PROPOSITIONAL LOGIC

A proposition is a statement that is either true or false

example 1 $2+2=4$ is a proposition
$x+2=4$ is not a proposition

Truth Tables are used to plot the various outcomes for a particular statement below is an example for two staments p and q

P	Q	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$
1	0	0	1	0
1	1	1	1	1
0	1	0	1	1
0	0	0	0	1

NOT = $\neg$ AND = $\wedge$ IF ONLY IF = $\leftrightarrow$
OR = $\vee$ IF THEN = $\rightarrow$ EXCUSIVE OR = $\oplus$

ORDER OF OPERATIONS = $\neg \wedge \vee \rightarrow \leftrightarrow$

- ★ A statement is CONSISTENT if it is sometimes true.
- ★ A statement is INCONSISTENT if it is never true. otherwise known as a contradiction
- ★ A TAUTOLOGY is a statement that is always true
- ★ A propositional EQUIVALENCE is two statements with the exact same truth table

COMPOUND STATEMENTS are statements made up of multiple operators.

example 1 - prepare the truth table for $(P \wedge Q) \vee \neg r$

P	Q	R	$(P \wedge Q)$	$\neg r$	$(P \wedge Q) \vee \neg r$
1	1	1	1	0	1
1	0	1	0	0	0
0	1	1	0	0	0

NOTE: Not all options are shown here

De morgans law

- ★ $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
- ★ $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
- ★ $(P \rightarrow Q) \equiv (\neg P \vee Q)$
- ★ $(P \rightarrow Q) \equiv \neg Q \rightarrow \neg P$

Predicates and Quantifiers

A statement such as $x > 3$ or $x = y + 3$ are not propositions until variables are defined. The predicate is the variable properties eg >3 and the quantifier is the result of the variable.

We write statements like this $P = >3$ x = variable

eg $P(x) = x > 3$ or $P(4)$ = TRUE $P(2)$ = FALSE

You can then assign quantifiers to the statements

1. EXISTENTIAL $\exists$ some elements are true
2. UNIVERSAL $\forall$ all elements are true

example 1 $P(x) = x + 1 > x$ domain is all real numbers

$$\boxed{\forall x P(x) \text{ is TRUE}}$$

example 2 $P(x) = x^2 > 10$ for all real numbers. Since $2^2 > 10$ is false but $5^2 > 10$ is true we get

$$\boxed{\exists x P(x) \text{ is TRUE}}$$

To negate such statements you do not have to provide a complete opposite but simply one example that exists that isn't true. eg to negate the statement "ALL MEN WEAR HATS" you simply need to show that 'SOME MEN DON'T WEAR HATS'

example $\forall x P(x)$ = ALL MEAN WEAR HATS

$\exists x \neg P(x)$ = SOME MEAN DON'T WEAR HATS

It is also worth noting that 'some men don't wear hats' is equivalent to 'not all mean wear hats':

$$\neg \forall x P(x) \equiv \exists x \neg P(x)$$

PROOFS

A proof is a connected sequence of logical statements that prove something is true.

While a computer can confirm if something is true or not within a range it cannot make the logical leap to make the statement that is always true

DIRECT PROOFS is where you try to prove the statement is true directly. For example $P \to Q$ is only ever false if Q is false and P is true. If you can show that that never occurs then you have directly proved the statement

example 1 if n^2 is odd then n^2 is odd

let $n = 2k + 1$ where k is an integer any number multiplied by 2 is even then add 1 makes it odd.

$$n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = \underbrace{2(2k^2 + 2k)}_{\text{anything multiplied by 2 is even}} + 1$$

plus 1 at the end means it is odd

THEREFORE THE STATEMENT IS TRUE AND PROVEN DIRECTLY

INDIRECT PROOF otherwise known as a contrapositive proof. We first find a statement that has a propositional equivalence eg $P \to Q \equiv \neg Q \to \neg P$. If you cannot prove the first statement is true then you can try and prove the second statement is true

example 1 if n^2 is odd the n is odd, to prove indirectly we would try and show if n^2 is even then n is even.

let $n = 2k$

$$n^2 = (2k)^2 = 4k^2 = 2(2k^2)$$

↳ any number multiplied by 2 is even

PROOF BY CONTRADICTION is where you assume the opposite of your original statement. If you can prove the contradiction is true then the original statement is false and vice versa.

example 1 the $\sqrt{2}$ is an irrational number, we take the negative which is $\sqrt{2}$ is a rational number

A irrational number is one that can be expressed as an fraction

$\sqrt{2} = \frac{a}{b}$ or $2 = \frac{a^2}{b^2}$

We can rearange this to get $2b^2 = a^2$ since $2b^2$ is even then a^2 is also even. If a number is divided by 2 is not irrational, ~~be~~

THEREFORE $\sqrt{2}$ IS AN IRRATIONAL NUMBER

PROOF BY INDUCTION

for certain statements you may not automatically have a true false statement eg $x + > 4$ induction is used to show $\forall n\, P(n)$.

