Nikolai Sazonov

Boolean Algebra

=	equals
≠	not equals
¬	not
v	or
⋀	and
≡	logically equivalent
→	implication
↔	explicit implication / biconditional

Predicate

∀	for all
∃	there exists
∈	is an element of
∉	is not an element of

Set Notation

∅	empty set
⊆	subset
⊂	proper subset
⊃	superset
⊇	proper (strict) superset
P(S)	powerset
∩	intersection
∪	union
∴	therefore

A **deterministic finite-state automaton (DFA)**

M = (Q, ∑, q, δ, F)

- Q = finite set of states
- ∑ = input alphabet
- q_0 ∈ Q = start state
- F ⊆ Q = final/accepting state
- ε = stop regex (empty string)
- δ = Q x ∑ → Q transition function
- δ(q_m, a) = $q_{m+1 (n)}$
- # = Blank Symbol

Jan 27, 2025

Boolean Algebra

Prove that **$p \rightarrow q \equiv \neg p v q$**

p	q	$p \rightarrow q$	$\neg p v q$	$p \leftrightarrow q$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	T
T	T	T	T	T

Prove that **$p \hat{} (p \rightarrow q) \equiv p \hat{} q$**

$p \hat{} (p \rightarrow q) \equiv p \hat{} (\neg p v q)$ see above
$\equiv (p \hat{} \neg p) v (p \hat{} q)$ Distributive laws
$\equiv F \; v \; (p \hat{} q)$ always false
$\equiv (p \hat{} q)$

Prove that **$((p v q) \hat{} \neg p) \rightarrow q \equiv T$**

$\equiv \neg((p v q) \hat{} \neg p) \; v \; q$ distributive
$\equiv \neg(p \hat{} \neg p) v (q \hat{} \neg p) \; v \; q$
$\equiv \neg(q \hat{} \neg p) \; v \; q$ de morgan transformation
$\equiv (\neg q v \neg\neg p) \; v \; q$
$\equiv (\neg q v p) \; v \; q$ associative
$\equiv T \; v \; p$ v p is redundant
$\equiv T$

P(x)
P - predicate "green"
P(x) applies P to x
P(x="grass") P(x="board in Gullic 100")
W() = "wins"
W(x="HWS", y="Hockey") = "HWS wins Hockey"

let P be a **one-place predicate** (one input: … is happy)
$\forall x(P(x))$ is a proposition, which is true iff P(a) is true for all entity a in the domain of discourse for P.
$\exists x(P(x))$ … … there is at least one entity a … for which P(a) is true
Proposition examples:
H(x) - x is happy
C(y) - y a computer
O(x, y) - x owns y

Jack owns a computer Can be translated into
$\exists x(O(jack, x) \hat{} C(x))$
there exists some x, where jack owns x, and x is a computer

Everyone who owns a computer is happy
$\forall x(\exists y(O(x, y) \hat{} C(y)) \rightarrow H(x))$

all persons who own something which is a computer, are happy

Everyone is unhappy
$\forall$ x(¬H(x))
all persons x are not happy

At least two people are happy
$\exists$ x $\exists$ y(H(x)^H(y)^(x $\neq$ y))
there exists persons x and y which x,y are happy, and x is not y.

Jan 29, 2025
Set Theory

A **set** is an **unordered** collection of **unique elements**
{ NY, London, Paris }
{ food, B, Car, kyle }
∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈ ∈
∈ - is an element of
NY ∈ of { NY, London, Paris }
∅ - empty set { }

Set notation with description { x | P(x) }
{ x | x is a city with Population > one million }
{ (a,b) | a>b ^ b>0 }
{ X | X is a set that has size 5 } *notice uppercase X

⊆ - subset of a set

Let A, B be sets, **A⊆ B** iff
$\forall x(x \in A \rightarrow x \in B)$
A is subset of B if and only if x, which belongs to A, implies x belonging to B
“lo Wo” ⊆ “Hello World!”

