Lecture 1

EVEN-EVEN

↳ each a,b occure even number of times

↳ eg. ε, aa, bb, abab, abbabb, abba, baab ...

DOUBLEWORD

↳ concatenating some string with itself

↳ eg. ε, aa, bb, aaaa, abab, baba, abbabb ...

↳ $\varepsilon\varepsilon = \varepsilon$

PALINDROMES

↳ same forward, backward

↳ eg. ε, a, b, aba, bab, bbb, baab, aaaa ...

Theorem : DOUBLEWORD $\subseteq$ EVEN-EVEN (subset)

member of DOUBLEWORD = member of EVEN-EVEN

Proof : let $w \in$ DOUBLEWORD

$w = xx$ for some x where x is a word

no. of a in w = 2(no. of a in x) ⇝ EVEN

no. of b in w = 2(no. of b in x) ⇝ EVEN

HENCE, $w \in$ EVEN-EVEN

LECTURE 2

proposition → statement
→ TRUE / FALSE

eg. ① TRUE proposition statement
↳ $1+1=2$

② FALSE proposition statement
↳ Earth is flat

③ proposition
↳ It will snow tommorow
↳ neither true or false → can't be confirmed because it's NOT YET tommorow

④ NOT proposition
↳ come and work for us ! } no truth value
↳ this is a false statement
↳ what is the statement ? how can one know it is a false statement

logical operations

↳ NOT	$\neg$	negation	it is not the case that
↳ AND	$\wedge$	conjunction	both ... and
↳ OR	$\vee$	disjunction	either ... or ... (or both) ... at least one of ...
↳ implies	$\Rightarrow$	implication	if ... then ..., implies, only if, sufficient
↳ equivalence	$\Leftrightarrow$		... if and only if ...
↳ some	$\exists$	existential quantifier	there exist x such that ...
↳ all	$\forall$	universal quantifier	for all x ...

logically equivalent
↓
same meaning, different form

tautology

↳ always TRUE statement, logically equivalent
↳ eg. $\neg(P \wedge Q) = \neg P \vee \neg Q$ → proof using truth table

Distributive law

- $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$

De morgan law

- $\neg(P \vee Q) = \neg P \wedge \neg Q$
- $\neg(P \wedge Q) = \neg P \vee \neg Q$

Disjunctive Normal Form (DNF) ⇒ disjunction of conjunctions of literals

A	B	OUTPUT	
T	T	T	$X \wedge Y$
T	F	F	* WHY?
F	T	T	$\neg X \wedge Y$
F	F	T	$\neg X \wedge \neg Y$

} ONLY TRUE OUTPUT STATEMENTS

conjunction conjunction conjunction

$P = (\neg X \wedge \neg Y) \vee (\neg X \wedge Y) \vee (X \wedge Y)$

disjunction

Conjunctive Normal Form (CNF) ⇒ conjunction of disjunction of literals

"clause"

① at least one of H, E, R, G — $H \vee E \vee R \vee G$

② A only if have N — $A \Rightarrow N$

③ none or both F, O — $F \Leftrightarrow O$

④ NOT MORE THAN ONE OF V, B, D — $(\neg V \vee \neg B) \wedge (\neg V \vee \neg D) \wedge (\neg B \vee \neg D)$

① ∧ ② ∧ ③ ∧ ④ = $(H \vee E \vee R \vee G) \wedge$

$(\neg A \vee N) \wedge$

$(\neg F \vee O) \wedge (F \vee \neg O) \wedge$

$(\neg V \vee \neg B) \wedge (\neg V \vee \neg D) \wedge (\neg B \vee \neg D)$

LECTURE 3

Predicate → statement
→ values : TRUE / FALSE } OUTPUT

** Quantifiers use only with variables

$\exists x$: computer (x) $\wedge$ human (x)
- some computer is human
- there exist a human computer
- there exist something that is both human & computer
- incorrect: $\exists x$: computer(x) $\Rightarrow$ human(x)
  - implies that there's something that's not human or computer
    - both together and individually (thinking wise)

$\forall x$: computer (x) $\Rightarrow$ human(x)
- every computer is human
- for everything, if it's a computer, then its human
- everything that's computer is also human
- incorrect : $\forall x$: computer(x) $\wedge$ human(x)
  - everything is both computer & human
    - implies that all human is a computer
      - implies that everything is a human computer

logic with quantifiers

- $\forall x (p(x) \wedge q(x))$ logically equivalent $\forall x\, p(x) \wedge \forall x q(x)$
- $\exists x (p(x) \vee q(x))$ logically equivalent $\exists x\, p(x) \vee \exists x q(x)$
- $\neg\forall x$ logically equivalent $\exists x \neg$
  - not everyone is happy
  - there exists some unhappy person
- $\neg\forall x \neg$ logically equivalent $\exists x$
- $\neg\exists x \neg$ logically equivalent $\forall x$

Assessed preparation: Question 1

1)(a) $V_{0,1}$

(b) $\neg V_{0,2}$

(c) $V_{0,1} \wedge \neg V_{0,2} \wedge \neg V_{0,3} \wedge \neg V_{0,4}$

(d) $V_{5,1} \vee V_{5,2} \vee V_{5,3} \vee V_{5,4}$

(e) $\neg V_{5,1} \wedge \neg V_{5,3}$

(f) $\neg(V_{5,1} \wedge V_{5,2}) \wedge \neg(V_{5,1} \wedge V_{5,3}) \wedge \neg(V_{5,1} \wedge V_{5,4}) \wedge \neg(V_{5,2} \wedge V_{5,3}) \wedge \neg(V_{5,2} \wedge V_{5,4}) \wedge \neg(V_{5,3} \wedge V_{5,4})$

(g) $(V_{5,1} \wedge \neg V_{5,2} \wedge \neg V_{5,3} \wedge \neg V_{5,4}) \vee (\neg V_{5,1} \wedge V_{5,2} \wedge \neg V_{5,3} \wedge \neg V_{5,4}) \vee (\neg V_{5,1} \wedge \neg V_{5,2} \wedge V_{5,3} \wedge \neg V_{5,4}) \vee (\neg V_{5,1} \wedge \neg V_{5,2} \wedge \neg V_{5,3} \wedge V_{5,4})$

(h) $(V_{3,2} \wedge V_{4,1}) \vee (V_{3,2} \wedge V_{4,3})$

(i) total number of clauses required for the CNF expression

= sum of clauses in each t from 1 to 3

$t=0 \Rightarrow \neg V_{0,1}, \neg V_{0,2}, \neg V_{0,3}, \neg V_{0,4}$

$t=1 \Rightarrow (V_{0,1} \wedge \neg V_{1,2}), (\neg V_{0,1} \wedge V_{1,2} \wedge \neg V_{1,3}), (\neg V_{1,2} \wedge V_{1,3} \wedge \neg V_{1,4}), (\neg V_{1,3} \wedge V_{1,4})$

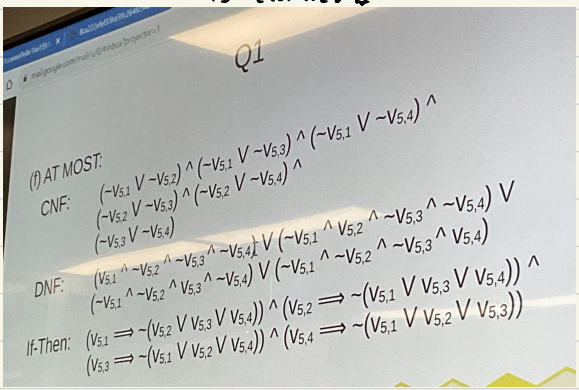
$t=2 \Rightarrow (V_{1,2} \wedge \neg V_{2,3}), (\neg V_{1,2} \wedge V_{2,3} \wedge \neg V_{2,4}), (\neg V_{2,3} \wedge V_{2,4})$

$t=3 \Rightarrow (V_{2,3} \wedge \neg V_{3,4}), (\neg V_{2,3} \wedge V_{3,4})$

HENCE, sum of clauses in each t from 1 to 3

$= 4 + 4 + 3 + 2$

$= 13$ clauses ✱



5) (a) ¬Judith ∨ ¬margaret

(b) ¬Judith ∧ ¬margaret

(c) Judith ∨ margaret ∨ katherine

(d) (¬Judith ∨ ¬margaret) ∧ (¬Judith ∨ ¬katherine) ∧ (¬katherine ∨ ¬margaret)

(e) (Judith ∨ margaret ∨ katherine) ∧ (¬katherine ∨ ¬margaret) ∧ (¬Judith ∨ ¬katherine) ∧ (¬Judith ∨ ¬margaret)

(f) (Judith ∨ margaret) ∧ (Judith ∨ katherine) ∧ (margaret ∨ katherine)

(g) ¬Judith ∨ ¬margaret ∨ ¬katherine

(h) (Judith ∧ margaret ∧ ¬katherine) ∨ (Judith ∧ ¬margaret ∧ katherine) ∨ (¬Judith ∧ margaret ∧ katherine)

(i) Judith ∧ margaret ∧ katherine

(j) ¬Judith ∧ ¬margaret ∧ ¬katherine

7) $(a \vee b) \wedge (\neg a \vee \neg b) \wedge$
$(a \vee c \vee d) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge (\neg c \vee \neg d) \wedge$
$(b \vee c \vee e) \wedge (\neg b \vee \neg c) \wedge (\neg b \vee \neg e) \wedge (\neg c \vee \neg e) \wedge$
$(d \vee e) \wedge (\neg d \vee \neg e)$

① ② ③ ④ — one for each vertex

at least — at most

Question 7 relevant to the assignment !