1 BASIS STEP show that $P(1)$ is TRUE
2 INDUCTIVE STEP show if $P(k)$ is TRUE then $P(k)+1$ is TRUE

Example 1 show that $1 + 2 + 3 \ldots + n = \frac{n(n+1)}{2}$

STEP 1 - BASIS STEP $\frac{1(1+1)}{2} = 1$ this is TRUE

STEP 2 - INDUCTIVE STEP

We assume $1 + 2 + 3 \ldots + n = \boxed{\frac{n(n+1)}{2}}$

And $\boxed{1 + 2 + 3 \ldots + n} + (n+1) = \frac{(n+1)(n+1+1)}{2}$

Substituting the values we get

$$\frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+1+1)}{2}$$

Simplifying we get $n^2 + 2n + 2 = n^2 + 2n + 2$

PROVED TRUE!

Combinatorial Principles

PERMUTATIONS

is where the order matters, aka how many different ways can you arrange something

eg 5 people in a queue $= 5 \times 4 \times 3 \times 2 \times 1$ or $5!$ ways

example 1, out of 5 people how many pairs can you make in a queue

$$\frac{5!}{(5-2)!} = \frac{5 \times 4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = 5 \times 4$$

PERMUTATION WITH REPETITION

eg a lock combination, this is the most straightforward

example 2 - 4 digit lock with 9 numbers
$9 \times 9 \times 9 \times 9$ or 9^4

PERMUTATION WITHOUT REPITITION

eg top 3 finishes in a race of 10 people

example 3 $= \frac{10!}{(10-3)!} = \frac{10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1}{7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1} = 10 \times 9 \times 8$

example 1. A password must be between 6-7 characters long, it is made up of the lowercase alphabet and 0-9. It must contain at least 1 letter.

5 CHARACTERS

Step 1: Total digits available for selection $(26+10)^5$
Step 2: Identify non-qualifying passwords 10^5
Step 3: $36^5 - 10^5 = 60{,}366{,}176$ permutations
Step 4: Repeat for 6 + 7 characters and then add together

example 2 How many ways can MISSISSIPPI be arranged? The order of repeating letters can be ignored.

M I S P
I S P
I S
I S
4! 4! 2!

$11! \div (4!\,4!\,2!)$

example 3 How many ways can you arrange $\{a, b, c, d, e, f, g\}$ containg the word bad?

STEP 1: length 3 = 1 option 'bad'
STEP 2: length 4 = 4 available letters × 2 on either side
aka 'xbad' or 'badx'
STEP 3: length 5 = 4 letters ~~×3~~ × 3 × 3
aka 'xxbad' 'xbadx' 'badxx'
STEP 4: length 6 = 4 × 3 × 2 × 4
STEP 5: Add the options together to get the answer

Think of it as though you are reducing the set to $\{c, e, f, g\}$ and cycle bad through the gaps eg 5 characters:

xxbad, xbadx, badxx then you have
4×3 or $\frac{4!}{2!}$ letters to fill the x's
$4 \times 3 \times 3$ or 36 ways for 5 characters

COMBINATIONS

where the order doesn't matter but it's still to do with the arrangements of the things. For example Top 3 finishers of a race, the order doesn't matter, just the top 3

➔ **COMBINATION WITH REPITITION** eg scoops of icecream you can have 3 scoops, and there are 5 flavours. You can have any combination you'd like

General formula $\frac{(r+n-1)!}{r!(n-1)!}$ where n = total to choose from
r = number of choices

Using the icecream example this becomes

$$\frac{(5+3-1)!}{3!(5-1)!} = \frac{7!}{3!4!}$$

➔ **COMBINATION WITHOUT REPITITION** is much more straight forward. Treat it as a permutation and then divide it by the number of ways the final selection can be ordered. eg top 3 runners from a group of 100 runners:

$\frac{100!}{(100-3)!} = 100 \times 99 \times 98$ then 3 runners can be arranged in 3! ways
$(100 \times 99 \times 98) / 3$

example 1: How many binary strings of length 8 contain equal numbers of 1 and 0?

