Sets  
Cardinality   
Sx = { 1, 2, 3 }  
|Sx| = |{ 1, 2, 3 }| = 3  
Prove equality  
A=B iff A⊆B and B⊆A  
Relations - Equivalence  
Reflexive: ∀x ∈ A, (x, x) ∈ R  
Symmetric: ∀x, y ∈ A, if (x,y) ∈ R, (y, x) ∈ R  
Transitive: ∀x, y, z ∈ A, if (x,y) ∈ R and (y, z) ∈ R, (x, z) ∈ R  
Relation R ⊆ A x B is a function iff maps exactly 1 element of A to B  
R1 = {(a,x), (a,y)} (not a function) R2 = {(a,x), (b,y)} (function)  
Closure of Relation  
Relation R ⊆ A x A is closed under some property if it possesses that property  
v = or, ^ = and

Truth  
Rules  
Modus ponens: (P -> Q) and P, then Q   
Modus tollens: (P -> Q) and ¬Q, then ¬P  
Quantifier exchange:   
From ¬∃x(P), conclude ∀x(¬P)   
From ∀x(¬P), conclude ¬∃x(P)   
From ¬∀x(P), conclude ∃x(¬P)   
From ∃x(¬P), conclude ¬∀x(P)  
Construction, contradiction, counterexample, enumeration  
Induction: base (n=1, or lowest value), assume (n=k), induct (n=k+1)

General Definitions  
Alphabet (∑) is a finite set of symbols  
String is a finite sequence of symbols chosen from alphabet ∑  
Language is a (finite/infinite) set of strings chosen from alphabet ∑

Languages  
Languages are sets with generators and recognisers, e..g { w ∈ { a, b} : w is even }  
Small cardinality over Σ (L = **Ø**) is 0, largest is countably infinite  
If Σ =/= **Ø, the set of languages over** Σ (𝒫 Σ\*) is uncountably infinite  
Kleene star = 0 or more concatenations   
Kleene plus = 1 or more concatenations (L+ = L\* - ɛ) iff L =/= ɛ  
Lexicographical order = smaller string first, then alphabetical  
Power set – 𝒫(L)  
𝒫(L) = { w : w ⊆ L } Includes empty string  
Intersection of two power sets is not an empty set, it is a set containing one element { **Ø** }  
L = { s : s ∈ Σ\* ∃ w1, w2 (w1 =/= w2 ^ |w1| = 2 ^ |w2|= 2 ^ substr(w1, s) ^ substr(w2, s)) }   
Prefix – first prefix always ɛ

Closure

Binary relation *R* on set A is closed under property *P* iff *R* possesses *P.*  
The closure of R under P is the smallest set that obeys this.

FSM  
DFSM quintuple: M = (K(states), Σ(alphabet), ð(transition function from (K × S) to K), s(start), A(accepting))   
NDFSM quintuple: M = (K, Σ, ð(transition relation (K × (S ∪ {ε}) × K)), s, A)  
Configuration: an element of K x Σ\* - captures current state and future input  
Computation: finite sequence of configurations for n => 0 == C0 |-M C1 |-M etc

**mindfsm**  
-active classes = { A, K-A }  
-get transitions out of each class; results are equivalence classes  
-split equivalence classes if patterns don’t match, repeat old steps if this affects them

**ndfsmtodfsm**  
-compute eps(q) for each state q  
-get ɛ-trans of start state  
-compute ð’ by getting trans of each following state; add eps() for each new created state  
-accepted states are any new states containing any elements of the old state

**fsmtoregex**  
-remove unreachable states  
-return **Ø** if no accepting states  
-return ɛ if only one state  
-if start state is part of a loop, create new start with ɛ-tran to old start  
-if > 1 transitions or any transitions out, create a new accepting state with ɛ-trans to new state; old states no longer accept  
-if any transitions do not exist, make them (standardise)  
-if there are is more than one transition between 2 states, collapse them.  
-rip ==== (not start or accept), for all transitions, including new ones