Let A, B be sets, **A=B** iff
$\forall x(x \in A \leftrightarrow x \in B)$

∅∈A *Nothing is a subset to every Set*

A∈B, B∈A iff A=B, A∈A A=A *Every Set is a subset to itself, and equal to itself*

If $A \in B$, $A \neq B$, then A is a **Proper Subset**

Intersection $\cap$ only variables seen in both A and B
$A \cap B = \{ x \mid x \in A \wedge x \in B \}$

union $\cup$ both A and B, without duplicate values
$A \cup B = \{ x \mid x \in A \vee x \in B \}$

complement $\bar{A}$ the Universe without A
$\bar{A} = \{ x \in U \mid x \notin A \}$
the $\bar{A}$ of $\bar{A}$ is A

difference A – B all of A, without any of B
$A - B = \{ x \mid x \in A \wedge x \notin B \}$

$|A|, |B|\ |A \cup B| \neq |A| + |B|$
$|A|, |B|\ |A \cup B| = |A| + |B| - |A| \cap |B|$
A ***union*** B is both values, without duplicate middle

$|A| \cap |B| = |A| + |B| - |A \cup B|$
A ***intersection*** B is both values, and then subtracting Union, to only keep the duplicates

Let
$A = \{ a,b,c \}$
Proper Subset of A is P(A), a set of all possible subsets of A
$P(A) = \{ \varnothing, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{a,c\}, \{a,b,c\} \}$

Prove **DeMorgan's Law**

let *c(A) = $\bar{A}$ because i can't write it on this keyboard*
let U be everything in the universe
$c(A \cup B) = c(A) \cap c(B)$
$c(A \cup B) = \{ x \in U \mid x \notin (A \cup B) \}$
$= \{ x \in U \mid \neg(x \in (A \cup B) \}$
$= \{ x \in U \mid \neg(x \in A \cup x \in B) \}$
$= \{ x \in U \mid x \notin A \wedge x \notin B \}$
$= \{ x \in U \mid x \in c(A) \wedge x \in c(B) \}$

$A \cap c(B \cup c(A))$
$= A \cap c(B) \cap c(c(A))$ separate compliments
$= A \cap c(B) \cap A$

= A ∩ c(B)

Jan 31, 2025

Set and Set Notation

Cross Product **A x B**
A x B = { (a,b) | a∈A ˆ b∈B }

C = { boiled, fried, baked }
F = { egg, steak, cake }

CxF = { (boiled egg), (boiled steak), (boiled cake), (fried egg), (fried steak), (fried cake), (baked egg), (baked egg), (baked cake) } These are all *tuples*

<u>Function</u> y = xˆ2 + 2x
A function from A to B (f = A->B) is a subset (⊆) of A x B (a cross b) which has the property that for each a∈A, C contains one and only one ordered pair whose first coordinate is a.

a function is **onto** *iff*
$\forall b \in B(\exists a \in A(b = f(a))$
B is range, A is domain
for every value in the range, there is an a within the domain, where we can produce the value of f(a)   (it doesn't say how many a, you can have two)   y = 2x is onto

one-to-one   "for every $b \in B$, there is only one $a \in A$"
$\forall x \in A\ \forall y \in A\ (x \neq y \to f(x) \neq f(y))$

Proof: Deduction

If you believe some <u>premises</u> are true, then logic forces you to accept that the conclusion is true
let premises = $p \rightarrow q$ and p
let $\mathbf{p \rightarrow q}$ = "If today is tuesday, then im John Cena"
let **p** = "Today is tuesday"   <- <u>Premises</u>
Therefore in conclusion ($\therefore$), im John Cena

premises: if 2+3 = 5, then 3+2 = 5
                 and 2+3 = 5
Conclusion: $\therefore$ 3+2 = 5

let P and Q be any proposition (grass is green) or predicate logic (exists for all) formulas,
P => Q is used to mean $P \rightarrow Q$ is tautology. In all cases, if P is true, then Q is also true.
P being premises, and Q being Conclusion

Feb 3, 2025
Proof of logic

Proof of logic
let people A, B, C, D
A will be at the party id B is there and C is not there
B will be there if the party is on Fri or Sat
If C is there, D will be there
D cant be there if party is on Fri
The Party is on Fri

A will be at the party *trying to prove*

in logic terms
$(b \hat{} \neg c) \rightarrow a$ $\therefore$ a
$(FvS) \rightarrow b$ $\therefore$ b
$c \rightarrow d$ $\therefore$ $\neg c$
$F \rightarrow \neg d$ $\therefore$ $\neg d$
F $\therefore$ $\neg d$
Use the logic terms to get therefore's and complete the proof