Does the order matter? NO, so it is a combination
Do the items repeat? YES, but see logic below

We have 4 1's and there are 8 available spaces, if order mattered we would arrange these $8 \times 7 \times 6 \times 5$ ways. As order doesn't matter we divide it by $4 \times 3 \times 2 \times 1$ ways, or 70 possible ways

Pidgeon Hole Theory

If you have k boxes and $k+1$ items then at least 1 box must contain at least $\lceil N/k \rceil$.

eg if there are 367 students in a class then at least 2 have the same birthday

eg if you pick 8 countries at random at least 2 are on the same continent.

eg If we have a deck of cards how many must you pick in order to get 4 of the same suit?
= you can pick 3 cards from each suit (3×4) but the moment you pick the 13^{th} you have 4

AUTOMATA *theory*

A set is a collection of unordered and distinct objects called elements.

IMPORTANT
1 There is no order in a set
2 Repeated elements are listed once
3 Sets can be finite or infinite

IMPORTANT SYMBOLS

$\in$ = inclusion $\emptyset \in A$ "$\emptyset$ is an element in A"
$\notin$ = exclusion $\emptyset \notin A$ "$\emptyset$ is not an element in A"
$|B|$ = number of elements in set B
Σ = An alphabet
ε = an empty string
$\emptyset$ = an empty set

The symbol Σ^* is all of the strings that can be made from a language Σ

eg $\Sigma = \{0, 1\}$ $\Sigma^* = \{\varepsilon, 0, 1, 00, 11, 10, 01 \ldots\}$

If we don't want an empty string then we use Σ^+. Sometimes we might be interested only in strings of a particular length k. we would write this as Σ^k

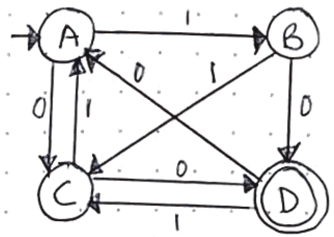
eg $\Sigma^2 = \{00, 11, 10, 01\}$
$\Sigma^3 = \{000, 111, 101, 011, 110 \ldots\}$

Using permutation rules we would say that there are $2 \times 2 \times 2$ or 2^3 permutations for $\Sigma^k = |\Sigma|^k$

WHAT IS AN AUTOMATON

An automaton is a simplistic computer with limited memory. A 5 tuple automaton has the following components

1. States **Q** - A finite set
2. Alphabet Σ - a finite set
3. Transitions δ - state + alphabet moves to another state
4. Starting State q_0 where $q_0 \in Q$
5. Accept state **F** where F is a subset $\subseteq$ of Q



In this example
$Q = \{A, B, C, D\}$
$\Sigma = \{1, 0\}$
$q_0 = A$
$F = \{D\}$
δ =

	0	1
A	C	B
B	D	C
C	D	A
D	A	C

For the string to be accepted it must end on an accept state
eg $110 = A \xrightarrow{1} B \xrightarrow{1} C \xrightarrow{0} D$

The set of all the strings accepted by the automaton is called the LANGUAGE. If M has an alphabet Σ then $L(M)$ is the language.

$$L(M) = \{ \underbrace{x \in \Sigma^*}_{\text{strings}} \mid \underbrace{M \text{ accept } x}_{\text{criteria}} \}$$

DETERMINISTIC FINITE AUTOMATON

A DFA is an automaton where

- For each state there is one transition only
- There is a unique starting state

If either state is not met then it is not a DFA

LANGUAGES

Operations on sets can be done as follows:

UNION $A \cup B = \{x \mid x \in A, x \in B\}$

eg $A = \{red, green, pink\}$ $B = \{apple, banana, kiwi\}$

$A \cup B = \{red, green, pink, apple, banana, kiwi\}$

1 $A \cup B = B \cup A$
2 $(A \cup B) \cup C = A \cup (B \cup C)$
3 $A \cup B \cup \emptyset = A \cup B$
4 $A \cup A = A$

CONCATENATION $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

eg $A \circ B = \{$redapple, redbanana, redkiwi, green apple, greenbanana...$\}$

1 $(A \circ B) \circ C = A \circ (B \circ C)$
2 $A \circ \varepsilon = \varepsilon \circ A = A$
3 $A \circ \emptyset = A$
4 $A \circ B \neq B \circ A$

COMBINING OPERATIONS combining union and concat

1 $(A \cup B) \circ C = (A \circ C) \cup (B \circ C)$
2 $A \circ (B \cup C) = (A \circ B) \cup (A \circ C)$