2) let $x = \{a,b\}^*$

let n = num of a's in x

let m = num of b's in x

case 1 : let $w = xx$

$2n$ = num of a's

$2m$ = num of b's

$w \in \overline{ODD\text{-}ODD}$

$PALINDROME \in \overline{ODD\text{-}ODD}$

$PALINDROME \subseteq \overline{ODD\text{-}ODD}$

* $C \Rightarrow$ proper subset

case 2 : let $w = xyx^{\leftarrow}$

if $y = a$ $w = xax$

$2n+1$ number of a's

$2m$ number of b's

$w \in \overline{ODD\text{-}ODD}$

if $y = b$ $w = xbx^{\leftarrow}$

$2n$ number of a's

$2m+1$ number of b's

$w \in \overline{ODD\text{-}ODD}$

8) (i) taller (father (claire) , taller (father (max) , max))

taller (father (max) , max) $\wedge \neg$ taller (father(max), father(claire))

(ii) $\exists X$ taller(x, father(claire))

(iii) $\forall X \exists Y$ taller(X, Y)

(iv) $\forall X$ (taller(x, claire) $\rightarrow$ taller(x, max))

* $\exists$ = some , $\forall$ = all

Lecture 4

Prove $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$ is a tautology

tautology proving : demonstrate true for all possible truth value

type of proof : proof by contradiction

↳ assume it's not a tautology

↳ assign truth / false values and find which = false

P	Q
T	T
T	F
F	T
F	F

① TRUE - TRUE

$(P \wedge (P \Rightarrow Q)) \Rightarrow Q$

$(\text{TRUE} \wedge (\text{TRUE} \Rightarrow \text{TRUE})) \Rightarrow \text{TRUE}$

$(\text{TRUE} \wedge \text{TRUE}) \Rightarrow \text{TRUE}$

$\text{TRUE} \Rightarrow \text{TRUE}$

TRUE ✱

② TRUE - FALSE

$(\text{TRUE} \wedge (\text{TRUE} \Rightarrow \text{FALSE})) \Rightarrow \text{FALSE}$

$(\text{TRUE} \wedge \text{FALSE}) \Rightarrow \text{FALSE}$

$\text{FALSE} \Rightarrow \text{FALSE}$

TRUE ✱

③ FALSE - TRUE

$(\text{FALSE} \wedge (\text{FALSE} \Rightarrow \text{TRUE})) \Rightarrow \text{TRUE}$

$(\text{FALSE} \wedge \text{TRUE}) \Rightarrow \text{TRUE}$

$\text{FALSE} \Rightarrow \text{TRUE}$

TRUE ✱

④ FALSE - FALSE

$(\text{FALSE} \wedge (\text{FALSE} \Rightarrow \text{FALSE})) \Rightarrow \text{FALSE}$

$(\text{FALSE} \wedge \text{TRUE}) \Rightarrow \text{FALSE}$

$\text{FALSE} \Rightarrow \text{FALSE}$

TRUE ✱

∴ For all possible cases, it is proven that $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$ is proven to be held TRUE for all possible truth values for P and Q. This shows that the expression $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$ is indeed a tautology.

type of proofs: ① proof by construction
② proof by cases ---------→ proof by exhaustion (brute force)
③ proof by contradiction ---→ proof by example
④ proof by induction

proof by contradiction
↳ assume negation of the statement
↳ contradict yourself
↳ proof yourself to be wrong and hence
"the negation of your negation of the statement shall be held true"

De morgan's laws proofing
↳ slide 04 15-17/27

proof by mathematical induction
eg. prove $S(n)$ holds for every natural number n
IF induction basis AND inductive step IS TRUE,
THEN statement IS ALSO TRUE
* induction basis : $S(1)$ is TRUE
* inductive step : $S(k+1)$ is TRUE
* statement : $S(n)$ holds for all natural number n

* more example :
slide 04 21-24/27
* exercise :
slide 04 25/27

induction basis : $\neg P_1 = \neg P_1$
inductive hypothesis : $\neg(P_1 \vee \dots \vee P_k) = \neg P_1 \wedge \dots \wedge \neg P_k$
inductive step : $\neg(P_1 \vee \dots \vee P_{k+1}) = \neg((P_1 \vee \dots \vee P_k) \vee P_{k+1})$ grouping
$= \neg(P_1 \vee \dots \vee P_k) \wedge \neg P_{k+1}$ de morgan's law
$= \neg P_1 \wedge \dots \wedge \neg P_k \wedge \neg P_{k+1}$ inductive hypothesis

conclusion : by the principle of mathematical induction, statement is held TRUE ✱

Lecture 5

① GOOD proofing eg. Euclid, Pythagoras, Cantor
② BAD proofing
③ UGLY proofing

Euclid

↳ "There are infinitely many prime number"

↳ contradiction : There are only finitely many prime number ⇒ fixed no. of prime all declared in list

n = number of prime $(p_1, p_2, \dots, p_n)$

q = product of all prime number + 1

↳ q bigger than all prime so q != any prime in list

must q be composite when it != any prime within list ?

q ÷ any prime = remainder 1 cus q = all prime multiplied + 1

- contradiction :

1. q is divisible by any prime but leave remainder 1

↳ if it's like that, q is not a multiple of the prime

HENCE, the assumption is FALSE so there can't be only finitely many prime

THEREFORE, there are infinitely many prime numbers

LECTURE 6

awk : text manipulation

alternative

↳ indicated by $\cup$ (union)

↳ eg. $1 \cup 2 \cup 3 \cup 4 \cup 5$

regular expression for $\{1,2,3,4,5\}$

grouping

↳ indicated by ( ) (bracket)

↳ eg. $(ab \cup ba)(e \cup g)$

regular expression for $\{abe, abg, bae, bag\}$

finite languages

↳ finite number of words

↳ eg. $\{abaaba, abbba, abbaba\}$

regular expression : $abaaba \cup abbba \cup abbaba$

$ab(aa \cup bb \cup ba)ba$

kleene star

↳ indicated by * (astenick)

↳ match 0 or more expressions

↳ eg. a^* represents $\{\varepsilon, a, aa, aaa \dots\}$

$(ab)^*$ represents $\{\varepsilon, ab, abab, ababab \dots\}$

ab^* represents $\{\varepsilon, ab, abb, abbb \dots\}$

$(aa \cup bb)^*$ represents $(aa \cup bb)^0 \cup (aa \cup bb)^1 \cup (aa \cup bb)^2 \dots$

$\downarrow$ ε | $\downarrow$ $(aa \cup bb)$ | $\downarrow$ $(aa \cup bb)(aa \cup bb)$

regular language

↳ language describe by regular expressions

↳ word belong to language described by regular expressions
= word match the regular expression

↳ regular expression :

① $\emptyset$ ε ✓ regular expression

② all letters in alphabet ✓ regular expression

③ if R & S is regular expression then...

i. (R) ✓ regular expression

ii. RS ✓ regular expression

iii. R ∪ S ✓ regular expression

iv. R^* ✓ regular expression

↳ eg. EVEN-EVEN = $\{\varepsilon, aa, bb, aaaa, aabb, abab, abba \ldots\}$

regular expression : $(aa \cup bb \cup (ab \cup ba)(aa \cup bb)^* (ab \cup ba))^*$

sequence of digits

one digit : $0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9$ } = D

two digit : $(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9) \times (0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)$ } DD OR D^2

three digit : $(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9) \times (0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9) \times (0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)$ } DDD OR D^3

one or more digit : $(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9) \times (0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^*$ } DD^*

integers : $Z = N \cup (-N)$

nonnegative integers : $N = DD^*$

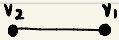
floating point number : $F = Z \cup (Z.) \cup (.N) \cup (-.N) \cup (Z.N)$

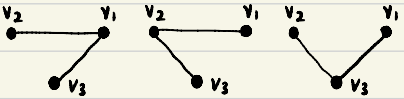
Assessed Preparation : Question 3

For all n, the number of trees on n vertices is at least $(n-1)!$

t_n = number of trees on n vertices

(a) one vertices : 

two vertices : 

three vertices : 

(b) When $n=1$

The number of trees on 1 vertex is 1 as the vertex itself can be considered a tree.

Following the statement where For all n, the number of trees on n vertices is at least $(n-1)!$

It is true as $n=1$, $(n-1)! = (1-1)! = 0! = 1$

Hence the statement is true when $n=1$

(c) $n \geq 1$

assuming $n=1$ is true then $n+1$ is true for the statement too.

if $n=1$, $0!=1$

when $n+1 = 2$,

$(n-1)! = (2-1)! = (1)! = 1$ tree

from the pre-proof exploration we can see that there is indeed only 1 tree when there's 2 vertices so $((n+1)-1)!$ is indeed true.

(d) not possible, all trees obtained from each different number of vertices is distinct

(e) $t_{n+1} \geq n(t_n)$

(f) $t_n \geq (n-1)!$ - ① $t_{n+1} \geq n(t_n)$ - ② ①-②

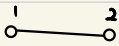
$t_{n+1} \geq n(n-1)! \longrightarrow t_{n+1} \geq n!$

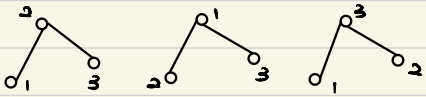
no. of trees on $(n+1)$ vertices at least $n! = (n+1)-1)!$

(g) By mathematic induction, the statement stated above is proven to be TRUE for $n \geq 1$ hence for all n, the number of trees on n vertices is at least $(n-1)!$

Assessed preparation (teacher's solution)

(a) One vertex : 

Two vertices : 

Three vertices : 

(b) P(n) = the number of trees on n vertices is at least $(n-1)!$

Base case = P(1) is true because the only tree with one vertex is •,
so the number of trees with 1 vertex is $1=(1-1)!$

Inductive step = let $n \geq 1$

Assume P(n) is true

need to show that P(n+1) is true

$t_{n+1} \geq n(n)$

$\geq n(n-1)!$

$\geq n!$

$\geq ((n+1)-1)!$

HENCE, P(n+1) is true

By the principle of mathematical induction,
P(n) is true for all $n \geq 1$.

4) P(n) : every tree on n vertices has n-1 edges

Prove P(n) for all $n \geq 1$

Base case : the only tree with 1 vertices has 0 edges so P(1) is true

Inductive step :

let $n \geq 1$, assume P(n) is true (need to show P(n+1) is true

let T be tree with n+1 vertices (by assumption T^* have n-1 edges)

remove a leaf to make T^* (all trees with ≥ 1 vertex has 1 leaf)

T has one more vertex & edge than T^*

so P(n+1) is true ↳ have n+1 vertices, n edges

Therefore, by the mathematical induction, P(n) is true for all $n \geq 1$

2) all dogs are cute

↳ there exist a dog that is not cute

every non-empty hereditary language contain the empty string

↳ there exist a non-empty hereditary language that don't contain the empty string

L = non-empty hereditary language

there is a shortest string in L (string x)

x^{-1} is x but with a character remove

↳ possible, because it's hereditary

↳ x^{-1} is in L

BUT this means that x^{-1} exist in L as the shortest string WHEN you have already found the supposed shortest string in L which is x.

↳ eg. "contains" is the supposed shortest BUT remove a character "s" it became "contain" which is now the newest shortest string in L as "contains" > "contain"

↳ another example:

"a" is the supposed shortest string in L BUT remove a character "a" it became "" which is an empty string which is now the newest shortest string in L as "a" > ""

conclusion: there exist a non-empty hereditary language that contain the empty string Hence the statement that there exist a non-empty hereditary language that does not contain the empty string is indeed FALSE.