Feb 5, 2025
Biconditional

Biconditional
$(p\rightarrow r)v(p\rightarrow r) \equiv (p\hat{}q)\rightarrow r$
R $\leftrightarrow$ S

p	q	r	p→r	q→r	pˆq	(p→r)v(q→r)	(pˆq)→r	R↔S
F	F	F	T	T	F	T	T	T
F	F	T	T	T	F	T	T	T
F	T	F	T	F	F	T	T	T
F	T	T	T	T	F	T	T	T
T	F	F	F	T	F	T	T	T
T	F	T	T	T	F	T	T	T
T	T	F	F	F	T	F	F	T
T	T	T	T	T	T	T	T	T

Feb 12, 2025

Infinity Hotel thought experiment, by David Hilbert.

When the hotel is full, the person in the first room has to move to the second, and the person in the second room then has to go to the third, and ect..   In this thought experiment, the last person will not be taken care of, however in infinity, there is no last person, so everyone gets a room.

$\infty + 1 = \infty$ $1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 \ldots = \infty$

You can group infinity differently for a **different solution**,

$1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 \ldots \infty = 0$

$1 + (-1 + 1) + (-1 + 1) + (-1 + 1) + (-1 + 1) \ldots \infty = 1$!!!!?

$2 * \infty > \infty$

one-to-one (every x has only 1 y)

onto (one to one, and all y have an x) (same *cardinality*)

Natural Numbers: 0, 1, 2, 3, 4, ∞ -> **countable**

ˆ? *the cardinality of integers Has to be greater than of natural numbers, there is 2∞, 2∞>∞*

Integers: -∞, -2, -1, 0, 1, 2, ∞ -> **not *onto***

Feb 17, 2025

FORMAL LANGUAGE THEORY

An **alphabet** is a finite, non-empty set
whose elements are called **symbols**

A finite sequence of symbols $a_1, a_2, ..., a_n$
from an alphabet is called a **String** over that alphabet

Strings have operations:

.Length |x|

$|\varepsilon| = 0$

$|\lambda| = 0$

Concatenation

$x = a_1, a_2 .., a_n$

$x = a_1, a_2 .., a_n, b_1, b_2 .., b_n$

Reversal

$x^R = a_n, ..., a_2, a_1$

Set of all strings over an alphabet Σ is denoted **Σ***

$\Sigma = \{ 0, 1 \}$, $\Sigma^* = \{ \lambda, 0, 1, 10, 11, 110, 111, ... \}$

A **language** over an alphabet ∑ is a subset of ∑*

$\sum = \{ 0, 1 \}$ <- Alphabet ∑

$L_1 = \{ 00, 01, 10, 11 \}$ <- Language of all 2-digit binary numbers

$L_2 = \{ x \in \sum^* \mid n_0(x) = n_1(x) \}$

$n_0(x)$ = number of 0s in String x

$L_3 = \{ x \mid x \% 5 = 0 \}$

A **deterministic finite-state automaton (DFA)**

$M = (Q, \sum, q, \delta, F)$

Q = finite set of states

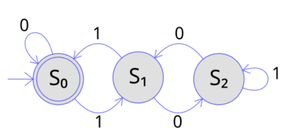
∑ = input alphabet

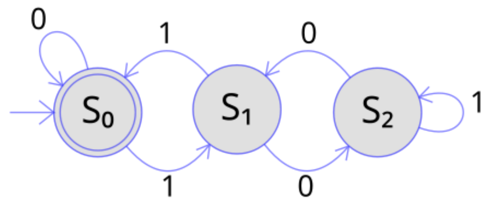
$q_0 \in Q$ = start state

$F \subseteq Q$ = final/accepting state

$\delta = Q \times \sum \rightarrow Q$ transition function

$\delta(q_m, a) = q_{m+1(n)}$

M = 



Formally

$Q = \{ q_0, q_1, q_2 \}$

$\sum = \{ 0, 1\}$

$q_0 = q_0$

$F = \{ q_1, q_2 \}$

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_1$

$\delta(q_1, 1) = q_2$

Figure 1

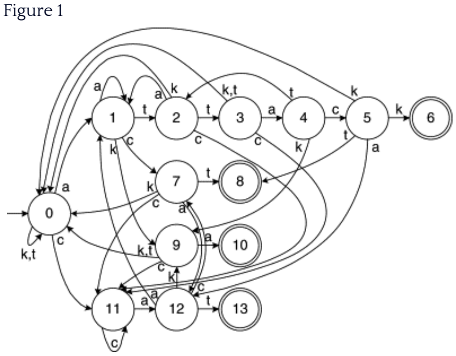
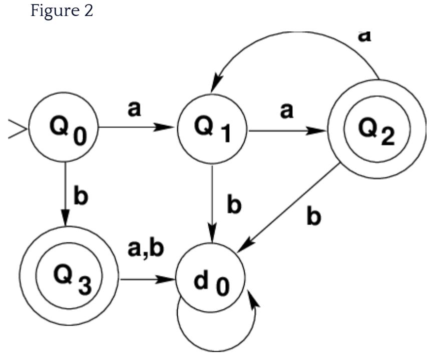


Figure 2



For **figure 2**

L = { two consecutive a's or three consecutive b's }

d_0 = trap

The DFS java program can be used to get to the states. available on canvas.

Feb 19, 2025

Regular Expression

Let ∑ be an alphabet

The following patterns are Regular Expressions over ∑

1: Φ and E are regex

2: a ∈ ∑

3: if r_1 and r_2 are regular expressions, then so are $r_1 \mid r_2(r_1 + r_2)$, $r_1r_2(r_1r_2)$, r_1^*, r_1^+, (r_1)

Language generated (matched) by a regex r (L(r)) is defined:

1: L (Φ) = ∅ , no string matches Φ

2: L (ε) = { ε }

3: L (a) = { a }

4: $L(r_1 \mid r_2) = L(r_1 + r_2) = L(r_1) \cup L(r_2)$

5: $L(r_1 r_2) = L(r_1) * L(r_2)$

6: $L(r_1^*)$ matches sequences of r_1 0 or more times

7: $L(r_1^+)$ ....1..............

8: $L((r_1))$ matches r_1

A language is regular if its generated by a regular expression

∑ = { a, b }   using only characters from alphabet ∑

$L(a^* \mid b^*)$ = { string of all a's or all b's and ε }

$L(ab^*)$ = { x | x starts with a, ends with only 0 or more b's }	ex:	a, ab, abb, abbb
$L((ab)^*)$ = { x | ab repeating }	ex:	ab, abab, ababab
$L((a \mid b)^*)$ = { everything (within ∑)}	ex:	abaa, ababbaa, bbb

∑ = { 0, 1 }

$1(0 \mid 1)^* 1$ = starts and ends with 1, contains as many 0's or 1's as you want

$(0 \mid 1)^* 00$ = contains as many 0's or 1's as you want, ends with 00

Feb 21, 2025

Regular Expression Grammar

Grammar contains 3 componentes:
1: Terminal Symbol, $\in \{ \sum \} \cup \{ \varepsilon \}$
2 : Non-terminal Symbol
3: Production Rules
4: Starting Production Rule

Example Grammar: (not regular grammar)
$\sum = \{ a, b \}$
$S \rightarrow aBa \mid aa$
$B \rightarrow bC \mid bb$
$C \rightarrow a$

aa	$S \rightarrow aBa \mid aa$
aBa	$S \rightarrow aBa \mid aa$
abba	$B \rightarrow bC \mid bb$
abaa	$C \rightarrow a$

Regular Grammar can **only** have the following 3 form of production rules
1: $A \rightarrow x$
2: $A \rightarrow xB$ (right linear)
3: $A \rightarrow Bx$ (left linear)
$A \rightarrow B$

$L((ab)^*)$ * ab abab ababab
$S \rightarrow \varepsilon \mid aB$
$B \rightarrow bS$
Starting at S, either stop, or continue, printing a and going to B printing b and goes back to S
It can be formally drawn as a visual graph, L () function, or Grammatical Expression

Feb 26, 2025

$(0^* 1 0^+ 0^*)^*$

Feb 28, 2025

Nondeterministic Finite-State Automata (NFA)

$\sum = \{ a, b \}$

NFA accepts a string iff (if and only if)

1. All symbols are consumed by NFA
2. Accepting states in a subset of all possible final states

L = { x | x ends with 01 or 10 }

λ - means go somewhere? somehow?