KLEENE STAR

1 $\emptyset^* = \emptyset$
2 $\varepsilon^* = \varepsilon$
3 $(A^*)^* = A^*$
4 $A^*A^* = A^*$
5 $(A \cup B)^* = (A^*B^*)^*$

COMPOUND REGULAR EXPRESSIONS

CONCATENATION = if R_1 and R_2 are regular expressions then so is $R_1 \circ R_2$.
UNION = if R_1 and R_2 are regex then so is $R_1 \cup R_2$
KLEENE STAR = if R is regex then R^* is also

examples $ab^* = \{a, ab, abbb, \dots\}$
$ab^* \cup b^* = \{a, ab, abbb \dots\} \cup \{\varepsilon, b, bb \dots\} = \{\varepsilon, a, b, ab, bb\}$
$ab^+ \cup b^+b = \{ab, abb \dots\} \cup \{bb, bbb \dots\} = \{ab, bb, abb, bbb\}$

Using a binary alphabet $\Sigma = \{a, b\}$

1 $\Sigma^* = \{\varepsilon, a, b, ab, ba, aa, bb \dots\}$
2 $\Sigma^* a = \{a, aa, ba, aaa, aba \dots\}$ the same as above ending in a.
3 $\Sigma^* a \Sigma^* = \{a, aa, ba \dots\}$ all combinations with at least 1 a.

ORDER OF OPERATIONS $*$, CONCATENATION, UNION

for example $a \cup bc^* = a \cup b(c^*) = a \cup (b(c^*))$

$\{a, bc, bcc, bccc \dots\}$

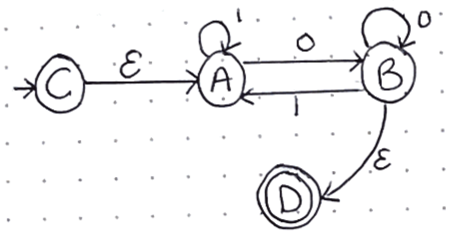
Building a regular expression from a string can be done as follows:
$\{bb, abba, abb, bba, bbb, aabb, abbb, babb\}$

1. We can see bb is present in all strings
2. It can have an empty string b or a in front of it
3. It can have an empty string b or a after it.
4. $(a \cup b)^* bb (a \cup b)^*$ or $\Sigma^* bb \Sigma^*$

KLEENES THEORUM ⇨ is a language is regular only if it can be described by a regular expression

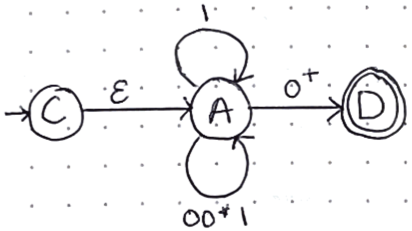
Regex to finite automaton

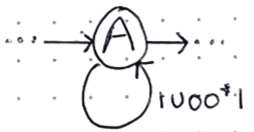
If there is a finite automaton it can be represented by a regular expression



Consider the automaton to the left to find the regex we start to remove transitions

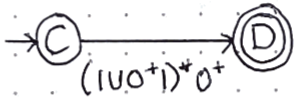
Consider state B, the path always comes from A along a 0, it then loops around 0 a number of times before outputting a 1 or ε.



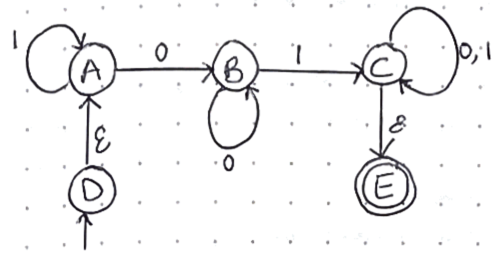


The loops going around A can just be unionised to $1 \cup 00^*1$
(00^* can also be written as 00^+)

We then consider the transition from $C \to A \to D$ and combine the ingoing, outgoing and looping expressions

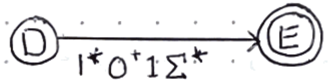


example 2

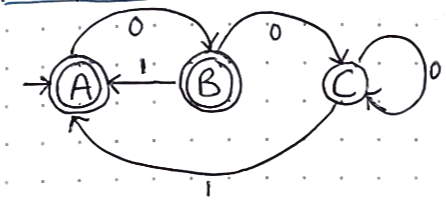


Step 1 → look at removing C. We have an incoming 1. Then we either have a 1 or 0 to loop indefinitely $(1 \cup 0)^*$ so we can replace it with the expression $1(1 \cup 0)^*$ or $1\Sigma^*$