1)(e) $Crosses\,To\,move(p) \wedge$

$\exists x_1 \exists x_2 \exists x_3$ "there exist move 1, there exist move 2, there exist move 3"

$Crosses\,To\,win(p(\exists x_1 \exists x_2 \exists x_3))$

WRONG:

$\exists x_1 \forall x_2 \exists x_3$

"there exist move 1, for all move 2, there exist move 3"

5) (a) $n! \leq (n-1)^n$, $n \geq 3$

Base case: $n = 3$

Induction hypothesis: $n+1 \geq 3$

Inductive step:

Lecture 7

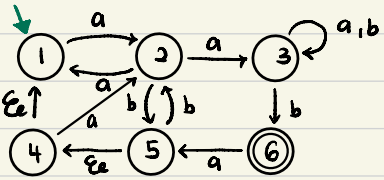
Finite Automaton (FA)

↳ aka Deterministic Finite Automaton (DFA)

↳ determine if word belong to Regular Language or not

↳ define Regular Language

↳ eg.



** empty string / Lemda
↳ accepted from DFA
↳ initial = final state
for specifically a DFA

} unique start point

} accepting final state

transitioning
↳ directed edges
↳ labelled by letters

** finite set of ...
↳ states
↳ transitions

↳ String is only accepted by FA when path ends on Final State
↳ else, rejected
↳ accepted = FA recognise the language

Special cases

① all words accepted
② all words rejected
③ only empty words accepted
④ only non-empty words accepted
⑤ single word accepted

Complement

- If L = language over alphabet
- then L complement ($\bar{L}$, L', L^c) = set of words over alphabet not in L
- $\bar{L} = \Sigma^* \setminus L$

** ε = empty words
↳ included in label for transitions in NFA

Lecture 8

revision (properties)

- accept a string if there is at least one path from start → final state
- reject a string if there is no path from start → final state

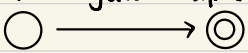
Kleene's Theorum

- any language defined by...
  - regular expressions
  - finite automata (FA)
  - nondeterministic finite automata (NFA)
  - generalized nondeterministic finite automata (GNFA)
- can be defined by any other method
- regular expressions → NFA
  - NFA → FA
  - GNFA ← FA

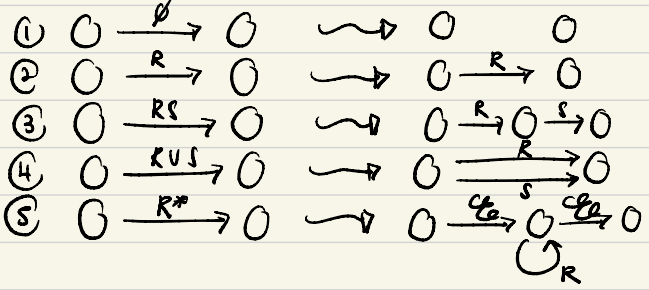
convert regular expressions to NFA

6/28 slide

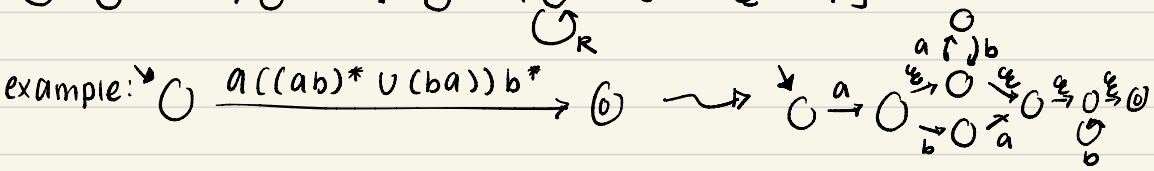
- regular expression



main rule:



example: $a((ab)^* \cup (ba))b^*$



Convert NFA to FA (Finite Automaton) 16/28 slide

Summary:

↳ FA: only follow clear path from start to end

↳ NFA: follow many path at the same time to find possible end states and find which end state is accepted

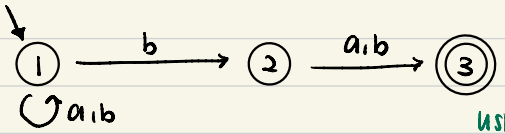
Finite Automaton (FA):

- Imagine you have a machine that reads strings (sequences of symbols).
- Every string you give this machine follows a clear path, starting from a specific starting point called the Start State.
- The path the machine follows ends at a particular state, which we'll call endState(w), depending on the string it reads.
- If the end state of a string is a Final State (a designated accepting state), the machine accepts the string; otherwise, it rejects it.
- When you give the machine an empty string (ε), it simply stays at the Start State.
- In this setup, there's only one way to go from one state to another for each input symbol.

Non-Deterministic Finite Automaton (NFA):

- Picture a machine that's a bit more flexible. When it reads a string, it can take multiple paths, not just one.
- For any given string, this machine can end up at a set of states, which we'll call endStates(w).
- This set of states might contain no states, a single state, or multiple states.
- To decide whether it should accept a string, the NFA checks if any of the states in endStates(w) are Final States.
- Just like in the FA case, when given the empty string (ε), if there are no ε transitions (transitions without reading any symbol), the machine's set of end states is just the Start State.
- The NFA can "jump" or transition from one state to multiple possible states based on the same input symbol.

Convert NFA to FA (Finite Automaton)



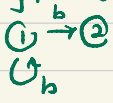
using path a

state	a	b
start {1}		

→

state	a	b
start {1}	{1}	{1,2}
{1,2}		

→ using path b



↓

state	a	b
start {1}	{1}	{1,2}
{1,2}	{1,3}	{1,2,3}
{1,3}		
{1,2,3}		

another example!

slide 25-26/28

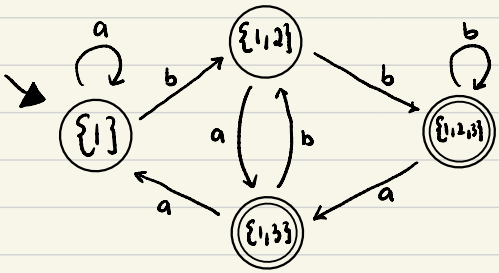
↓

state	a	b
start {1}	{1}	{1,2}
{1,2}	{1,3}	{1,2,3}
{1,3}	{1}	{1,2}
{1,2,3}	{1,3}	{1,2,3}

↓

	state	a	b
	start {1}	{1}	{1,2}
	{1,2}	{1,3}	{1,2,3}
Final	{1,3}	{1}	{1,2}
Final	{1,2,3}	{1,3}	{1,2,3}

NFA have empty transitions (ε)
- ↳ take account of empty transitions
- ↳ include empty transitions in states we can reach



Lecture 9

FA (Finite Automaton) → GNFA (Generalised Nondeterministic FA)

GNFA
- ↳ NFA with transitions labelled by regular expressions
- ↳ string accepted if can divide into substring
  - ↳ some sequence of transitions
  - ↳ start: start state ; finish: final state
  - ↳ labelled by regular expression
  - ↳ not accepted by GNFA = rejected

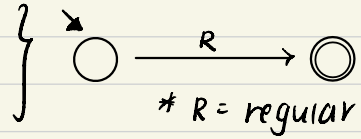
Standard GNFA
- ↳ GNFA with 1 final state
  - ↳ Final state ≠ start state
  - ↳ start state have no incoming transitions
  - ↳ Final state have no outgoing transitions

FA → Standard GNFA Slide 6/23

Standard GNFA → Regular Expression Slide 8/23

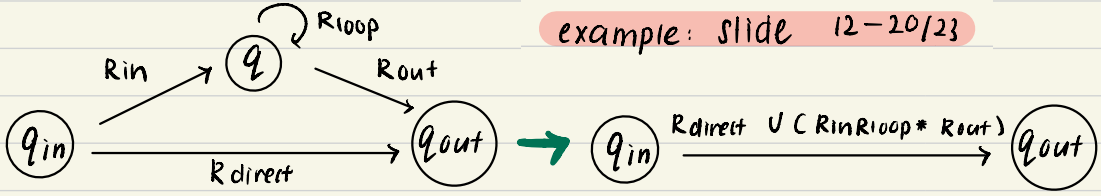
① convert to GNFA with one fewer state

② continue until left 2 state 1 transition:
- ↳ start state x1
- ↳ Final state x1
- ↳ transition x1



* R = regular expression

visualisation:

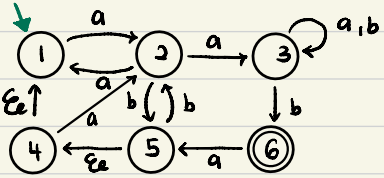


example: slide 12–20/23

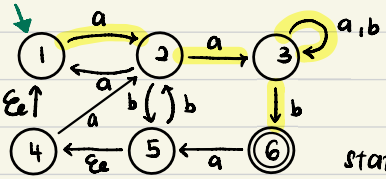
Assessed Preparation : Question 5

5) Nondeterministic Finite Automaton

attendance name row : 279

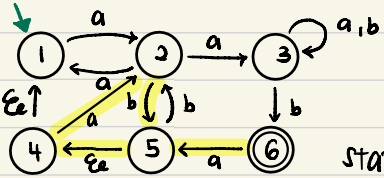


(a)



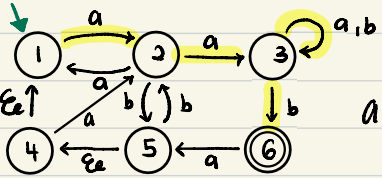
state 6

(b)



state 5

(c)



accepted, start ① ends ⑥ is one of the accepting states of aaab

(d) if (aaab) is accepted then it will end on an accepting state like ⑥ hence if (aaab) is added to the end of the original aaab string then it will go through the same transitions as before as $(aaab)^2$ hence if $(aaab)^2$ is allowed and accepted then $(aaab)^n$ is also accepted and so $(aaab)^{n+1}$ will also be accepted and lead to the accepting state again by the NFA.

