Comp2022 cheatsheet

COMP2022 Models of Computation (University of Sydney)

RE

A language $L \subseteq \Sigma^*$ is called **regular** if $L = L(M)$ for some DFA M, where $L(M)$ is the language produced by that DFA M.

That is, a language is **regular** if every string in it can be accepted by one DFA.

Regular languages are closed under:

- Complement, if L is regular, then $\Sigma^* \backslash L$ is regular.
- Union, if L_1 and L_2 are regular, then $L_1 \cup L_2$ is regular.
- Intersection, if L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.
- Concatenation, if L_1 and L_2 are regular, then $L_1 L_2$ is regular.
- Kleene star, if L is regular, then L^* is regular.

For $\Sigma = \{a, b\}$, $(a|b)^*a$ means any string ending in a.

DFA

e.g $\Sigma(\Sigma\Sigma)^* \Rightarrow$ all odd strings

Deterministic finite automata, has:

$\Sigma^* 1 \Rightarrow$ all strings ending in 1

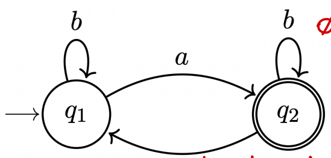
- Q: finite set of states
- Σ: alphabet
- δ: $Q \times \Sigma \to Q$: transition function if $\delta(q, a) = q'$, we write $q \xrightarrow{a} q'$, a transition
- $q_0 \in Q$: initial state
- $F \subseteq Q$: final/accepting states

$\Sigma^* 11 \Sigma^* \Rightarrow$ all strings contain a 11

$1^*1(1^*01^*)^* \Rightarrow$ 0 must pad el

$\varepsilon \Rightarrow$ empty string of length 0

$\emptyset \Rightarrow$ set of no strings



For example, this DFA accepts all strings that end in a.

closed under union, complement, intersection, star, concatenation (de Morgan)

State	a	b
q_1	q_2	q_1
q_2	q_2	q_1

Note the transition function can be written as a table.

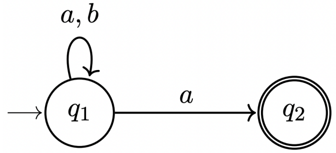
- Membership Problem: For a DFA M and a string w, decide if $w \in L(M)$.
- Non-emptiness Problem: For a DFA M, decide if $L(M) = \emptyset$.
- Equivalence Problem: For two DFAs M_1 and M_2, decide if $L(M_1) = L(M_2)$.

NFA

Nondeterministic finite automata, has the same ingredients as DFA. NFA is a DFA except states can have zero, one, or more outgoing transitions on the same input symbol.

We also allow epsilon-transitions, which are transitions that do not use the next symbol.

The input is accepted if at least one of it's runs is accepting. • traverse like a tree

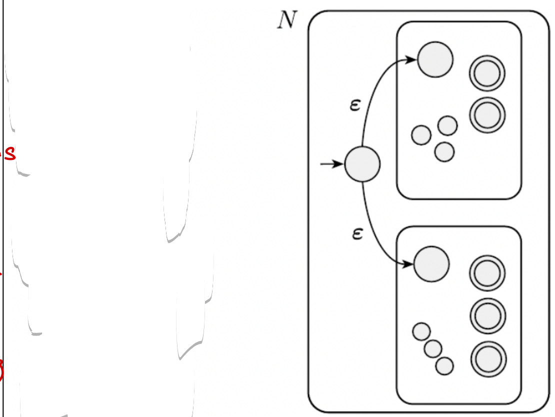


For example, this NFA accepts all strings that end in a.

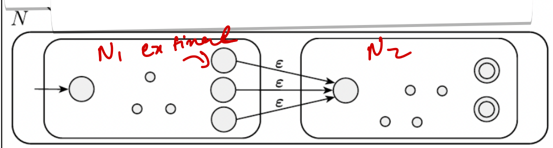
For every DFA M, there is at least one NFA N where $L(M) = L(N)$.

NFAs are closed under complement, union, intersection, concatenation and Kleene star. NFAs only recognise RE.