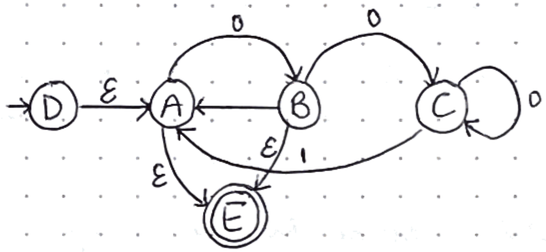
We then look at removing B. Which is an incoming 0 with multiple loops of 0 or 00^* or 0^+ and then A is a loop of 1 or 1^* we combine this to $1^*0^+1\Sigma^*$



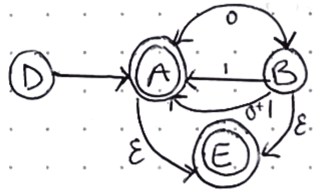
Example 3



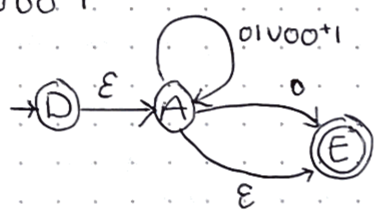
Step 1 → create a new initial state with an ε transition to A.
Step 2 → Create a final state connecting the two current final state.



Then we first look at removing state C. It has an incoming transition of 0, repeats 0 on loop and exits on 1, so we get 00^*1 or 0^+1



Then we look at removing B. This must cover the transition from $B \to E$ and $B \to A$. For $B \to A$ we have an incoming of 0 and and an outgoing of 1 and 0^+1 so we get $01 \cup 00^+1$



Removing A we then get

$(01 \cup 00^+1)^*(\varepsilon \cup 0)$

PUMPING LEMMA

Is a way of helping to prove if a language is regular or not. A regular language is regular or not if it is one that can be:

- ★ Accepted by a finite automaton
- ★ Can be explained by a regular expression
- ★ Every DFA language is regular

One way of identifying if it is regular is to break it down into its component parts and then build it back up again.

CLOSURE PROPERTIES

1. $L_1 \cup L_2$ the union is regular if L_1 and L_2 are regular
2. $L_1 \cap L_2$ the intersection of L_1 and L_2
3. $L_1 L_2$ the product of L_1 and L_2
4. L^* the kleene star of L_1
5. $U - L_1$ $(\Sigma^* - L_1)$ or the compliment of L_1

example prove $L = \{x \in \{a,b\}^* \mid \# a \text{ in } x = \# b \text{ in } x\}$ is not regular
$L = \{ab, aabb, abab, abba\}$

STEP 1 - Assume that L is regular
STEP 2 - We know $L' = \{x \in a^+ b^*\}$ is regular
STEP 3 - If L' and L is regular then $L \cap L'$ is regular
STEP 4 - $L \cap L = \{a^n b^n \mid n \in \mathbb{N}\}$ n number of a's and n number of b's
STEP 5 - $\{a^n b^n \mid n \in \mathbb{N}\}$ is not a regular language therefore L is not regular.

The pumping lemma theory states that all regular languages have a special property.
This special property is all strings in the language can be pumped if they are longer or equal to the pumping length. eg the string has a section that when repeated and still be in the language

GRAMMAR →

Grammar is a set of rules for connecting strings, it is another way of representing a language.
It consists of:

- ★ V variables, a finite set of symbols
- ★ Σ terminals, a finite set of letters
- ★ R rules a finite set of mappings each rule being a variable and a string of variables and terminals
- ★ S start variable, member of V

example 1
S = Variable
S = Start variable
a, b = terminals

rules: $S \Rightarrow bSa$
$\boxed{S} \Rightarrow \boxed{ba}$

We can generate strings by substitution $S \Rightarrow b\boxed{S}a \Rightarrow b\boxed{ba}a$

$S \Rightarrow bSa \Rightarrow bbSaa \Rightarrow bbbaaa$

example 2
S, T = variables
S = start variables
a, b = terminals

$S \Rightarrow aS \mid T$
$T \Rightarrow b \mid \epsilon$

find 3 strings derived from S

$S \to aS \to aaS \to aab$
$S \to aS \to a$
$S \to T \to b$

Language of Grammar

If $G = (V, \Sigma, R, S)$ the $L(G) = \{W \in \Sigma^* \mid S \Rightarrow^* W\}$

eg the language is the set of all words in sigma star such that W can be derived from S.

example 1, what is the languge of the following grammar

$S \Rightarrow bSa$
$S \Rightarrow ba$

STEP 1 → what does the language look like? ba, bbaa, bbbaaa...