Mar 12, 2025

$L - M$ Strings in L but not in M ?

Reverse regular expression

Give E

if E is a symbol a,b, ε, $\varnothing$, then $E^R = E$

if E is $r_1 + r_2$, $E^R = r_1^R + r_2^R$

r_1, r_2 $E^R = r_2^R r_1^R$

r_1^*, $E^R = (r_1^R)^*$

L^R is also regular ?

lowkey idk what i'm even writing

Homomorphism

a function that maps a string for each symbol in the alphabet $\sum = \{ 0, 1 \}$

f(0) = ab f(1) = a

010101 = aba aba aba

ab being f(0), which prints 0, and a being f(1), which prints 1

apply f to each symbol in e ab(ab+a)*ab

Inverse Homomorphism (we do that later)

Mar 26, 2025

A context free grammar is a tuple (V, $\sum$, P, S) where

1, V is a set of symbols. this the non-terminal symbols

2, $\sum$ is a set of terminal symbols $\sum \cap V = \varnothing$

3, P is a set of production rules in the form A -< w, where

$A \in V$, $w \in (V \cup \Sigma)^*$

4, $S \in V$, as the state symbol

Example of regular grammar:

A -> x

A -> xB

B -> yA

vs contect free grammar:

S -> aSb | ε

S -> SS   S -> aS

Derivation Tree for S -> aSb | ε

	S	
a	S	b
a	S	b
	ε	

L = $a^n b^n$ <-   Context free grammar

A language produced by a context free gramma is a context free language

L = { palindrome } = { ww^R }
abb bba
S -> aSa | bSb | a | b | ε

Mar 28, 2025

L = { $n_a(a) = n_b(b)$ } $a^n b^n$ abababab   abba
S -> aSb | bSa | SS | ε
SS breaks down the string into as many pieces as you want

L = { palindrome } ∑ = { a, b }
S -> aSa | bSb | ε

Mar 31, 2025

Push Down Automata

Push Down Automata (PDA)
A push down automata M is a six-tuple   **M = (Q, ∑, Γ, q_0, δ, F)**
M = (Q, ∑, Γ, q_0, δ, F)

1. Q finite set of states
2. ∑ input alphabet (language alphabet)
3. **Γ stack alphabet**
4. q_0 starting state
5. δ transition functions
6. F set of final states

-> q_0 -> (a, ε | Δε)   then you can put things onto a stack

δ $[(q_i, a \in (\sum \cup \varepsilon), r \in (\sum \cup \Gamma)), (q_j, \boldsymbol{w} \in (\sum \cup \Gamma)^*)]$
if at state q_i, next input symbol is **a** and the pop Γ has symbol r

if (state q_i)
&& next input $a \in (\sum \cup \varepsilon)$
&& Γ pop $r \in (\sum \cup \Gamma)$

then transit to state q_j && push $\boldsymbol{w} \in (\sum \cup \Gamma)^*$

Transition Functions:
[$(q_i, a, r), (q_j, \varepsilon)$] -> pop **r** from Γ, replace with ε*
[$(q_i, a, r), (q_j, sr)$] -> push **s** onto stack Γ

[$(q_i, a, r), (q_j, s)$] -> replace **r** with **s** this involves **two steps**
1) [$(q_i, a, r), (q_j, \varepsilon)$] consume a to get nothing
2) [$(q_{i2}, \varepsilon, \varepsilon), (q_{j2}, s)$] check nothing and place s

Example of Push Down Automata for $a^n b^n$
L = $a^n b^n$ (btw this expression is not regular)
∑ = { a, b }
Γ = { ƶ, a, b } ƶ is the bottom of the stack
[$(q_i, a, r), (q_j, w)$]
q_i a r w q_j
ε, ε | ƶ a, ƶ | aƶ b, a | ε ε, ƶ | ƶ
-> q_0 ------> q_1 --------> q_2 ------> q_3 ------> q_4 <- Terminate
q_0 and q_4 are accepting ↺ a, a|aa ↺ b, a|ε

Apr 7, 2025

bA -> ...
aA -> ... context based language

How can you **convert context free grammar** to **push-down automata**

S -> aSa | bSb | a | b | ε L = { ww^R } palindrome of even or odd length

Example:	*abba*	Example2:	abaa
S	*aSa*	*S*	*aSa*
aSa	*bSb*	*aSa*	*???*
abSba	*ε*	*broken* (has no *ab*)	
abba	*Final*		

The example is a good instance of **nondeterminism** that it will always make the right choice you always pick the right choice with nondeterminism, and at the end you are left with only NONTERMINAL Symbols (such as big S) .