(e) base case ⇒ n = 1 (part a)

inductive hypothesis ⇒ assume $(aaaab)^n$ is accepted by NFA

inductive step ⇒ part d reasoning to prove assumption is accepted by NFA and show $(aaab)^{n+1}$ is also accepted.

conclusion ⇒ by the principle of mathematical induction, $(aaab)^n$ is true for all n

1) $(ab \cup ba)(aa \cup bb)^* ab$ ∴ when there's '*', can be kosong / empty

= abaaaaaab = baaaaaaab = abab = abaaab

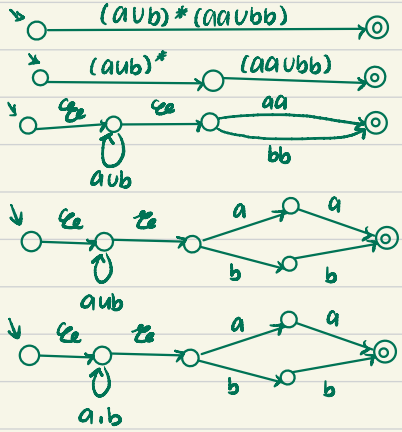
= abaabbbab = baaabbbab = baab = abbbab

= abbbaaaab = babbaaaab = baaaab

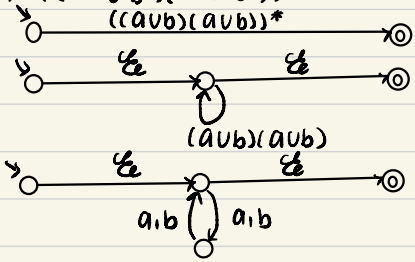
= abbbbbbab = babbbbbab = babbab

2)(i) $aa \cup bb \cup ab \cup ba$

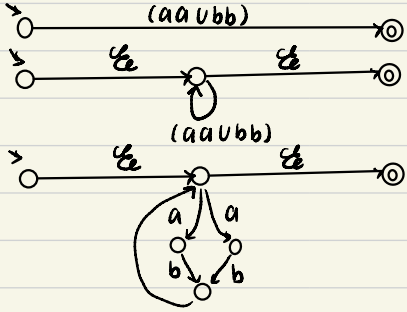
6) (i) $(a \cup b)^*(aa \cup bb)$ "concatanation"



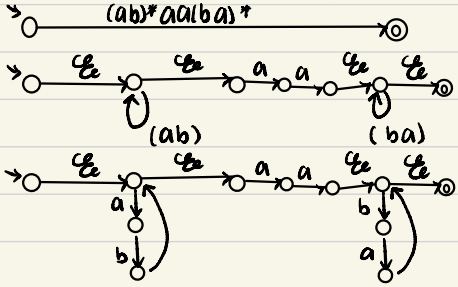
(ii) $((a \cup b)(a \cup b))^*$



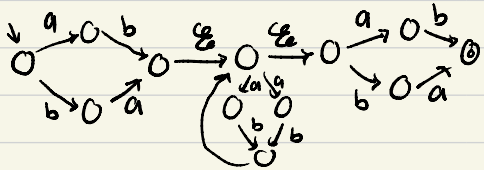
(iii) $(aa \cup bb)^*$



(iv) $(ab)^* aa(ba)^*$

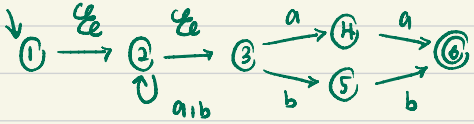


(v) $(ab \cup ba)(aa \cup bb)^*(ab \cup ba)$



start state

"where can I get to with empty string"

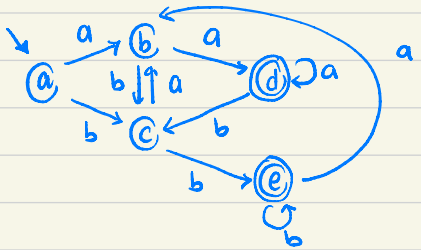


	state	a	b	
start	{1,2,3}	{2,3,4}	{2,3,5}	a
	{2,3,4}	{2,3,4,6}	{2,3,5}	b
	{2,3,5}	{2,3,4}	{2,3,5,6}	c
final	{2,3,4,6}	{2,3,4,6}	{2,3,5}	d
final	{2,3,5,6}	{2,3,4}	{2,3,5,6}	e

what if read a or b cannot go anywhere?

- put in empty state '$\emptyset$'
- eg.

state	a	b
⋮	⋮	⋮
$\emptyset$	$\emptyset$	$\emptyset$



Lecture 10

matching regular expression

"identify whether string w matches regular expression"

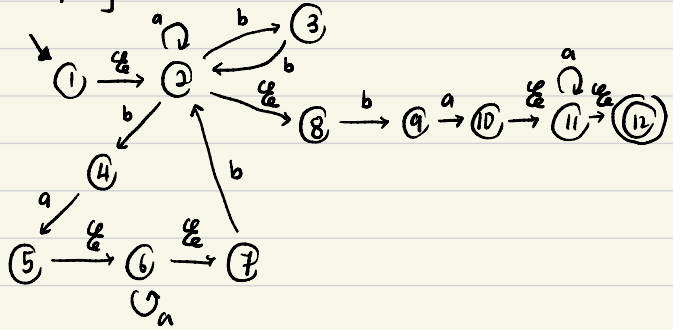
regular expression : $(a \cup bb \cup baa^*b)^*baa^*$

① convert regular expression to NFA

② convert NFA to DFA

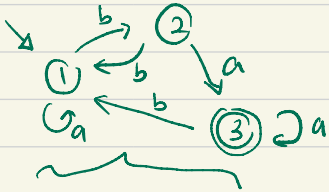
③ simplify DFA

NFA:



DFA:

		a	b
start	{1,2,8}	{2,8}	{3,4,9}
	{2,8}	{2,8}	{3,4,9}
	{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final	{5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final	{6,7,11,12}	{6,7,11,12}	{2,8}



	a	b
start 1	1	2
2	3	1
Final 3	3	1

Final states x Non-Final states CANNOT BE COMBINED.

	a	b
start {1,2,8}	{2,8}	{3,4,9}
{2,8}	{2,8}	{3,4,9}
{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final {5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final {6,7,11,12}	{6,7,11,12}	{2,8}

} same : combine

} same : combine

pattern ⇒ specified by regular expression

token ⇒ name of pattern
↳ attribute association (maybe)

lexeme ⇒ sequence of character that matches pattern to token
↳ implemented using Finite Automaton (FA)

conventions — slide 21/22

↳ match largest possible lexeme at each stage

↳ same length lexeme? choose first listed token

Lecture 11

regular language closure properties

operation done on some regular language produces another regular language
- ↳ class of regular language is closed under that operation
  - ↳ closed under: ① complement
    ② union
    ③ intersection
    ④ concatenation

** complement of regular language = regular
- ↳ prove: Kleene's Theorum

closure properties = rules showing operation performed will always result in another set of words following same rule (pattern)
- ↳ if output is always a set that follow same pattern = regular language "closed" under the action

complement section:
1. start regular language (fit pattern)
2. find its complement (don't fit pattern)
   - ↳ still a regular language

* meaning: "rule / pattern" that define original regular language also works to define new complement set

regular language closure properties theorem proving

Theorem: The complement of regular language is regular

regular language example: word starts with "A"
↳ eg. Apple, Ass, Aiyo ...

complement: words that DON'T start with "A"
↳ eg. Bass, Car, Duck...

Prove complement is also a regular language:

- Normally... in the original set of regular expression...
  it accepts the words that start with "A" (fits pattern A-...)
- Now...
  it accepts the words that DON'T start with "A" (fits pattern !A-...)

Kleene's Theorum

"If you can write down a set of instruction (regular expression) to make special words that follow specific pattern then you can create a little machine to follow same instructions"
↳ Finite Automaton (FA)

If there's a regular expression, there will be an equivalent FA

proving examples

slide 5 - 7/22

Circuits in FA

circuit
- ↳ directed path
- ↳ start, end at same state

length of circuit
- ↳ number of edges in path

Finite Automaton (FA) states :
1. before circuit
2. goes around circuit
3. after circuit

∴ visualisation of circuit and FA states — slide 9-11/22

pumping lemma

slide 12/22 start

- ↳ theorem that show language L is non-regular
  - ↳ example question where L = regular expression assumption
- ↳ if a language is regular, it always satisfy pumping lemma
  - ↳ pumping lemma cannot show / prove its regular but only if its irregular
    - ↳ form : "Every language has the following property..."
    - ↳ pumping lemma has the form
      - ↳ show language has no form and/or necessary parts
        - ↳ prove irregularity
- ↳ pumping = generate n number of substrings
  - ↳ check if have form and/or necessary parts
- ↳ slide 12-14/22

non-regular languages

slide 15-22/22

Assessed Preparation : Question 4

	state	a	b
Start	1	2	4
	2	2	6
	3	3	4
Final	4	5	3
Final	5	5	5
Final	6	5	1

- Start states : 1
- Final states : 4,5,6
- Non-Final states : 1,2,3

String finishes in Final state = Accepted
String finishes in Non-Final state = Rejected

	state	a	b
Start	1	2	4
	2	2	6
	3	3	4
Final	4	5	3
Final	5	5	5
Final	6	5	1

same pattern

same pattern

different pattern

	state	a	b
Start	1	2	4
	2	2	6
	3	3	4
Final	4	5	3
Final	5	5	5
Final	6	5	1

① ② ③

Final answer for question 4 :

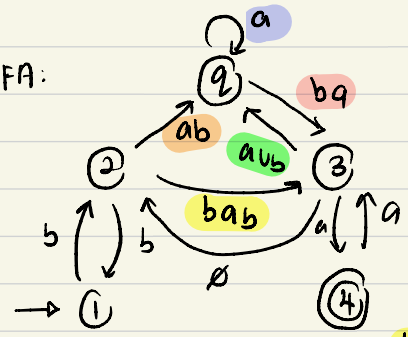
state	a	b
Start	1	2
	2	3
Final	3	3

	state	a	b
Start	1	1	4
	4	5	1
Final	5	5	5

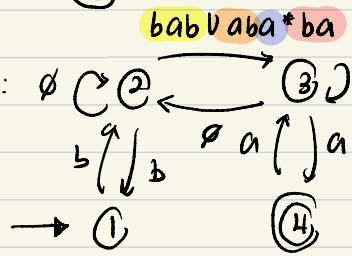
	state	a	b
S F	0	6	2
	1	4	4
	2	2	1
	3	6	0
	4	1	5
F	5	5	1
	6	1	4

minimising FA
All is different colour
∴ this is indeed the minimised FA

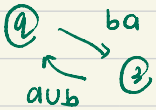
1) GNFA:



removing q : babUaba*ba

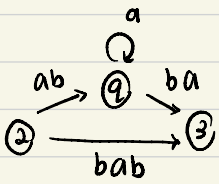


↳ looping itself
↳ a



" go & come back "
(aUb)a*ba

* cus reach mid-point and going to another point

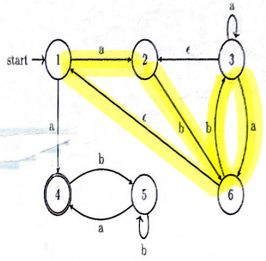


2 → q → 3
↳ from one node to go another
↳ kleenix star (*)
↳ aba* ba
↳ adding with edge 2 : bab
↳ bab U aba*ba

1 Questions

Induction with NFAs

Prove, by induction on n, that for all positive integers n, the string $(abba)^n$ is accepted by this NFA.