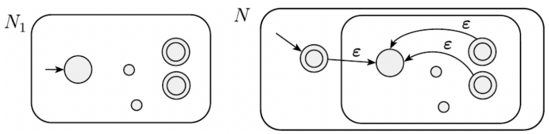
(Union) NFA N that recognises $L(N_1) \cup L(N_2)$:



(Concatenation) NFA N that recognises $L(N_1)L(N_2)$:

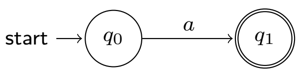


(Kleene star) NFA N that recognises $L(N_1)^*$:

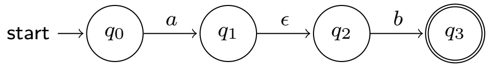


RE to NFA

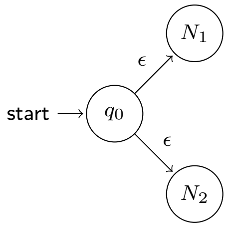
NFA for $R = a$, $a \in \Sigma$:

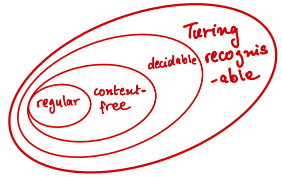


NFA for $R = ab$, $a, b \in \Sigma$ (concatenation):

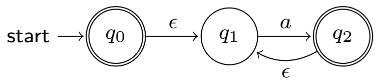


NFA for $R = a \mid b$, $a, b \in \Sigma$ (union): NFA ≡ DFA





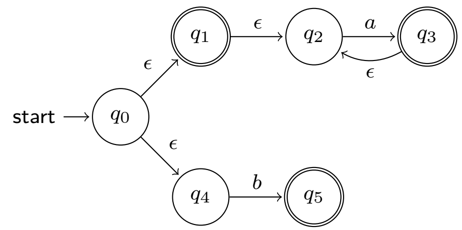
NFA for $R = a^*$, $a \in \Sigma$ (Kleene star):

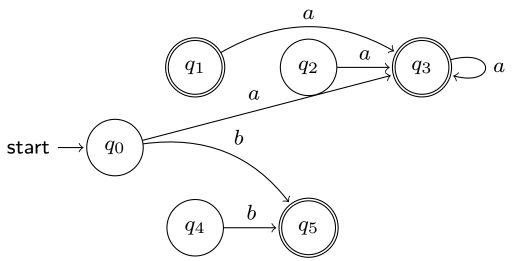


We can put these NFA's together to build an NFA that expresses the a required RE.

NFA to NFA without ϵ-transitions $O(n)$

Simply remove all epsilon-transitions, and add final states if necessary. For example:

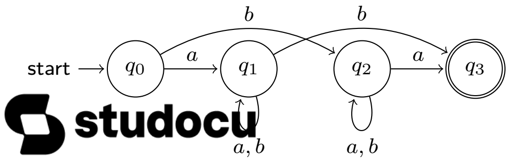




Note we do not remove states when removing epsilon transitions.

NFA without ϵ-transitions to DFA $O(2^n)$

We make the transition table of the NFA, then relabel states until we have a DFA. For example:



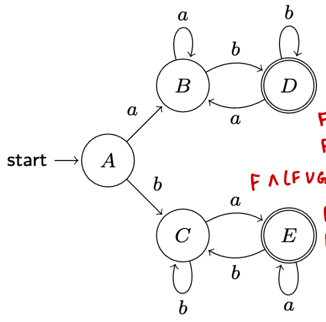
Original NFA:

State	a	b
q_0	q_1	q_2
q_1	q_1	q_1, q_3
q_2	q_2, q_3	q_2
q_1, q_3	q_1	q_1, q_3
q_2, q_3	q_2, q_3	q_2

Relabelling states to get DFA:

State	a	b
A	B	C
B	B	D
C	E	C
D	B	D
E	E	C

Start state is A, or q_0 from NFA. Accept states are D and E, since they had the accept state q_3 from NFA.



