Theory of Computation Cheat Sheet

Induction

Proof by induction has two steps: proving the base case and the inductive step. These two parts should convince us that $S(n)$ is true for every integer n that is equal to or greater than the basis.

E.g. Prove $S(n) = n! > 2^n$ for $n \geq 4$

Step 1: Basis
$S(4) = 4! > 2^4 = 24 > 16$.

Step 2: Induction
Inductive hypothesis: assume true for $k \geq 4$.
If $S(k) = k! > 2^k$ then $S(k+1) = (k+1)! > 2^{k+1}$.
Rewrite $S(k+1)$ so it can make use of $S(k)$.
$S(k+1) = (k+1)\,k! > 2 \cdot 2^k$
$S(n)$ tells us that $k! > 2^k$. If we remove that assumed truth from the above statement, we only need to show $k+1 > 2$. Since $n \geq 4$, we get $4+1 > 2$ which holds.

Technique of Diagonalization

Diagonalization is used to show that a set is uncountable. An uncountable set is an infinite set that contains too many elements to be countable (i.e. it is not enumerable). The uncountability of a set is closely related to its cardinal number: a set is uncountable if its cardinal number is larger than that of the set of all natural numbers. The set of natural numbers (and any other countably infinite set) has cardinality aleph-null ($\aleph_0$).

1: Assume set is countable, enumerate all subsets
2: Create new subset using elements from existing subsets
3: Show new subset is in set but not in enumeration

E.g. A function $f : N \to N$ is monotone-increasing if $f(i) < f(i+1)\ \forall\, i \in N$. Prove, using diagonalization, that the set of monotone-increasing functions is uncountable.

1) Assume set of all monotone-increasing functions is countable

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & \cdots & i \\ \hline f_1 & \mathbf{1} & 2 & 3 & \cdots & \\ f_2 & 2 & \mathbf{4} & 6 & \cdots & \\ f_3 & 3 & 5 & \mathbf{7} & \cdots & \\ \vdots & & & & & \\ f_i & & & & & \end{array}$$

2) Create a new monotone-increasing function $f(i) = f(i-1) + f_i(i),\ f \neq f_i\ \forall\, i$.
$f(1) = 0 + f_1(1) = 0 + 1 = 1$
$f(2) = 1 + f_2(2) = 1 + 4 = 5$
$f(3) = 5 + f_3(3) = 5 + 7 = 12$
$\vdots$

3) $f(i) \in N$ but not in enumeration (differs in some column with every row). $\therefore$ the set of monotone-increasing functions is uncountably infinite.

E.g. Prove, using diagonalization, that the power set of natural numbers (2^N) is uncountable.

1) Assume 2^N is countable. Enumerate all subsets (in binary representation)

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & \cdots & N \\ \hline S_1 & \mathbf{0} & 0 & 0 & \cdots & \\ S_2 & 1 & \mathbf{1} & 1 & \cdots & \\ S_3 & 1 & 0 & \mathbf{1} & \cdots & \\ \vdots & & & & & \\ S_i & & & & & \end{array}$$

2) Create a new subset $S(i) = 1 - S_i(i),\ S \neq S_i\ \forall\, i$.
$S(1) = 1 - S_1(1) = 1 - 0 = 1$
$S(2) = 1 - S_2(2) = 1 - 1 = 0$
$S(3) = 1 - S_3(3) = 1 - 1 = 0$
$\vdots$

3) $S(i) \in 2^N$ (because the power set contains all subsets of set N) but not in enumeration. $\therefore$ the power set of all natural numbers is uncountably infinite.

Set Theory

Sets are (1) well defined, (2) have no ordering of elements, and (3) contain no duplicates – but multi-sets do.