$L(G) = \{b^n a^n \mid n > 1\}$

Therefore $G = (S, \{a, b\}, R, S)$ where $R = \{S \to bSa, S \to ba\}$

variable | language | start variable

BUILDING A GRAMMAR

Example, all binary strings with an even number of 0's.

STEP 1 → If the first is a 1 then it must be followed by an even number of 0s. We shall call these A. So $S \to 1A$ where A = even number of 0's.

STEP 2 → If the first letter is 0, then it can have strings before and after it but it must contain another 0 so $S \to 0A0B$ where A and B = even number of 0s.

STEP 3 → we then replace A and B with S to get

$\boxed{S \to 1S \mid 0S0S \mid \varepsilon}$

example 2 all strings of the form 0^+1^+

STEP 1 → all the strings starting with 0^+ are $U \to 0U \mid 0$
Step 2 → all the strings starting with 1^+ are $V \to 1V \mid 1$

we combine these to say

$S \to UV$ $\{0, 1, 01, 00, 11, 000, 011...\}$
$U \to 0U \mid 0$ $\{0, 00, 000, 0000...\}$
$V \to 1V \mid 1$ $\{1, 11, 111, 1111...\}$

example 3. design a context free grammar for $\{a^m b^n \mid n \geq m\}$ aka all strings where the number of b's is greater than or equal to the number of b's

STEP 1 decompose the string, if $n \geq m$ we can say it will be the same as

$a^m b^n = a^m \times b^{n-m} \times b^m$

STEP 2 if $n - m = 0$ then we get $a^m b^m$ this can be written as

$S \to aSb \mid \varepsilon$ $\{ab, aabb, aaabbb...\}$

STEP 3 if $n - m > 0$ then we get b^i where $i = n - m$.

$U \to bU \mid b$

Context free Grammar

COMSKY NORMAL FORM is where

- ★ Every rule is of the form $S \to XU$
  $S \to a$
- ★ Where a is any terminal
- ★ Where XUS are non-terminals
- ★ X and U are not the start variables
- ★ $S \to \varepsilon$ is permitted if S is the start variable

An example that is NOT chomsky normal form is :

1. $S \to 1S \mid 0S0S \mid \varepsilon$, because S appears on the right
2. $V \to \varepsilon$, if V is not the start variable it cannot go to ε
3. $U \to V$, where V is a variable
4. $X \to 1UV$, where length of the rule is ≥ 3

you can convert something into chomsky normal form by doing the following

1. $S \to 1S \mid 0S0S \mid \varepsilon$ becomes $S_0 \to S$

$$S \to 1S \mid 0S0S \mid \varepsilon$$

2. $V \to aU$ and $U \to a\varepsilon$ becomes ~~Vto~~ $V \to aU \mid a$ and $U \to a$ and ε is eliminated

3. Remove $A \to B$ eg $S \to YU$, $U \to Y \mid a$, $Y \to UY \mid b$ becomes $S \to YU$, $U \to a \mid UY \mid b$, $Y \to UY \mid b$ as we wanted to remove $U \to Y$ we just substitute the values

TURING MACHINE

A turing machine consists of $(Q, \Sigma, \Gamma, \delta, q_1, q_{acc}, q_{rej})$

- Q a finite set a states
- Σ input alphabet, a subset of Γ
- Γ is the alphabet including blank symbols
- δ is the transition left or right
- q_1 is the start state
- q_{acc} the accept state
- q_{rej} the reject state

TRANSITION function doesn't just move left or right but also takes the state and the letter and returns

→ state of the automaton
→ a letter to write into the current cell
→ a direction to move either left or right

NOTATION EXAMPLES

$b \to \square, R$ input is b, overwrite $\square$ blank move R right
$a \to b, R$ input a, write b move R right

You can imagine a turing machine as a FSA with a tape head, where the tape head provides the input string but can also be overwritten. The process is

STEP 1 → Tape head is on the first input character
STEP 2 → It reads the character
STEP 3 → Overwrite the character (it can be blank)
STEP 4 → Moves tape head left or right
STEP 5 → Changes states
STEP 6 → As soon as it hits accept or reject it terminates

Then follows sorting algos covered in Algos + data 1