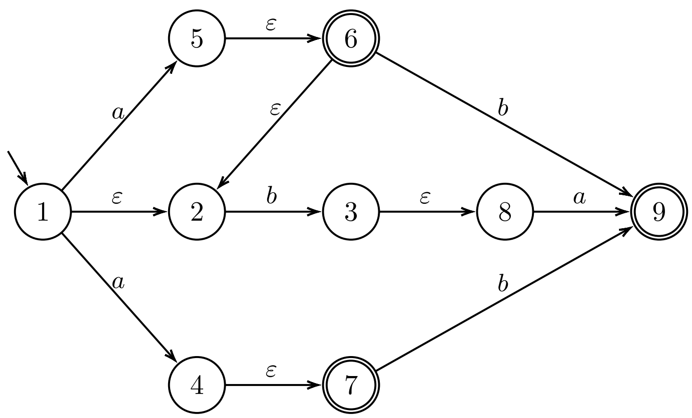
$$\forall \text{NFA} N = (\, Q, \Sigma, \delta, s, F \,),$$
$$\exists \text{DFA} M = (\, Q', \Sigma, \delta', s', F' \,) \; s.t. \; L(N) = L(M)$$

Proof of Theorem 2.1.

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

□

Example 2.3: Convert the following NFA to DFA



Start with the initial state, state 1, form a set of its ε-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 $\{\, \delta(1, a), \delta(2, a) \,\} = \{\, 4, 5, \varnothing \,\} = \{\, 4, 5 \,\}$, Let I_a be its ε-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.

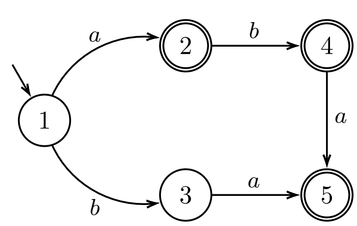
The following table neatly traces the whole process:

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

With the table, we use the ε-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,*

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

For a language,

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

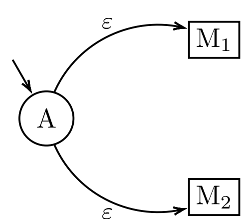
It is easy to find that

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

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

Recall Example 1.6, with NFA it is much easier to proof 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 ε arrows:



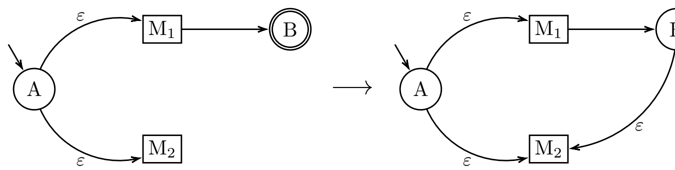
□

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 ε 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 ε 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, [2]
- intersection, [3]
- concatenation, and [4]
- star. [5]

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)

[2] if $L_1, L_2 \in \mathbb{R},$ then $L_1 \cup L_2 = \{ w \mid w \in L_1 \vee w \in L_2 \} \in \mathbb{R}$.
[3] if $L_1, L_2 \in \mathbb{R},$ then $L_1 \cap L_2 = \{ w \mid w \in L_1 \wedge w \in L_2 \} \in \mathbb{R}$.
[4] if $L_1, L_2 \in \mathbb{R},$ then $L_1 \cdot L_2 = \{ w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2 \} \in \mathbb{R}$.
[5] 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.

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

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:

$$adigit \text{ and } anumberof(anumberofdigits \text{ and } acomma) \text{ and } adigit$$

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

A regex E has the following rules

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

In fact, the rule

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

can be replaced with

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

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.

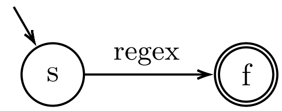
Definition 4.1 GNFA. *A GNFA is defined as a* 5*-tuple*

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

where

$$\delta \colon (Q \backslash \{ f \}) \times (Q \backslash \{ s \}) \mapsto \mathbb{R}(\Sigma)$$

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:

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

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. $\forall$ regex $E, L(E) \neq L$. With symbols $\{0, 1, \cdot, +(\cup), *, (,), \varnothing\}$, any regular language can be expressed,

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

Map each of the symbols to $0 = 000, 1 = 001, \cdot = 010, \cdots \varnothing = 111$ and use function φ to rewrite the regex above,

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

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

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

Proof of $\varphi(''\varnothing'') ='' 111'' \notin L(\varnothing) = \varnothing$. Assume for controdiction 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 controdiction.

Definition 4.2 Property P.

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

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