Definitions	A : set	A is a set
	$x \in A$	x is an element/member of A
	$\emptyset, \{\}$	empty set
	$\emptyset \subseteq A$	empty set is a subset
	$A \subseteq A$	set is a subset of itself
	$A \subseteq B$	A is a subset of B if $a \in A, a \in B$
	$A \subset B$	proper subset if $a \in A, a \in B, A \neq B$
	2^A	power set (all subsets of set A)
	$\|A\|$	cardinality of A (number of elements)
	$2^{\|A\|}$	number of subsets of A

Operations	Union	$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ e.g. $\{1,2\} \cup \{2,3\} = \{1,2,3\}$
	Intersection	$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$ e.g. $\{1,2\} \cap \{2,3\} = \{2\}$
	Set Difference	$A - B = \{x \mid, x \in A, x \notin B\}$ e.g. $\{1,2\} \setminus \{2,3\} = \{1\}$
	Cartesian Product	$A \times B = \{(a,b) \mid a \in A, b \in B\}$
	Complement	everything not in the set

Properties	Commutative	$A \cup B = B \cup A$ $A \cap B = B \cap A$
	Associative	$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$
	Distribution	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

E.g. Is the set of all functions $f : \{0,1\} \to N$ countable?

This function has two cases: $f_i(0) \to N$ and $f_i(1) \to N$. $|N| = \aleph_0$ which means the set of N is countable. $f : \{0,1\} \to N$ is equivalent to $f : \{0\} \to N \bigcup f : \{1\} \to N$. The union of two countable sets results in a countable set, $N \bigcup N = N$ thus $f : \{0,1\} \to N$ is countably infinite.

One-to-one and Onto Functions

• A onto $B \Rightarrow$ every element in B is mapped
• A 1-1 $B \Rightarrow$ every element in A has a unique mapping
• 1-1 correspondence $\Rightarrow$ bijective (1-1 *and* onto). *i.e.* No two values map to the same value. Every element in the codomain is mapped. $|A| = |B|$, same cardinality.
• Identity function $\Rightarrow$ input parameter is the same as the output value.
• A function has a single outcome for each parameter

E.g. functions $f : N \to N$ where $N = \{1, 2, 3, \dots\}$

1. f is 1-1 but not onto:
   $f(x) = x + 1$
   $f(1) = 2$
   $f(2) = 3$
   $f(3) = 4$
   $\vdots$

2. f is onto but not 1-1:
   $f(x) = \lceil \frac{x}{2} \rceil$
   $f(1) = 1$
   $f(2) = 1$
   $f(3) = 2$
   $f(4) = 2$
   $f(5) = 3$
   $f(6) = 3$
   $\vdots$

3. f is a 1-1 correspondence and not an identity function:
   $f(x) = x - (x + 2\, mod\, 3) + (x\, mod\, 3)$
   $f(1) = 2$
   $f(2) = 3$
   $f(3) = 1$
   $f(4) = 5$
   $f(5) = 6$
   $f(6) = 4$
   $\vdots$
   Conditional functions would also work.

Countability

Set A is countable if $\exists$ a 1-1 correspondence between A and N. i.e. Can systematically enumerate all elements in A eventually, labeling each with a unique natural number. You might have to find a creative pattern to list them 1-by-1.

E.g. Set of $\pm$ rational numbers is countable if you enumerate as $x_1, -x_1, x_2, -x_2, \cdots$

E.g. $N \times N \times N$ is countable. 3-tuple (i, j, k). If $a = 1$, there are a^3 tuples (finite) such that $1 \le i \le a$, $1 \le j \le a$, $1 \le k \le a$. Finite if $a = 2, 3, \cdots$

E.g. $|(0,1)| = |2^N| = \aleph_1 =$ uncountable. ($\aleph_0$ is countable)

E.g. Set of binary functions $f : N \to \{0,1\}$ is uncountable.