Contradiction

Let a language be called *huggy* if and only if for every pair of distinct words, x and y, in the language, the words xyx and yxy are also in the language.

Prove by contradiction that every *huggy* language that contains at least two words is infinite.

Base case: $n=1$

String $(abba)^n$ is accepted by NFA when $n=1$ as $(abba)^1 = abba$ and the possible states are 3, 6, 4 and state 4 is the final state so it will be accepted by NFA after transition.

Inductive hypothesis:

Assume that for all positive integer n, the string $(abba)^n$ is accepted by the NFA

Inductive step:

Assume $(abba)^{k+1}$ is accepted by NFA.

There exist path from start state to start state:

$1 \to 2 \to 6 \to 3 \to 6 \to 1 \to 2 \to 6 \to 3 \to 6 \to 1$

There exist path from start state to end state from base case:

$1 \to 4 \to 5 \to 5 \to 4$

Concatinating both path, it will be a path existing from start to end that is $(abba)^{k+1}$ so the string $(abba)^{k+1}$ is accepted by the NFA and hence $(abba)^n$ is accepted by the NFA

Conclusion:

By the principle of mathematic induction, the string $(abba)^n$ is accepted by the NFA for all positive integer n.

		a	b
S	1	2	3
	2	1	5
	3	1	4
F	4	2	5
F	5	3	5

	a	b
1	1	1
1	1	2
2	1	2

		a	b
S	1	2	3
	2	1	5
	3	1	4
F	4	2	5
F	5	3	5

	a	b
1	2	2
2	1	3
3	2	3

①

	a	b
1	2	2
2	1	4
4	2	4

②

* if it's element is not numbers (like above) but sets (like {1,2,3,4}) then use colours to notate and minimise
  ↳ label colours with numbers like ①

* if its element is numbers (like above) use the numbers given with colour coded to label like ②

Started on	Saturday, 2 September 2023, 9:44 AM
State	Finished
Completed on	Saturday, 2 September 2023, 9:47 AM
Time taken	3 mins 40 secs

Print friendly format

Question **1**

Not answered

Marked out of 2.00

Let P, Q, R be the following propositions:

P: My family name comes before my given name.

Q: I am Chinese.

R: I am Thai.

if got "if" then use ⇒

Using P, Q, R and appropriate connectives, construct a proposition in Conjunctive Normal Form to express the fact that my family name comes before my given name if I'm Chinese, while it comes after my given name if I'm Thai.

The following symbols are provided for you to copy if you wish (though not many of them are needed in this question, and you are not limited to using the symbols that are listed here): ∃ ∀ ∧ ∨ ¬ ⇔ ⇒ ⇐ ∈ ≤ ≥ ≠

Answer: $(P \wedge Q \wedge \neg R) \vee (R \wedge \neg Q \wedge \neg P)$

Question **2**

Not answered

Marked out of 2.00

Suppose that:

remove bracket

- variables X and Y can each represent any Nondeterministic Finite Automaton (NFA);
- the function L(...) returns the language recognised by the NFA given to it;
- the predicate **`Deterministic`**(X) is True if and only if X is actually also a Deterministic Finite Automaton (FA).

Using these, write a logical statement using quantifiers to express the fact that every NFA has an equivalent FA.
The following symbols are provided for you to copy if you wish (though not many of them are needed in this question, and you are not limited to using the symbols that are listed here): ∃ ∀ ∧ ∨ ¬ ⇔ ⇒ ⇐ ∈ ≤ ≥ ≠

For all X, there exist Y that recognise same language as X

Answer: $\forall X(\exists Y\ (\text{Deterministic}(Y) \wedge L(X) = L(Y)))$

Question **3**

Not answered

Marked out of 3.00

Write a regular expression for the language of all strings over the alphabet {**a**,**b**} in which every occurrence of the letter **`a`** has the letter **`b`** next to it on both left and right sides.

Answer: $(bb^*abb^*a^*)^*$

Question **4**

Not answered

Marked out of 2.00

Write all the words of length ≤ 5 that belong to the language represented by the regular expression ((**aaa***)∪(**bbb**))*

Answer: $\varepsilon, aa, aaa, bbb, aaaa, aaaaa, bbbaa, aabbb$

Question **5**

Not answered

Marked out of 2.00

Which one or more of the following types of automata can recognise all regular languages over the alphabet {**a**,**b**}?

Select one or more:

- ✓ a. FA with an input alphabet of at most two symbols
- ☐ b. NFA with no directed circuits
- ☐ c. FA with no directed circuits
- ✓ d. NFA with no empty-string transitions
- ☐ e. NFA with only one Final State
- ✓ f. FA with at most two states

Question **6**

Not answered

Marked out of 6.00

Let x be a string, and let M be a Finite Automaton with just one Final State that accepts the strings x and xx.

(a) Prove, by induction on n, that M accepts the string x^n for every $n \geq 1$.

(b) Would the same statement hold if M is a *Nondeterministic* Finite Automaton, also with just one Final State, instead? Why or why not?

(a) Base case: $n = 1$

when $n = 1$, $x^1 = x$ and x is accepted by m so m accepts x when $n = 1$

inductive hypothesis: Assume that m accepts x^n when $n \geq 1$

inductive step: WTTP m accepts x^{k+1} where $k \geq 1$ since m accepts xx which is x^n when $n = 2$

conclusion:

Therefore, by the principle of mathematical induction, m accepts the string x^n where $n \geq 1$

(b) is m is a NFA with just one final state, the same statement may not hold.

$x^{k+1} = x^k \cdot x$

$=$ when $n = k$. From BC, since is x, then m also accepts

$\therefore$ x^{k+1} is accepted by m

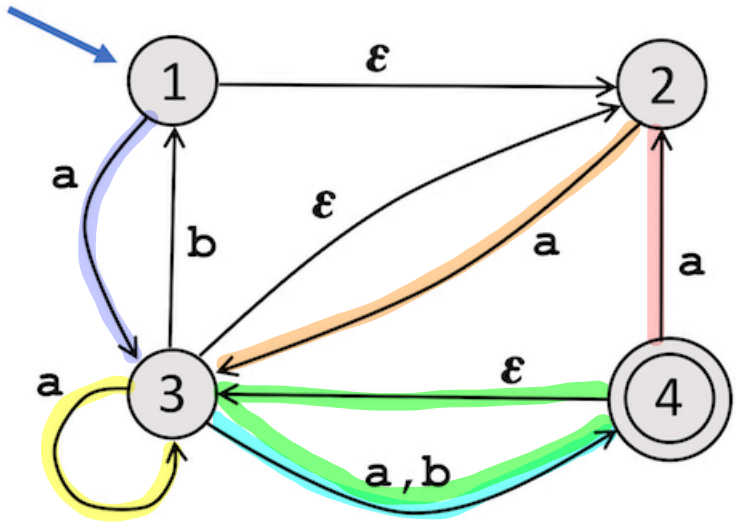
Question 7

Complete

Marked out of 7.00

Convert the following Nondeterministic Finite Automaton (NFA) into an equivalent Deterministic Finite Automaton (FA).

Your FA must be presented by filling in some rows in the table further down. You may not need all the rows available.



state	a	b
Start {1}	{3}	-
{3}	{3, 4}	{4}
{3,4}	{2}	{4}
{4}	{2}	{4}
{2}	{3}	-

Question **8**

Complete

Marked out of 5.00

Consider the five-state Finite Automaton represented by the following table.

state	a	b
Start 1	3	1
2	4	2
Final 3	1	3
Final 4	2	4
Final 5	3	5

Find an equivalent FA with the minimum number of states.

Enter your simplified FA in the table below. You may not need all rows of the table.

state	a	b
Start {1,2}	{3,4}	{1,2}
Final {3,4}	{1,2}	{3,4}
Final {5}	{3,4}	{5}

Prove using pumping lemme :

① assume x is regular
② $w = xyz$
③ $y \neq \varepsilon$
④ $xy \leq n$
⑤ $xy^i z$ = regular language

Question **9**

Not answered

Marked out of 7.00

Let ANTIPALINDROME be the language of all even-length strings over the alphabet {a,b} such that, for all i, the i-th letter from the start is *different* to the i-th letter from the end. Examples of strings in this language include:

ε, **ab, ba, aabb, abab, baba, bbaa, aaabbb,** ...

Using the Pumping Lemma for Regular Languages, prove that ANTIPALINDROME is not regular.

Assume L = ANTIPALINDROME is regular
N = number of states of FA that recognises L
W = member of L = $a^n b^n$
$w = xyz$ where xyz is the substrings of w
y is not empty
$|xy| \leq n$
by pumping lemma theorum, $xy^i z$ is a member of L when $i \geq 0$
x and y consists of as but z can be consists of b's only or a and b
assuming $x = a^g$, $y = a^c$, $z = a^{(n-g-c)} b^n$ where $c, n > 0$ and $g \geq 0$
when $i = 0$ such that $xy^0 z = xz$ then $a^g (a^{(n-g-c)} b^n = a^{(n-c)} b^n$
since $n - c < n$ there is a contradiction, Hence its not part of ANTIPALINDROME
as there's less a than b

Hence by contradiction and pumping lemma, ANTIPALINDROME is not regular

Question **10**

Faculty of Information Technology
Monash University

FIT2014 Theory of Computation
SAMPLE MID-SEMESTER TEST

2nd Semester 2016

Question 1 **(5 marks)**

A language L is called **hereditary** if it has the following property:

> For every nonempty string x in L, there is a character in x which can be deleted from x to give another string in L.

Prove by contradiction that every nonempty hereditary language contains the empty string.

Base case:
For every nonempty string x in L, there is a character in x which can be deleted from x to give another string in L

Inductive hypothesis:
Assume that every nonempty hereditary language does not contain the empty string.

Inductive step:
since L is nonempty, it contains one string which can be the shortest. x = shortest string in L and x is not empty as assumed so length of $x \geq 1$ and if x is the shortest it's length $= 1$
If L is hereditary, x from L can have a letter deleted from it to be given to another string found in L and if x length $= 1$ and it have a letter removed, it would have length $= 0$ and that is an empty string. It is assumed that L is nonempty but this example of length of $x = 1$ prove that there could not exist a nonempty hereditary language that does not contain the empty string.