$F \equiv F \wedge F \equiv F \vee F$

$F \wedge G \equiv G \wedge F$

$F \vee G \equiv G \vee F$

$F \wedge (G \wedge H) \equiv (F \wedge G) \wedge H$

$F \vee (G \vee H) \equiv (F \vee G) \vee H$

$F \wedge (F \vee G) \equiv F \equiv F \vee (F \wedge G)$

$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$

$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$

$\neg(F \wedge G) \equiv \neg F \vee \neg G$

$\neg(F \vee G) \equiv \neg F \wedge \neg G$

$\neg\neg F \equiv F$

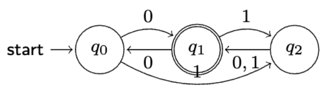
$F \vee T \equiv T$

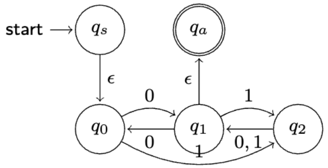
DFA to RE

First convert the DFA to a GNFA: $O(4^n)$

- Instead of a, b in a transition, write $a \mid b$.
- Add initial state and ϵ-transition to original initial state, and final states.

Then eliminate states until only 2 states left with 1 transition. For example:





$F \wedge T \equiv F$

$F \vee \bot \equiv F$

$F \wedge \bot \equiv \bot$

$T \equiv (F \vee \neg F)$

$\bot \equiv (F \wedge \neg F)$

$\neg T \equiv \bot$

$\neg \bot \equiv T$

$(F \to G) \equiv \neg F \vee G$

$(F \leftrightarrow G) \equiv (F \to G) \wedge (G \to F)$

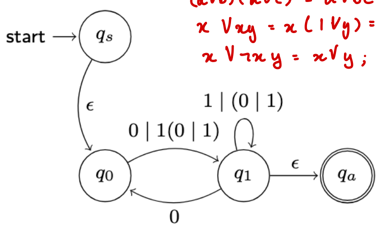
$p \to q \equiv \neg q \to \neg p$

Eliminate state q_2:

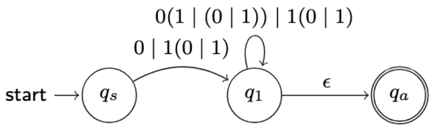
$(a \vee b)(a \vee c) = a \vee bc$

$x \vee xy = x(1 \vee y) = x$

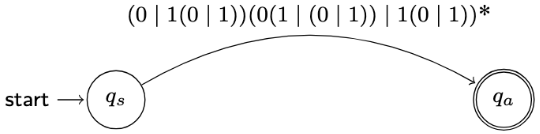
$x \vee \neg x y = x \vee y$; $x(\neg x \vee y) = xy$



Eliminate state q_0:



Eliminate state q_1:



So we get RE (0 | 1(0 | 1))(0(1 | (0 | 1)) | 1(0 | 1))*

Prove a language is not regular

We can conclude $L(M) \neq L$ for a language L and any DFA M; if we can find strings x and y that go to the same state in M, then find another string z such that $xz \in L$ but $yz \notin L$.

That is there is no DFA for L and L is not regular.

For examample, $L = \{a^n b^n : n \geqslant 0\}$ is not regular:

1. Let $x_i = a^i$, $i \in \mathbb{N}$
2. For $i \neq j$, let $z = b^i$
3. Then $x_i z = a^i b^i \in L$
4. But $x_j z = a^j b^i \notin L$

• or use pigeonhole : if k states in DFA, then consider x (i) with k+1 strings. Using pigeon hole, atleast 2 unique strings in x (i) must be in same state : x(a), x(b). x(a)z∈L but x(b)z∉L i.e. contradiction.

CFG

Context free grammars, has:

closed under:
- union $S_1 \cup S_2 = S \to S_1 | S_2$
- concatenation $S_1 S_2 = S \to S_1 S_2$

- Variables, S, T
- Terminals (input symbols), a, b, c
- Rules, 3 in this case — star $S^* = S \to SS | \epsilon$
- Start variable S

Predicate : all A's are B's $\forall x (A(x) \Rightarrow B(x))$

$S \to aSb$ — some As are Bs $\exists x (A(x) \wedge B(x))$

no As are Bs $\forall x (A(x) \Rightarrow \neg B(x))$

$S \to T$ — some As are not Bs $\exists x (A(x) \wedge \neg B(x))$

$T \to c$ — everyone loves someone $\forall x \exists y (\text{loves}(x,y))$

The grammar derives strings of terminals as following:

1. Write the starting variable
2. Repeat the following until no variables are written:
   (a) Pick a variable X that is written down
   (b) Pick a rule $X \to \ldots$
   (c) Replace X with the RHS of the rule
3. The remaining string is derived from the grammar

For example: $S \implies T \implies c$

$S \implies aSb \implies aTb \implies acb$

Note a leftmost derivation always replaces the leftmost variable in the grammar at step 2(a).