Languages

• An alphabet Σ is a finite set of symbols, e.g. $\Sigma = \{0,1\}$
• A language over alphabet Σ is a set of strings, each having its characters drawn from Σ
• Length of a string s, $|s|$, is the # of symbols in string s
• Empty string ϵ has length 0
• Substrings of $abc = a$, b, c, ab, bc, abc
• Prefix is any # of leading symbols, e.g. ϵ, a, ab, abc
• Postfix is any # of trailing symbols, e.g. ϵ, c, bc, abc
• Reversal is the string in reverse, e.g. $s = abc$, $s^R = cba$

Special languages:

1. $\emptyset$ is the empty language
2. $\{\epsilon\}$ language that contains empty string
3. Σ^* (Kleene Closure) contains all strings over Σ
4. Σ^+ (Positive Closure) contains all strings over Σ except the empty string

Operations

Concatenation	$\{a\} \cdot \{b\} = \{a\}\{b\} = \{ab\}$
Union	$\{a\} \cup$ or $+$ or $\mid$ $\{b\} = \{a, b\}$
Kleene Closure	$\{a\}^* = \{\epsilon,\ a,\ aa,\ aaa,\ \cdots\}$
Positive Closure	$\{a\}^+ = \{a,\ aa,\ aaa,\ \cdots\}$

E.g. First 5 strings of language $L = \{x \in \{a,b,c\}^* : x$ contains at least one a and at least one $b\}$ in lexicographical order. $\Rightarrow$ ab, ba, aab, aba, abb

E.g. Let $X = \{aa, bb\}$ and $Y = \{\epsilon, b, ab\}$.

1. List the strings in the set XY.
   $XY = X \cdot Y = \{aa, bb\}\{\epsilon, b, ab\}$
   $= \{aa,\ aab,\ aaab,\ bb,\ bbb,\ bbab\}$

2. List the strings of the set Y^* of length three or less.
   $Y^* = \{\epsilon, b, ab\}^*$ of length 3 or less
   $= \{\epsilon,\ b,\ bb,\ bbb,\ ab,\ bab,\ abb\}$

3. How many strings of length 6 are there in X^*?
   $X^* = \{aa, bb\}^*$ of exactly length 6
   Each symbol has length 2, so there have to be 3 symbols in each string. $2^3 = 8$ strings.
   (a) $000 \to aaaaaa$
   (b) $001 \to aaaabb$
   (c) $010 \to aabbaa$
   (d) $011 \to aabbbb$
   (e) $100 \to bbaaaa$

(f) $101 \rightarrow bbaabb$

(g) $110 \rightarrow bbbbaa$

(h) $111 \rightarrow bbbbbb$

E.g. Let $L_1 = \{aaa\}^*$, $L_2 = \{a,b\}\{a,b\}\{a,b\}\{a,b\}$, and $L_3 = L_2^*$. Describe the strings that are in the languages.

1. L_2 = strings of length 4 that are any combination of a's and b's.
2. L_3 = closure of L_2, 0 or more occurrences of L_2. i.e. empty string and strings that are a multiple of 4 with any combination of a's and b's. E.g. ϵ, $aaab$, $bbbbaaaa$, etc.
3. $L1 \bigcap L_3$ = intersection of L_1 and L_3, which are strings of symbol aaa that are multiples of 12. $L1 \bigcap L_3 = \{aaa\,aaa\,aaa\,aaa\}^*$

To show a language is regular: show one option exists

1. regular expression
2. DFA
3. NFA
4. NFA w/ ϵ-moves

If L is finite, it is regular.

To show a language is irregular:

1. Pumping lemma
2. Show DFA that would require infinite states
3. Use closure properties that relate to other nonregular languages

Closure Properties of regular languages:
Regular languages are closed under certain operations (i.e. if L_1, L_2 are regular, then so is resulting language).

Union	$L_1 \cup L_2 = r + s$
Concatenation	$L_1 \cdot L_2 = rs$
Closure	$L_1^* = r^*$, $L_2^+ = s^+$
Complement	$\bar{L} = \Sigma^* - L$, (L regular, so is $\bar{L}$)
Intersection	$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
Set difference	$L_1 - L_2 = L_1 \cap \overline{L_2}$

E.g. For any fixed n, is $\bigcup_{i=1}^{n} L_i$ regular, where each L_i is regular?

True. L_i is regular so has regex r_i. Finite languages, so $\bigcup_{i=1}^{n} L_i$ has regex $r_1 + r_2 + \cdots + r_n$.