Conclusion:
From the principle of contradiction, every nonempty hereditary language does indeed contain the empty string.

Tutorial / Exercise 2 Question 2

ChatGPT

To prove by contradiction that every nonempty hereditary language contains the empty string, we'll assume the opposite, i.e., there exists a nonempty hereditary language that does not contain the empty string, and then derive a contradiction.

Let's assume there exists a nonempty hereditary language L that does not contain the empty string ε (empty string).

Now, we know that L is hereditary, which means that for every nonempty string x in L, there is a character in x that can be deleted from x to give another string in L.

Let's consider a string y in L such that y is the shortest string in L. Since L is nonempty, such a string y must exist. Now, because y is in L, it follows that there must be a character in y that can be deleted to obtain another string in L.

However, there's a problem here. If we delete any character from the shortest string y, we would necessarily obtain a shorter string than y, and by the definition of the shortest string in L, this new string cannot be in L. This contradicts the assumption that every character-deletion from a string in L results in another string in L.

Therefore, our initial assumption that there exists a nonempty hereditary language L that does not contain the empty string ε is false. We have shown that such a language cannot exist. Thus, every nonempty hereditary language must contain the empty string ε.

Question 2 **(8 marks)**

In this question, we write T for True and F for False.

Let P_n be the proposition $x_1 \wedge x_2 \wedge \cdots \wedge x_n$. Note that P_1 consists just of x_1, and P_n is equivalent to $P_{n-1} \wedge x_n$.

Prove by induction on n that, if $x_1 = \mathrm{F}$, then P_n is False.

Base case : when $n=1$

$P_n = P_{n-1} \wedge x_n$

$P_1 = P_{1-1} \wedge x_1$

$= P_0 \wedge x_1$

$P_1 = x_1$ which is true to the statement P_1 consists of x_1

Inductive hypothesis : Assume if $x_1 = F$ then P_n is F

when $n=1$, $P_1 = x_1$ so if $x_1 = F$ then P_1 is F

let $k \geq 1$ if $x_1 = F$ then $P_k = F$

Inductive step : if $x_1 = F$

$P_{k+1} = P_k \wedge x_{k+1}$ if $P_k = F \ldots$

$= F \wedge x_{k+1}$

$P_{k+1} = F$

Conclusion :

when P_{k+1} = False which shows that by the principle of mathematical induction, the statement holds for all n.

Tutorial / Exercise 2 Question 6

Question 3 **(3 marks)**

Write down a regular expression for the language of all binary representations of positive integers.

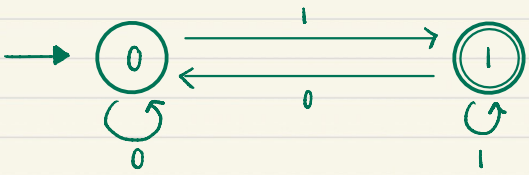
Your alphabet for this question is {0,1}. Leading zeros are not allowed. Do not use any signs or decimal points.

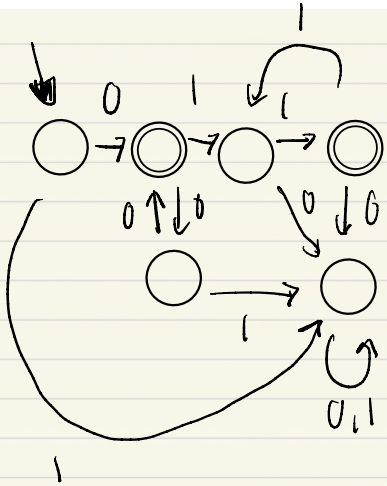
ANSWER: $1(0 \vee 1)^*$

Question 4 **(3 marks)**

Construct a Finite Automaton (FA) to recognise the language of strings consisting of an odd number of 0s followed by an even number of 1s.

(Note: zero is considered to be an even number.)

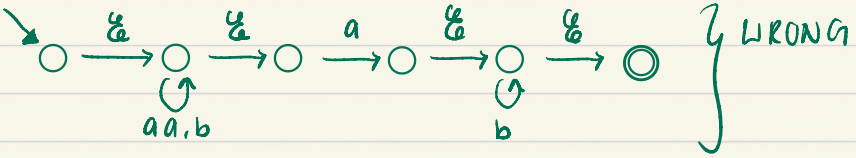


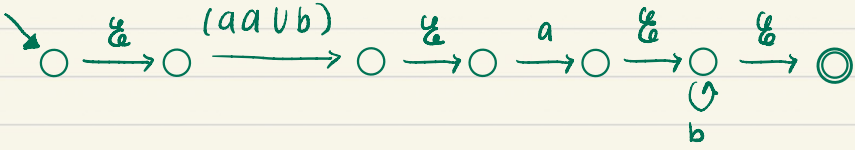


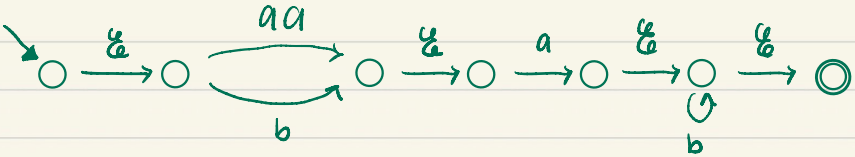
Question 5 **(7 marks)**

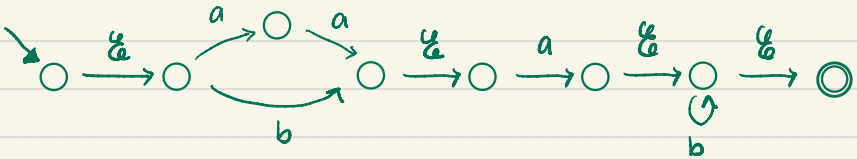
Construct a Nondeterministic Finite Automaton (NFA) to recognise the language represented by the regular expression

$$(\mathtt{aa} \cup \mathtt{b})^*\mathtt{ab}^*.$$









Question 6 **(4 marks)**

Consider the five-state Finite Automaton represented by the following table.

state		a	b
Start	1	2	3
	2	1	5
	3	1	4
Final	4	2	5
Final	5	3	5

Convert this into an equivalent FA with the minimum possible number of states.

Write your answer in the following table. You may not need all the rows available.

state		a	b
Start	1	2	2
	2	1	3
Final	3	2	3

Exercise 1 (Tutorial sheet 1) Languages and Logic

1) $V_{t,s}$ where t: time, s: square

(a) $V_{0,1}$

(b) $\neg V_{0,2}$

(c) $V_{0,1} \wedge \neg V_{0,2} \wedge \neg V_{0,3} \wedge \neg V_{0,4}$

(d) $V_{5,1} \vee V_{5,2} \vee V_{5,3} \vee V_{5,4}$

(e) $\neg V_{5,1} \wedge \neg V_{5,3}$

(f) $(\neg V_{5,1} \vee \neg V_{5,2}) \wedge (\neg V_{5,1} \vee \neg V_{5,3}) \wedge (\neg V_{5,1} \vee \neg V_{5,4}) \wedge (\neg V_{5,2} \vee \neg V_{5,3}) \wedge (\neg V_{5,2} \vee \neg V_{5,4}) \wedge (\neg V_{5,3} \vee \neg V_{5,4})$

(g) $(V_{5,1} \vee V_{5,2} \vee 5,3 \vee 5,4) \wedge (\neg V_{5,1} \vee \neg V_{5,2}) \wedge (\neg V_{5,1} \vee \neg V_{5,3}) \wedge (\neg V_{5,1} \vee \neg V_{5,4}) \wedge (\neg V_{5,2} \vee \neg V_{5,3}) \wedge (\neg V_{5,2} \vee \neg V_{5,4}) \wedge (\neg V_{5,3} \vee \neg V_{5,4})$

(h) $V_{3,2} \Rightarrow (V_{4,3} \vee V_{4,1})$

Forward: square 3 at time 4

Backward: square 1 at time 4

(i) time 0 clauses : 4

time 1,2,3 in exactly 1 square : 3(g) $\Rightarrow 3 \times 7 = 21$

forward / backward : 4×3(h) $\Rightarrow 12$

$4 + 21 + 12 = 37$ ✱

2) PALINDROMES is a subset of $\overline{\text{ODD-ODD}}$

- even length palindrome (eg. abba) $\in \overline{\text{ODD-ODD}}$
  - ↳ $\overline{\text{ODD-ODD}}$ = not odd no. of a and b = even no. of a & b
- odd length palindrome (eg. abcba, abbba) $\in \overline{\text{ODD-ODD}}$
  - ↳ $\overline{\text{ODD-ODD}}$ = (abcba) not odd no. of a and b
  - ↳ $\overline{\text{ODD-ODD}}$ = (abbba) not odd no. of a but odd b no.
    - ↳ since odd, no. of a and b cannot be odd or total is even
  - ↳ both cases $\in \overline{\text{ODD-ODD}}$

* AND $\wedge$
* OR $\vee$

3) (a) $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

P	Q	R	$Q \wedge R$	$P \vee (Q \wedge R)$
T	T	T	T	T
T	T	F	F	T
T	F	T	F	T
F	T	T	T	T
F	F	T	F	F
F	T	F	F	F
T	F	F	F	T
F	F	F	F	F

P	Q	R	$P \vee Q$	$P \vee R$	$(P \vee Q) \wedge (P \vee Q)$
T	T	T	T	T	T
T	T	F	T	T	T
T	F	T	T	T	T
F	T	T	T	T	T
F	F	T	F	T	F
F	T	F	T	F	F
T	F	F	T	T	T
F	F	F	F	F	F

3)(b)

P	Q	R	$P \wedge Q$	$P \wedge R$	$(P \wedge Q) \vee (P \wedge R)$
T	T	T	T	T	T
T	T	F	T	F	T
T	F	T	F	T	T
F	T	T	F	F	F
F	F	T	F	F	F
F	T	F	F	F	F
T	F	F	F	F	F
F	F	F	F	F	F

$P \wedge (Q \vee R) = (P \wedge R) \vee (P \wedge Q)$

Answer sheet:

$P \wedge (Q \vee R) = \neg\neg P \wedge (\neg\neg Q \vee \neg\neg R)$

$= \neg\neg P \wedge \neg(\neg Q \vee \neg R)$ De morgan's Law

$= \neg(\neg P \vee (\neg Q \vee \neg R))$ De morgan's Law

$= \neg((\neg P \vee \neg Q) \wedge (\neg P \vee \neg R))$

$= \neg(\neg(P \wedge Q) \wedge \neg(P \wedge R))$ De morgan's Law

$= \neg\neg(P \wedge Q) \vee \neg\neg(P \wedge R))$ De morgan's Law

$= (P \wedge Q) \vee (P \wedge R)$

** De morgan's Law:

↳ $\neg(P \vee Q) = \neg P \wedge \neg Q$ } change: AND ↔ OR

↳ $\neg(P \wedge Q) = \neg P \vee \neg Q$

4) $(P_1 \wedge \dots \wedge P_n) \Rightarrow C \equiv \neg P_1 \vee \dots \vee \neg P_n \vee C$

using de morgan's law on $\neg(a \wedge b) = \neg a \vee \neg b$

$(P_1 \wedge \dots \wedge P_n) \Rightarrow C = \neg(P_1 \wedge \dots \wedge P_n) \vee C$

$= \neg P_1 \vee \dots \vee \neg P_n \vee C$

** implication $A \Rightarrow B$ is equivalent to $\neg A \vee B$

** equivalence $A \Leftrightarrow B$ is equivalent to $(A \Rightarrow B) \wedge (A \Leftarrow B)$

which is $\longrightarrow$ $(\neg A \vee B) \wedge (A \vee \neg B)$

5) J = Judith ; m = margaret ; K = katherine

(a) $\neg J \wedge \neg m$

(b) $(\neg J \wedge m) \vee (\neg m \wedge J)$

(c) $(J \wedge m \wedge K)$

(d) $(\neg J \wedge \neg m) \vee (\neg J \wedge \neg K) \vee (\neg m \wedge \neg K)$

(e) $(J \wedge m \wedge K) \vee (\neg J \wedge \neg m) \vee (\neg J \wedge \neg K) \vee (\neg m \wedge \neg K)$

(f) $(J \wedge m) \vee (J \wedge K) \vee (K \wedge m)$

(g) $\neg J \wedge \neg m \wedge \neg K$

(h) $(\neg J \wedge \neg m \wedge \neg K) \vee (J \wedge m) \vee (J \wedge K) \vee (m \wedge K)$

(i) $J \wedge m \wedge K$

(j) $\neg J \wedge \neg m \wedge \neg K$

6) If graph is a tree, then its bipartite & have a leaf

tree $\Rightarrow$ bipartite + leaf $\quad \neg$tree $\vee$ (bipartite $\wedge$ leaf)

If graph is not a tree, then vertex of degree ≥ 2

$\neg$tree $\Rightarrow$ vertex of degree ≥ 2 $\quad \neg(\neg$tree$) \vee$ internal

$\hookrightarrow$ tree $\vee$ internal

$\therefore$ ($\neg$tree $\vee$ bipartite) $\wedge$ ($\neg$tree $\vee$ leaf) $\wedge$

(tree $\vee$ internal)

7) Perfect matching
↳ no 2 edges share a vertex
** Assignment 1

8) (i) taller(father(max), max) ∧
¬taller(father(max), father(claire))
(ii) ∃x taller(x, father(claire))
(iii) ∀y ∃x (taller(y, x))
(iv) ∀x (taller(x, claire) ⇒ taller(x, max))

9) (a) $\exists n \forall y : y \in L \Rightarrow |y| < n$
(b) $\forall n \exists y : y \in L \wedge |y| > n$

Exercise 2 (Tutorial sheet 2)
↳ Quantifiers
↳ Proofs

4) For all $n \geq 1$, every tree on n vertices has n-1 edges

Base case: $n = 1$

vertex = 1

edges = (1) - 1 = 0

1 vertex indeed can't have any edges connecting as the other end of the vertex connects to nowhere.

Inductive hypothesis:

For all $k \geq 1$, every tree on k vertices has k-1 edges

Inductive step:

k+1 vertices have (k+1) - 1 edges

every tree have a leaf if there's ≥2 vertices removing a leaf from trees with ≥2 vertices provides tree with 1 less leaf which is 1 less vertex and 1 less edge.

T is a tree, removing 1 leaf is like having k vertiex and from the base case it is held true and hence T with 1 less leaf is k-1 edges and k vertices. Removing 1 edge is (k-1) - 1 edges which is k-1 edges and k-1 edges is less than K vertices which shows that edges < vertices by 1.

Conclusion:

By the principle of mathematical induction, For all n, every tree with n vertices has indeed n-1 edges.

5) For all $n \geq 3$, $n! \leq (n-1)^n$

Base case: $n = 3$

$3! \leq (3-1)^3$

$3 \times 2 \times 1 \leq (3-1)(3-1)(3-1)$

$6 \leq (2)(2)(2)$

$6 \leq 8$

6 is indeed less than 8 so when $n = 3$, $n! \leq (n-1)^n$ is true.

Inductive hypothesis:

Assume For all $k \geq 3$, $k! \leq (k-1)^k$ is TRUE.

Inductive step:

$k+1 \geq 3$, $(k+1)! \leq ((k+1)-1)^{k+1}$ — ①

$(k+1)! = (k+1)(k!) \leq (k+1)(k-1)^k$

$= (k+1)(k-1)(k-1)^{k-1}$

$= (k^2-1)(k-1)^{k-1}$

$= k^2(k-1)^{n-1}$

$= k^{k+1}$

$= ((k+1)-1)^{k+1}$ — ②

① = ②

Hence the assumption is held true

Conclusion:

By the principle of mathematical induction, the assumption is held true for all $n \geq 3$, $n! \leq (n-1)^n$

Exercise 3 (Tutorial sheet 3)

↳ Regular languages

↳ Finite Automata (FA)

↳ Kleene's Theorum

1) abab, baab, abaaaab, abbbbab, baaaaab, babbab, abaaaaaab, abbbbbbab, babbbbbab, abbbaaaab, baaaaaaab, babbbbbab, ababbbab, abbbaaaab, baaabbab, babbaaab

2) (i) $aa \cup ab \cup ba \cup bb$

(ii) 2 bs $\rightarrow a^*ba^*ba^*$

3 bs $\rightarrow a^*ba^*ba^*ba^*$

(iii) $(a \cup b)^*(aa \cup bb)$

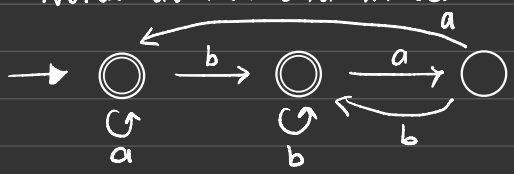
(iv) $(a \cup b)^*(ab \cup ba) \cup a \cup b \cup \varepsilon$

(v) $(b \cup \varepsilon)(ab)^*aa(ba)^*(b \cup \varepsilon) \cup (a \cup \varepsilon)(ba)^*bb(ab)^*(a \cup \varepsilon)$

(vi) $(aa \cup ab \cup ba \cup bb)^*$

(vii) $a^*((b \cup bb)aa^*)^*(b \cup bb \cup \varepsilon)$

4) Words do not end in ba (FA)



5) (a) 1, 3, 4, 5, 6

(b) 3, 6

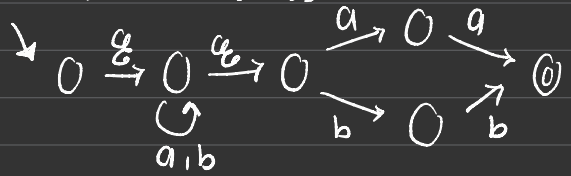
(c) accepted, end at 6

(d) 1 2 1 4 2 5 4 1 2 3 3 6

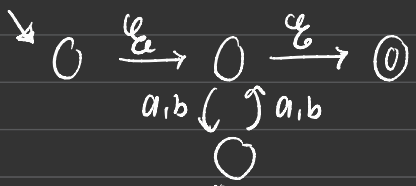
(e) base case (a), inductive (d), conclusion

Regular expression → NFA

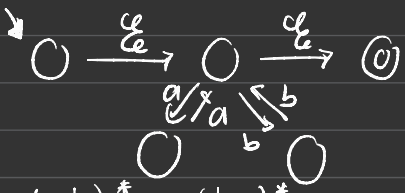
6)(i) $(a \cup b)^*(aa \cup bb)$



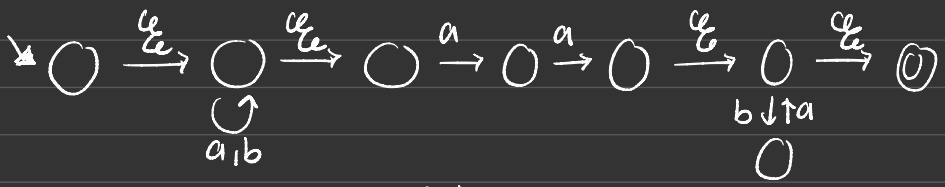
(ii) $((a \cup b)(a \cup b))^*$



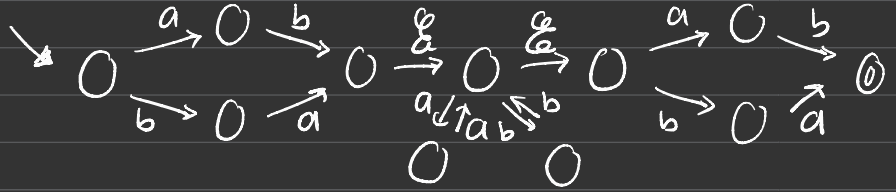
(iii) $(aa \cup bb)^*$



(iv) $(ab)^*aa(ba)^*$

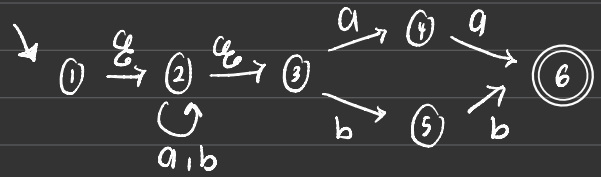


(v) $(ab \cup ba)(aa \cup bb)^*(ab \cup ba)$



NFA → FA

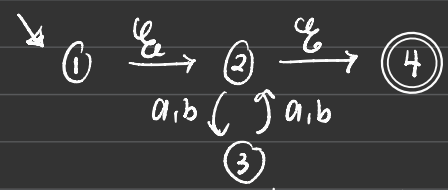
7)(i) $(a \cup b)^*(aa \cup bb)$



	a	b
start {1,2,3}	{2,3,4}	{2,3,5}
{2,3,4}	{2,3,4,6}	{2,3,5}
{2,3,5}	{2,3,4}	{2,3,5,6}
Final {2,3,4,6}	{2,3,4,6}	{2,3,5}
Final {2,3,5,6}	{2,3,4}	{2,3,5,6}