Derivation notation: someone loves everyone $\exists x \forall y (\text{loves}(x,y))$

$\implies$ derives in 1 step

$\overset{n}{\implies}$ derives in n steps

$\overset{0}{\implies}$ derives in 0 or more steps

$\overset{+}{\implies}$ derives in 1 or more steps

$\neg \forall x F \equiv \exists x \neg F$
$\forall x F \wedge G \equiv \forall x (F \wedge G)$
$\forall x \forall y F \equiv \forall y \forall x F$
$\exists x \exists y F \equiv \exists y \exists x F$

A language is called context-free if it can be derived from a CFG.

• $\wedge$ goes with $\exists$
• $\to$ goes with $\forall$ } usually

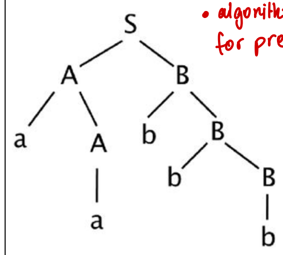
Parse Trees

$\forall x F \wedge G \equiv \forall x (F \wedge G) \equiv \vee$

$\exists x F \wedge G \equiv \exists x (F \wedge G) \equiv \vee$

- Root is labelled by start variable
- Nodes are labelled by variables
- Children are labelled by RHS of a rule
- Leaf nodes are a terminal or ϵ
- Read the string off the leaves, left to right

• algorithm for checking validity for predicate logic is TM recognisable ! not decidable.



$S \to AB$
$A \to aA \mid a$
$B \to bB \mid b$
Parse tree for string $aabbb$

$\forall x (P(x) \vee \neg P(x))$ is valid.

Ambiguous strings have $\geqslant 2$ parse trees in one CFG. A grammar is ambiguous if it generates at least 1 ambiguous string.

CFG to CNF

A CFG is in Chomsky Normal Form if every rule is in one of these forms:

1. $A \to BC$ — B,C not start
2. $A \to a$ — terminal
3. $S \to \epsilon$ — only start variable for empty string

We do **not** allow forms like $A \to 0A1$, $A \to B$, $A \to ab$, or the start variable on the RHS. $A \to aA$

We can use any amount of | in a CNF.

Note parse trees of a grammar in CNF are binary trees.

Every CFG is guaranteed to have a CNF.

To turn a CFG into CNF:

1. START: Eliminate start variable from RHS of all rules
2. TERM: Eliminate rules with terminals (except those of form $A \to a$)
3. BIN: Eliminate rules with more than 2 variables — $A \to BCD$: $A \to BN_1$, $N_1 \to CN_2$ ...
4. EPSILON: Eliminate epsilon rules ($A \to \epsilon$), except $S \to \epsilon$

   If $B \to uAv$ and $A \overset{+}{\implies} \epsilon$, then $B \to uv$
5. Eliminate unit rules ($A \to B$)

CYK $O(n^3)$

Find if any CNF CFG accepts a string. Example, string $aabb$:

S,T $aabb$			
$\varnothing$ aab	X abb		
$\varnothing$ aa	S,T ab	$\varnothing$ bb	
A a	A a	B b	B b

For CNF CFG:
$S \to AB \mid AX \mid \epsilon$
$T \to AB \mid AX$
$X \to TB$
$A \to a$
$B \to b$

Runs in $O(|w|^3)$ time for fixed CFG, varying w.