E.g. Give examples of languages L_1 and L_2 over alphabet $\{a,b\}$ that satisfy[1]:

[1]Hints: Known regular languages: $\Sigma^*, \epsilon, a^*, a^*b^*$, etc. Known nonregular languages: a^nb^n or languages with similar dependencies.

1. L_1 is regular, L_2 is nonregular, and $L_1 \bigcup L_2$ is regular.
   $L_1 = \{a,b\}^*$, $L_2 = \{a^nb^n | n \geq 0\} \rightarrow L_1 \bigcup L_2 = a^*b^*$
   $L_1 = a^*$, $L_2 = \{a^i \mid i \text{ is prime}\} \rightarrow L_1 \bigcup L_2 = L_1$
2. L_1 is regular, L_2 is nonregular, and $L_1 \bigcup L_2$ is nonregular.
   $L_1 = \{a^*\}$, $L_2 = \{a^nb^n | n \geq 0\} \rightarrow L_1 \bigcup L_2$
   $L_1 = \{aa\}$, $L_2 = \{a^i \mid i \text{ is prime}\} \rightarrow L_1 \bigcup L_2 = L_2$
3. L_1 is regular, L_2 is nonregular, and $L_1 \bigcap L_2$ is regular.
   $L_1 = \{a^*\}$, $L_2 = \{a^nb^n | n \geq 0\} \rightarrow L_1 \bigcap L_2 = a^*$
   $L_1 = \{aa\}$, $L_2 = \{a^i \mid i \text{ is prime}\} \rightarrow L_1 \bigcap L_2 = L_1$
4. L_1 is nonregular, L_2 is nonregular, and $L_1 \bigcup L_2$ is regular.
   $L_1 = \{a^i | i > 0, i \text{ is prime}\}$, $L_2 = \{a^i | i > 0, i \text{ is not prime}\} \rightarrow L_1 \bigcup L_2 = a^+$
   $L_1 = \{a^i | i \text{ is prime}\}$, $L_2 = \{a^i | i \text{ is not prime}\} \rightarrow L_1 \bigcup L_2 = a^*$

E.g. Prove or disprove the following:

1. If L^* is regular, then L must be regular.

   False. For example, $L = \{a^{2^i} | i \geq 0\}$ is nonregular, but L^* is. Or $L = \{a^i | i \text{ is prime}\}$
2. For any language L, L^* must be regular (discuss both cases when L is finite and infinite).

   If L is finite, then it must be regular (you can find a FSM or regex). And if L is regular $\Rightarrow$ L^* is regular, because the Kleene star/closure is a regular operation under closure properties. Therefore, if L is finite, L^* must be regular.
   However, if L is infinite, then it is nonregular because you would need an infinite number of states (you cannot find a FSM or regex). Therefore, if L is infinite, L^* can be regular or nonregular (i.e. it does not say anything).

Regular Expressions

Regular expressions (regex) are sets in a nicer notation.

$01 = 0 \cdot 1 = \{01\}$	Concatenation: 0 followed by 1
$0 + 1 = \{0, 1\}$	Union: 0 or 1
$0^* = \{0\}^*$	Kleene closure: zero or more 0's
$0^+ = \{0\}^+$	Positive closure: one or more 0's
$0? = \{\epsilon, 0\}$	zero or one 0
$(0+1)^* = \{0,1\}^*$	all strings over $\{0,1\}$
$0^*10^*10^* = 0^* \cdot 1 \cdot 0^* \cdot 1 \cdot 0^*$	strings containing exactly two 1's
$(0+1)^*11 = \{0,1\}^* \cdot \{1\} \cdot \{1\}$	strings ending with 11
$0(1+10)^* + (1+10)^*$ $= \{0\} \cdot \{1,10\}^* \cup \{1,10\}^*$	strings of 0's and 1's not containing substring 00

Deterministic Finite Automata (DFA)

DFA & NFA are Finite State Machines (FSM). They can only scan input once (i.e. new string requires new scan process) and have finite memory. Unlike Turing machines.