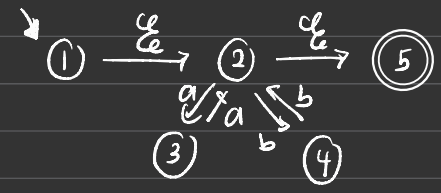
		a	b
S	1	2	3
	2	4	3
	3	2	5
F	4	4	3
F	5	2	5

(ii) $((a \cup b)(a \cup b))^*$



	a	b
start {1,2,4}	{3}	{3}
{3}	{2,4}	{2,4}
Final {2,4}	{3}	{3}

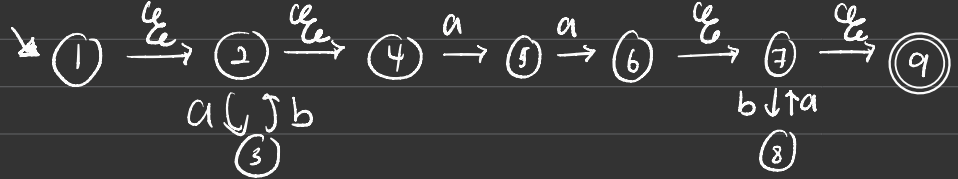
(iii) $(aa \cup bb)^*$



	a	b
start {1,2,5}	{3}	{4}
{3}	{2,5}	{∅}
{4}	{∅}	{2,5}
Final {2,5}	{3}	{4}
{∅}	{∅}	{∅}

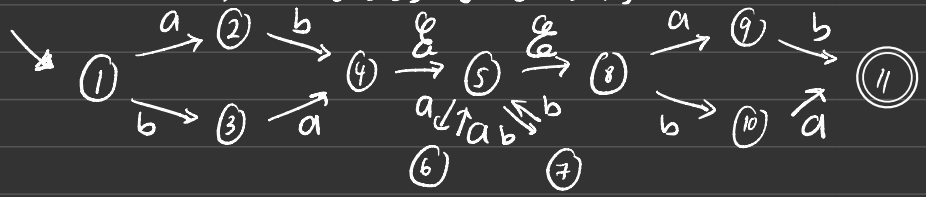
NFA → FA

(IV) $(ab)^* aa(ba)^*$



		a	b
start	{1,2,4}	{3,5}	{∅}
	{3,5}	{6,7,9}	{2,4}
Final	{6,7,9}	{∅}	{8}
	{2,4}	{3,5}	{∅}
	{8}	{7,9}	{∅}

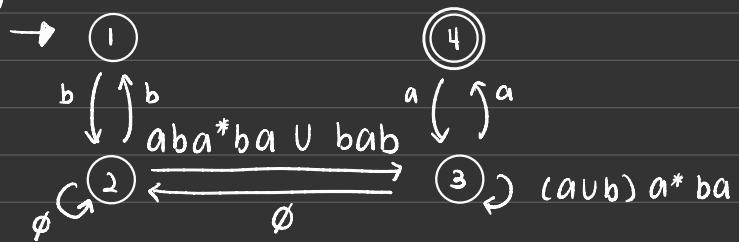
(V) $(abU\,ba)(aaU\,bb)^*(abU\,ba)$



		a	b
start	{1}	{2}	{3}
	{2}	{∅}	{4,5,8}
	{3}	{4,5,8}	{∅}
	{4,5,8}	{6,9}	{7,10}
	{6,9}	{5,8}	{11}
	{7,10}	{11}	{5,8}
Final	{11}	{∅}	{∅}
	{5,8}	{6,9}	{7,10}

Exercise 4 (Tutorial sheet 4)

1)



3)

state	a	b
Start 1	2	3
2	1	5
3	1	4
Final 4	2	5
Final 5	3	5

state	a	b
Start 1	2	3
2	1	5
3	1	4
Final 4	2	5
Final 5	3	5

minimization :

state	a	b
Start 1	2	2
2	1	4
Final 4	2	4

4)

state	a	b
Start 1	2	4
2	2	6
3	3	4
Final 4	5	3
Final 5	5	5
Final 6	5	1

state	a	b
Start 1	2	4
2	2	6
3	3	4
Final 4	5	3
Final 5	5	5
Final 6	5	1

state	a	b
Start 1	1	4
Final 4	5	1
Final 5	5	5

5) last week 7(i) is unique states with no minimisation simplification possible

6) GNFA → regular expression
last week 7(ii)

① DFA

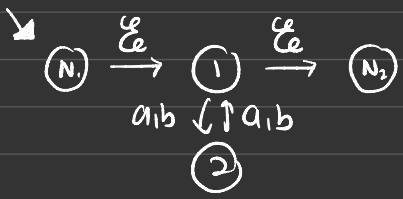
		a	b
Start	{1,2,4}	{3}	{3}
	{3}	{2,4}	{2,4}
Final	{2,4}	{3}	{3}

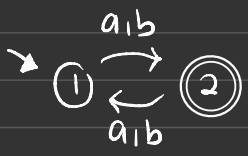
		a	b
Start	1	2	2
Final	2	2	2

② DFA

		a	b
Start	{1,2,4}	{3}	{3}
	{3}	{2,4}	{2,4}
Final	{2,4}	{3}	{3}

		a	b
Start	1	2	2
Final	2	1	1





↳ (aa ∪ ab ∪ ba ∪ bb)*
↳ ((a ∪ b)(a ∪ b))*

6.

We just do (ii) as the others involve long computations.

First, turn it into a GNFA by adding a new Final State 3 and an ε-transition from the Start State to it. The Start State is no longer a Final State.

Similarly, a new Start State is added, with a new empty string transition from it to the old Start State.

Then, applying the algorithm, we obtain (aa ∪ ab ∪ ba ∪ bb)*. This is equivalent to the original regular expression ((a ∪ b)(a ∪ b))*, and describes the language of all strings of even length.

* sets minimisation look at last 2 column values
↳ same value = can merge

Lecture 12

Assessed Preparation : Question 5

(a) VERY-EVEN is a regular language

(b) N is a positive integer such that any string in VERY-EVEN with at least N can be divided into 3 parts x, y and z where for any $i \geq 0$, xy^{iz} is also included in VERY-EVEN.

Assumption: There exists a pumping length N and VERY-EVEN is regular.

(c) $w = 10^{2N}$

↳ w = string, length $\geq N$

↳ binary number 10^{2N} is a multiple of 2^{2N} with length $2N+1$

↳ satisfy length bound of at most 3N bits

(d) For any placement of non-empty substring y, y can only consist of '0's

$w = 10^{2N}$, substring y can only be sequence of '0's with length $\leq N$

(e) For any placements of substring y within w, consider xy^{zi} and if $i = 2$, string $= 10^{2N+2y}$

$y \leq N$ hence $2N+2y$ is greater than 3N which don't follow VERY-EVEN length constraint

(f) (e) shows that VERY-EVEN is not a regular language

↳ pumping lemma conditions violated

(g) string $= 10^{2N+N}$

↳ VERY-EVEN, multiples of 2^{3N}, length $3N+1$ (length bound satisfied)

(h) N/A (not sure how to do)

(i) because (h) not sure how to do, not sure how to proceed with (i)

Assessed Preparation : Question 3

3)(a) $S \to i$

$S \to (S)$

$S \to S+S$

$S \to S-S$

(b) $(i+(i-i))-i$

$S \to (S)+S$

$(S+S)-i$

$S \to (S \to S)$

$(S+(S-S))-i$

$S \to i$

$(i+(S-S))-i$

$(i+(i-S))-i$

$(i+(i-S))-i$

$(i+(i-i))-i$

(c) $E \to d$

$E \to E+i$

$E \to E-i$

$E \to (E)$

$E \to E-E$

(d) Not regular. Not recognised by FA.
Date expression involve anthmetic operations and nested parentheses which is not recognised by FA or regular expression.
Balancing anthmetic operations and nested parentheses require context-free-grammar. Subtraction operations done by date expressions results in larger integer. Hence, making it further non-regular

Non-terminal → nonterminal nonterminal

Non-terminal → terminal

Steps

① remove ε production rules $x \to \varepsilon$

② remove unit production $X \to Y$

③ giving each terminal its own non-terminal

④ inputing the new non-terminal you've created

⑤ getting the rules with more than 2 non-terminals

1)

① $S \to aScc$	① $S \to aScc$
② $S \to X$	② $S \to aXcc$
③ $X \to bX$	③ $X \to bX$
④ $X \to b$	④ $X \to b$

prove by induction :

Base case :
abCC is the only word in language word length 4 that can be derived from the grammer.
① $S \to aXcc$
$\to abcc$

Induction hypothesis :
For all stung of the form $a^n b^i c^{2n}$ can be generated by the grammer

Inductive step :
let $k > 4$ for all t of $4 \leq t < k$
assume for all stung of length t of the form $a^n b^i c^{2n}$ can be generated by the grammer
need to show that all stungs of length $t+1$ of the form $a^n b^i c^{2n}$ can be generated by the grammer

let w be a string of length $k+1$ of the form $a^n b^i c^{2n}$

assignment: take arbitary word from it, step down

either $n > 1$ OR $i > 1$

If $i > 1$ = more than 1 b

let w^* be the string with 1 less b, it is still $\in L$

$|w| = k$ so by IH we have a derivative

↳ inductive hypothesis

the last rule used is ④

add new application of rule ③ b4 final rule ④

this generates w.

if $i = 1$ then $n > 1$ which means there's > 1 a

make w^* be string with 1 less a and 2 less c

length $|w^*| = k - 2$

by our inductive hypothesis

first rule in derivation of w^* is ①

+ additional rule ② after rule ① after rule ①

at the start which generates w

by the principle of mathematical induction...

Assessed preparation: Question 4

(a)

State	Input	Write	Move	Next State
q0	a	a	R	q0
q0	b	a	L	q1
q1	a	a	R	q1
q1	b	x	L	q1
q1	-	-	R	q2
q2	a	a	R	q2
q2	b	b	R	q3
q2	-	-	L	q4
q3	a	a	R	q3
q3	b	b	R	q3
q3	-	a	L	q5
q4	a	a	L	q4
q4	b	b	L	q4
q4	-	b	L	q5
q5	a	a	L	q5
q5	b	b	L	q5
q5	-	-	S	Accept state

(b)

(c)

(d) maximum number of steps $t(n)$

n = string length For all n, $t(n) \geq n$

Worst case scenario: traverse entire input twice

↳ once replace last letter, another replace first letter

$t(n) \approx 2n$ <= best upper bound expression