Regular Expressions and Grammar  
RE maps to FSM  
REs include **Ø,** ɛ, every element of Σ, ab/a∪b (if a and b are REs), a\*, a+ and (a) if a is RE  
Examples:  
L = { w ɛ {a, b}\* : |w| is even }  
((a∪b)(a∪b)\*) = (aa ∪ ab ∪ ba ∪ bb)\*   
L = { w ɛ {a, b}\* : w contains an odd number of a’s}  
b\*ab\*(ab\*ab\*)\*  
((a∪b)(a∪b)\*) = (aa ∪ ab ∪ ba ∪ bb)\*  
Regular grammar is a quadruple  
Operator precedence: Kleene star->concat->union  
**Regular Language closure properties:** Union, concatenation, Kleene star, Complement, Intersection, Difference, Reverse, Letter substitution

Kleene’s theorem (Week 5 – 50:00)  
If a is the regex b ∪ c and L(b) and L(c) are regular,   
M3 = L(M3)=L(a)=L(b) ∪ L(c) = ({s3} ∪ K1 ∪ K2, Σ, ð3= ð1 ∪ ð2 ∪ {((s3, ɛ),s1) ((s3, ɛ),s2)}, s3, A1 ∪ A2)   
If a is the regex bc and L(b) and L(c) are regular,   
M3 = L(M3)=L(a)=L(b)L(c) = (K1 ∪ K2, Σ, ð3= ð1 ∪ ð2 ∪ {((q, ɛ),s2): q ∈ A1}, s1, A2)  
Transitions between machines are ɛ

Regular Grammars  
Quadruple (V(rule alphabet of terminals and non-terminals), Σ(set of terminals – subset of V), R(finite set of rules in form X->Y), S(nonterminal start symbol))  
**grammartofsm**  
1. Create in M a separate state for each nonterminal in V.   
2. Start state is the state corresponding to S.   
3. If there are any rules in R of the form X -> a, for some a ∈ Σ, create a new state labeled #.   
4. For each rule of the form X -> a Y, add a transition from X to Y labeled a.   
5. For each rule of the form X -> a, add a transition from X to # labeled a.   
6. For each rule of the form X -> ɛ, mark state X as accepting.   
7. Mark state # as accepting.   
8. Complete M if incomplete (see the textbook for details).

Pumping theorem  
-split string into w = xyz, where |w|>=k, |xy| <= k and y =/= ɛ

Context-free grammar  
one non-terminal -> any number of terminals/non-terminals  
Self-embedding: S -> aSa  
Recursive: S -> T T -> S  
Backus-Naur Form (BNF): Non-terminal symbol surrounded by angle brackets (<program>, <variable>, etc), | means or  
NOTE: **L = { an1bn1an2bn2…ankbnk : k ≥ 0 and ∀*i* (ni ≥ 0)** is equivalent to **L = {anbn : n ≥ 0}\***

**removeunproductive(G:CFG)**Bottom-up, mark rules as productive if they end in a terminal, remove remaining rules as unproductive  
S -> AB (3rd pass - productive)  
S -> DE  
A -> a (1st pass - productive)  
B -> bC (2nd pass - productive)  
C -> c (1st pass - productive)  
D -> dF  
E -> e (1st pass - productive)  
F -> fD

S -> AB   
A -> a   
B -> bC   
C -> c   
E -> e

**removeunreachable(G:CFG)**Mark S as reachable, mark rules as reachable for each pass until no change, remove unreachable

S -> AB (start - reachable)  
A -> a (1st pass - reachable)  
B -> bC (1st pass - reachable)  
C -> c (2nd pass - reachable)  
E -> e

S -> AB   
A -> a   
B -> bC   
C -> c

Derivations and Parse trees

Chomsky hierarchy  
( Turing Machines ( ( PDAs ( FSMs | Regular Languages ) Context-free Languages ) D Languages ) SD Languages )

Difference between context-free, deterministic context-free and regular languages (and their grammar)

Machines: Pushdown Automata/Finite State Machine/Turing Machine

∪ ɛ Σ ð **Ø** ⊆ 𝒫 ∩ ⊂ ¬ ∈