DFA's are 5-tuple:	$M = (Q, \Sigma, \delta, q_0, F)$
Q	finite set of states
Σ	finite alphabet
$q_0 \in Q$	initial state
$F \subseteq Q$	final states
δ	transition function $Q \times \Sigma \to Q$
$\delta(q,a) = p$	where q, P are states in Q and a is symbol in Σ
$L(M)$	language accepted by machine M

• Deterministic means you can predict the path it will take (unlike in NFA).
• Every DFA is an NFA.
• Easy method to make a DFA that cannot contain a certain substring, is to make a DFA that has to accept that substring, and then complement the DFA (non-final states become final and vice-versa: $\bar{F} = Q - F$).

Nondeterministic Finite Automata (NFA)

NFA is a finite state machine where for each pair of state and input symbol there may be several possible next states. This distinguishes it from DFA, where the next possible state is uniquely determined. Although DFA and NFA have distinct definitions, they are equivalent, in that, for any given NFA, one may construct an equivalent DFA, and vice-versa (powerset construction). Both types of automata recognize only regular languages.

An NFA, similar to a DFA, consumes a string of input symbols. For each input symbol it transitions to a new state until all input symbols have been consumed. Unlike a DFA, it is non-deterministic in that, for any input symbol, its next state may be any one of several possible states. Thus, in definition, the next state is an element of the power set of states. This element, itself a set, represents some subset of all possible states to be considered at once.

An extension of NFA is NFA with ϵ-moves, which allows a transformation to a new state without consuming any input symbols (using ϵ as the symbol in the transition).

Accepting an input is similar to that for DFA. When the last input symbol is consumed, the NFA accepts if and only if there is some set of transitions that will take it to an accepting state (it's allowed to reach a stuck state). Equivalently, it rejects, if, no matter what transitions are applied, it would not end in an accepting state.

E.g. Give the state diagram of an NFA (without ϵ-moves) that accepts the language $(ab)^* + a^*$ and convert it to a DFA using the standard algorithm.

1. $(ab)^*+a^* = (a{\cdot}b)^* \mid a^* = 0$ or more ab's or 0 or more a's.

   For an NFA M, there exists a DFA M' such

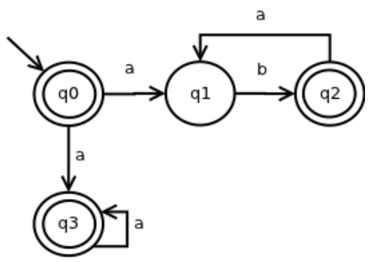

Figure 1: NFA without ϵ-moves for $(ab)^* + a^*$

that $L(M) = L(M')$.

$$\begin{aligned} M &= (Q, \Sigma, \delta, q_0, F) \\ Q &= \{q_0, q_1, q_2, q_3\} \\ \Sigma &= \{a, b\} \\ F &= \{q_0, q_2, q_3\} \end{aligned}$$

δ	a	b
q_0	$\{q_1, q_3\}$	$\emptyset$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_1\}$	$\emptyset$
q_3	$\{q_3\}$	$\emptyset$

Each state in M' corresponds to a subset of states from M.

$$\begin{aligned} M' &= (Q', \Sigma, \delta', q_0', F') \\ Q' &= 2^Q = \{\emptyset, [q_3], [q_2], [q_2, q_3], [q_1], [q_1, q_3], [q_1, q_2], \\ &\quad [q_1, q_2, q_3], [q_0], [q_0, q_3], [q_0, q_2], [q_0, q_2, q_3], [q_0, q_1], \\ &\quad [q_0, q_1, q_3], [q_0, q_1, q_2], [q_0, q_1, q_2, q_3]\} \\ F' &= \text{all elements in } 2^Q \text{ that contain a state in } F \\ &= \{[q_3], [q_2], [q_2, q_3], [q_1, q_3], [q_1, q_2], [q_1, q_2, q_3], [q_0], \\ &\quad [q_0, q_3], [q_0, q_2], [q_0, q_2, q_3], [q_0, q_1], [q_0, q_1, q_3], \\ &\quad [q_0, q_1, q_2], [q_0, q_1, q_2, q_3]\} \end{aligned}$$