TM

- Can only use variables (states) with finite domains
- Has infinite tape, made of cells
- There is a pointer/head on the tape
- The pointer can move left, right or stay still
- The pointer can read and write symbols from a bigger alphabet (tape alphabet, Γ)
- At first, the input string is written on the tape
- Depending on the state, the machine can decide to halt (stop running) and halt-accept or halt-reject
- Deterministic
- The language recognised by TM M is the set of strings it accepts

Each instruction in a TM is of form:

<current state><current symbol><new symbol><direction><new state>

E.g. **q1 a b R q2**, tells the TM if it reads an a and is in state q1, write a b, move right, and change to state q2.

Symbols are the tape alphabet Γ, which include the input alphabet Σ and blank ␣

Directions are L, R, and * (stay still, or S).

The machine stops when it reaches any state starting with "halt", i.e. halt-accept, halt-reject.

A TM can fail to accept an input for 2 reasons:

1. its computation is **rejecting**, i.e. halt-reject
2. its computation is **diverging**, i.e. it never halts

A language is **Turing-recognisable** if some TM M recognises it, that is it accepts all inputs in $L(M)$, but does not halt for all other inputs. → not closed under complement

A language is **Turing-decidable** if some decider (TM that halts on all inputs) recognises it.

A TM runs in time $f(n)$ if $f(n)$ is the largest number of steps taken by the TM for any input of length n.

Must-move TM: Head cannot stay, must move. Every basic TM is equivalent to a must-move TM.

Left-bounded TM: Head starts at left-most cell of the tape, and the head cannot move further left than that point. Every basic TM is equivalent to a left-bounded TM.

Multitape TM: Multiple tapes, each with their own head, heads move simultaneously. Every basic TM is equivalent to a multitape TM.

Decidable languages are closed under complement, union and intersection. non-decidable L are also closed under complement.

A language is decidable when it and its compliment are recognisable. TM decidable ⇔ TM's L( ) AND complement are RECOGNISABLE.

NTMs: Nondeterministic TM, multiple possible transitions at one state, kinda like NFA.

P vs NP

P is closed under union

P: Collection of languages decidable in polynomial time on **deterministic TMs**. Every RE and context-free language is in P. DFA membership, RE membership, DFA equivalence, CFG membership and CFG emptiness is in P.

NP: Collection of languages decidable in polynomial time on **nondeterministic TMs**. All languages in P are in NP. The CLIQUE problem is in NP. The **halting problem** (given an algorithm/program, find if it will ever halt) is in NP.

Propositional Logic

$\wedge$ conjunction (and), $\vee$ disjunction (or), $\neg$ negation (not). $\to$ implication, $\leftrightarrow$ bi-implication. $\top$ true. $\perp$ false.

An atom is a variable, and a formula is recursively by:

1. Every atom is a formula — a ⊨ F means a(F) = 1
2. If F is formula, $\neg F$ is formula
3. If F, G are formula, $F \vee G$ and $F \wedge G$ are formulas

F	G	$(F \to G)$
0	0	1
0	1	1
1	0	0
1	1	1

A formula is **valid** if every assignment satisfies it.
A formula is **satisfiable** if at least one assignment satisfies it.
Satisfiability problem, decide if a formula is satisfiable, is in NP.
$O(2^n)$ checking truth table

To show F is a logical consequence $\models$ of $E_0, \ldots, E_k$ we use truth tables, see if every assignment that satisfies all formulas E_i also satisfies F.

NNF: Negation Normal Form, where $\neg$ negations only occur in front of atoms $\neg(a \vee b)$ not allowed

CNF: Conjunctive Normal Form. Conjunction $\wedge$ of clauses (literal or disjunction $\vee$ of literals, a literal being p or $\neg p$).

$(a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c)$

Predicate Logic

Instead of "x is even" we write even(x). We cannot compose predicates, e.g. odd(prime(x)) is not a formula and has no meaning. We can use propositional logic syntax.

We have two quantifiers, $\exists x F$, existential qualifiers (there is an element in the domain that F is true when d replaces x), and $\forall x F$ for all, universal qualifiers (for all elements d in the domain, F is true when d replaces x).

Bound and free variables ex: $\forall \underline{x}(P(x, y) \to \exists \underline{y} Q(x, y, z))$