1. If a terminal symbol is on top of the stack, and it matches the input - pop it.
2. If a nonterminal symbol is on top of the stack, - choose the correct production rule, and put the rule on the stack (a production rules is aSa | bSb)
   (2 is nondeterminism, really doing all the work theoretically)

3. Accept when both string and stack is empty.
   $\rightarrow q_x \xrightarrow{\varepsilon, z / z} (q_T)$
   ↺ b, b / ε (b is without a terminal Symbol, so push nothing onto the stack)
   a, a / ε (a is without a terminal Symbol, so push nothing onto the stack)
   ε, S / aSa (S could be replaced by aSa on the stack)
   ε, S / bSb (S could be replaced by bSb)
   ε , z / Sz (to start off the expression, push S onto stack)

can cannot use pumping lemma to prove that a language is regular, only if not regular

PUMPING LEMMA for CFL (context free language)

(∈ - element of) (∃ there exists) (∀ for all)

If a language L is context free, then

∃ integer p >= 1 (pumping constant, pumping length)

such that ∀ w in L, |w| >= p

w can be written as

w = uxyzv such that

1. |xz| >= 1
2. |xyz| <= p
3. ∀ i >=0, $ux^2yz^2v \in L$

S -> aSa | bSb | z

abaSaba
 ^   ^^ ^   pumping x and z could result to
 x   y   z   a<u>bbbbbb</u>baSa<u>bbbbbb</u>ba

To Look at some more Examples:

L = { $a^nb^nc^n$ }

1. Assume L is context-free
2. let p = k (k is some generic number where we have no control)
3. let *s* = $a^kb^kc^k$
4. According to pumping lemma
   s = uvwxy
   a<u>aaa</u> bbbb cccc
    ^ ^ ^
    v w x

APPROACH A)

- let u = ε, v = a^m, w = a^n, x = a^o, y = b^kc^k
- let k = m + n + o
- let i = 2 or any int
- $uv^iwx^iy = a^{2m+n+2o}b^kc^k$
- $a^{2m+n+2o}b^kc^k \neq a^kb^kc^k \quad \notin L$

B)

- let $u = a^m$, $v = a^n$, $w = a^o$, $x = a^p b^q$, $y = b^{k-q} c^k$
- $m + n + o = k$
- let $i = 2$ or any int
- $uv^2wx^2y = a^{m + 2n + o + 2p} b^{q + k - q} c^k$

C)

- let $u = a^m$, $v = a^n$, $w = a^o$, $x = b^p$, $y = b^{k-p} c^k$
- let $k = m + n + o$
- let $i = 2$ or any int
- $uv^2wx^2y = a^{k+n} b^{k+p} c^k$

D) when v, x are only b, same as Approach A
E) when v is only b, x has b and c, same as Approach B
the rest of cases are the same as A)B)C)

Apr 9, 2025

More Pumping Lemma for Push-Down Automata

L = { $a^n c^m b^n$ | n > m >= 0 }

let $p = k$
let $S = a^{k+1} c^k b^{k+1}$
let $S = uvxyz$

Case 1: either *v* or *y* contains an 'a'
since $|vxy| \le k$
v | y must contain a string, that mean *vxy* must start with 'a' otherwise neither *v* | *y* have an 'a'
v has to start within a^{k+1} Neither *v* nor *y* will contain 'b'
uv^2xy^2z will have more 'a' than 'b'
(since *v* & *y* dont have 'b' only 'a', however 'a' & 'b' must be the same length!!)