Now the most important part, the transition function. To complete this part, we look at the NFA state transitions. For example, $\delta'([q_0], a) = [q_1, q_3]$ because $\delta(q_0, a) = \{q_1, q_3\}$.

$$\begin{aligned} \delta'([q_0], a) &= [q_1, q_3] \text{ ...new state!} \\ \delta'([q_0], b) &= \emptyset \\ \delta'([q_1, q_3], a) &= [q_3] \text{ ...new state!} \\ \delta'([q_1, q_3], b) &= [q_2] \text{ ...new state!} \\ \delta'([q_3], a) &= [q_3] \\ \delta'([q_3], b) &= \emptyset \\ \delta'([q_2], a) &= [q_1] \text{ ...new state!} \\ \delta'([q_2], b) &= \emptyset \\ \delta'([q_1], a) &= \emptyset \\ \delta'([q_1], b) &= [q_2] \end{aligned}$$

For an NFA with n states, the DFA could have up to 2^n states. In our case, the DFA will have 5 states (instead of 16).

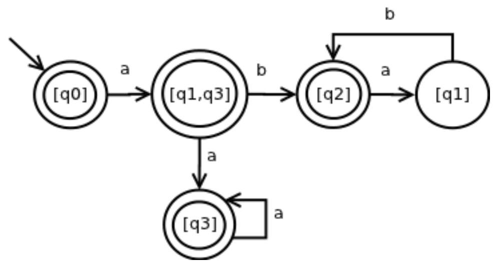

Figure 2: DFA converted from NFA

Pumping Lemma

E.g. Show that each of the following languages is not regular:

1. $\{a^i b^j | i > j\}$

   Proof by contradiction (using pumping lemma):
   Step 1: Assume L is regular, then let n be the constant in the lemma.

   Step 2: Select a specific string $z \in L$ such that $|z| \geq n$.
   $z = a^{n+1}b^n$

   Step 3: Split z into uvw.
   By the lemma, $|uv| \leq n$, thus v appears within the first n characters (v consists only of a's). $z = uvw = a^q a^r a^s b^n$ where $u = a^q, v = a^r$, and $a^s b^n = w$.
   From the lemma we know:
   $q + r + s = n + 1$
   $0 \leq |q| < n$
   $1 \leq |r| \leq n$ because $|v| \geq 1$
   $0 \leq |s|$
   $q + s < n + 1$

   Step 4: Find an i such that $uv^i w \notin L$, violating the necessary condition.
   The lemma states that for L to be regular, $uv^i w \in L$.
   If $i = 0$, then $uv^0 w = uw = a^q + a^s < n + 1 \notin L$ (because there must be more a's than b's, and if one a is missing, then it's false). Therefore $uw \notin L$ and L is nonregular.

2. The set of strings over $\{0, 1\}$ with an equal number of 0's and 1's.
   $L = \{w | w \in \{0, 1\}^*$ and # of 0's = # of 1's in $w\}$

   Intuition says we would need infinite states and thus a DFA would be impossible to build. Try pumping lemma.

   Select $z = 0^n 1^n \in L = uvw \in L$
   $|uv| \leq n$ so v consists of 0's. $u = 0^q$, $v = 0^r$, $w = 0^s 1^n$.
   We know $q + r + s = n$ and $n \geq |v| \geq 1$, so $q + s < n$.
   Lemma says $uv^i w \in L$ if regular. But for $i = 0$, we have too few 0's in our string, since $uv^0 w = uw = 0^q 0^s 1^n \notin L$ because $q + s < n$. Therefore L is not regular.

3. $\{a^i b^j c^{2j} | i \geq 0, j \geq 0\}$

   Select $z = a^n b^n c^{2n} \in L = uvw$.
   $|uv| \leq n$ so v consists of a's.
   $u = a^q, v = a^r, w = a^s b^n c^{2n}$.
   $q + r + s = n$
   Find an i such $uv^i w \notin L$.
   $i = n + 1 \rightarrow uv^{n+1}w = q + (n + 1)r + s + n + 2n = q + r + s + n + 2n + nr = n + n + 2n + nr$
   Which is not in L because then there would be more a's than c's. Thus L is not regular.

4. The set of nonpalindromes over $\{a, b\}$.

   If L is regular, then so is $\bar{L}$ (the set of palindromes).
   $\bar{L} = \{w \mid w \in \{a, b\}^*, w = w^R\}$. Use pumping lemma.
   Select $z = a^n b a^n = uvw$. $u = a^i, v = a^j, w = a^k b a^n$.
   $i + j + k = n$
   $i + k < n$
   $|uv| \leq n$
   $i + j \leq n$, so $n \geq j \geq 1$.
   Pump v 0 times ($i = 0$), we get $a^{i+k} b a^n \notin L$.
   Therefore $\bar{L}$ is not regular and thus neither is L.

Arbitrary

Prove false: Show a single example where it's false (proof by contradiction).

Cantor's Theorem: For any set A, $|A| < |2^A|$

Sufficient & Necessary Conditions: $P \Rightarrow Q$: If P, then Q (P iff Q). P is a sufficient condition for Q to be true. Q is a necessary condition for P to be true.
• If Q is false, we can say P is false.
• If Q is true, we cannot say anything about P.

• $\emptyset$ is a language, but not a string
• ϵ is a not a language, but it is a string
• every language is infinite or has an infinite complement
• some languages are infinite and have an infinite complement $L = \{w \in \{0, 1\} \mid |w|$ is odd$\}$
• empty set $\emptyset$ is a subset of every language
• kleene closure of a language is not always infinite, $\emptyset$ and $\{\epsilon\}$
• concatenation of infinite language and finite language is not always infinite

Context-Free Grammar (CFG)

CFG's are used for describing the structure of programming languages and other artificial languages. The idea is to use "variables" for sets of strings (i.e. languages). Variables are defined recursively. Every production rule is of the form $V \to w$, where V is a nonterminal symbol (variable) and w is a string on terminals and/or non-terminals (can be empty). TIP: using more variables makes it easier.

E.g. Find CFG's for the following languages:

$$\{a^nb^n | n \geq 1\}$$
$$S \to ab \,|\, aSb$$

$$\{a^nb^n | n \geq 0\}$$
$$S \to aSb \,|\, \epsilon$$

$$(a+b)^*a$$
$$S \to a \,|\, aS \,|\, bS$$

Set of all production rules for CFG's with $T = \Sigma = \{a, b\}$, and $V = \{A, B, C\}$

$$S \to X \underline{\to} Y$$
$$X \to A \,|\, B \,|\, C$$
$$Y \to \epsilon \,|\, aY \,|\, bY \,|\, AY \,|\, BY \,|\, CY$$

$$\{a^ib^jc^k | i \neq j\, or j \neq k\}$$

4 cases:

$i > j$, $j < i$, $j > k$, $j < k$

$$S \to ABC \,|\, DEF \,|\, GHI \,|\, JKL$$
$$A \to aA \,|\, a$$
$$B \to aBb \,|\, ab \,|\, \epsilon$$
$$C \to cC \,|\, \epsilon$$

Similar for $i < j, j > k, j < k$

$$\{w | w \in (a+b) \text{ and } a's \neq b's\}$$

3 cases:

$a = b$, $a > b$, $a < b$

$$S \to U \,|\, V$$
$$T \to aTbT \,|\, bTaT \,|\, \epsilon$$
$$U \to TaT \,|\, TaU$$
$$V \to TbT \,|\, TbV$$

Set of strings over $\Sigma = \{a, b, \cdot, +, (,)\}$

$$S \to a \,|\, b \,|\, S \cdot S \,|\, S + S \,|\, (S)$$