Case 2: both *v* and *y* only consist of 'c'
$uv^?xy^?z$ will have more 'c' than 'a' and 'b'

Case 3: Either *v* or *y* contains a 'b'
Same as Case 1, where n='b' > n='a' Count(b) > Count(a)

~~~~~~~~~~~~~~~~~~~~~~~~

L = { $n_a(w) = n_b(w)$ && $n_a(w) > n_c(w)$ }

$S = a^{k+1} c^k b^{k+1}$ Using the knowledge we have already

L = { $a^n b^m c^k$ && k = 2n }

This is context-free because
m is just * it has no req.
n & k are only called once to make k = 2n,
making this context free. If it is called more than once, it would not be C.F.
~~~~~~~~~~~~~~~~~~~~~~~~

L = { *ww* } ex: abb abb (just repeats again) (ww^r == palindrome)

Prove Context Free

let p = k

S = $a^k b^k a^k b^k$ < - selected string to pump ex: aaabbbaaabbb

S = $uvxyz$

Case 1: either v or y has an 'a'

$|vxy| \le k$, vxy occupy only one

$a^k b^k$ uv^2xy^2z will <u>not</u> have the same first half vs second half

Case 2: either v or y has a 'b'

same as Case(1)

Case 3: v or y only has 'a'

uv^2xy^2z have more 'a' than 'b'

Case 4: v or y only has 'b'

uv^2xy^2z have more 'b' than 'a'

Apr 14, 2025

Turing Machine

A T.M. is a 4-tuple (Q, ∑, q_0, δ)

1. Q is a finite set of states, including the halt state h
2. ∑ is alphabet which in cludes the black symbol # (□)
3. q_0 is the start state
4. $\delta\ (Q - \{h\}) \times \sum \rightarrow \sum \times Q \times \{L, R\}$

δ (q, σ) = (τ, d, r) means

1. starting state q
2. σ is the current symbol on the ***<u>tape</u>***
3. τ is the symbol replacing σ
4. d is the direction to move next on the ***<u>tape</u>***
5. r is the next state

$q_0 \xrightarrow{(a, b, R)} q_1$ a = Convert Symbol b = Replace Symbol R = move to the right

Example: L = { w | w starts with and ends with 'a' }

$$\mathbf{q_T} \xleftarrow{(b,\, \varepsilon\, R)} \mathbf{q_0} \xrightarrow{(a,\, \varepsilon,\, R)} \mathbf{q_1} \xrightarrow{(\#,\, \varepsilon,\, L)} \mathbf{q_2} \xrightarrow{(a,\, \varepsilon,\, R)} \mathbf{h}$$

$\circlearrowleft$(b, ε, R) | (a, ε, R)

Terminates if at q_0 (starting state) is 'b'

Keeps returning to q_1 for any letter

If its nothing # (the end of the string) move tape to the left (the end of string),
then checks if the last character is an 'a'

Terminate q_T is not necessary, because it only returns if it reaches the halt state.

Example: $L = \{ a^n b^n \ \&\& \ n > 0 \}$

$$q_0 \xrightarrow{(a\,c,\,R)} q_1 \xrightarrow{(b,\,d,\,L)} q_2 \xrightarrow{(c,\,c,\,R)} \textit{ret}\ q_0 \rightarrow q_0 \xrightarrow{(d,\,d,\,L)} q_3 \xrightarrow{(c,\,c,\,R)} q_4 \xrightarrow{(\#,\,\#,\,R)} h$$

q_1: ↺ (a, a, R) | (d, d, R) q_2: ↺ (a, a, L) | (d, d, L) q_3: ↺ (d, d, R)

Converts every single 'a' to 'c' and every 'b' to 'c' one by one
Cannot start with 'b'
if more 'a' than 'b' - > get stuck in q_1
if more 'b' than 'a' && anything else left -> get stuck in q_3
if not strictly only 'a' and then only 'b' it will get stuck and not accept

Lets Visualize:
a a b b a
c a d b a #   backtracking
c c d b a
c c d d a # q_3
Terminated   stuck in q_3 because no (a, ,) in q_3

April 25, 2025

Halting problem -> undecidable

Assume 5 state(T) can decide whether T has 5 states in it
new_5_state(T):
    if 5_state(T) == true, return false
    else return true
    Run new_5_state(new_5_state);

Proof
Assume Halt(T, n) exists such that halt can decide whether input n on T will halt
new_halt(T):
    if halt(T, T)== true, loop forever
    else halt
    Run new_halt(new_halt);

April 28, 2025

Turing Acceptable   L = recursive enumerable
Turing Decradagle   L = recursive
new_halt(TM)
if Halt(TM, TM): loop forever;
else halt;

Reduction
New problem reduced to halt -> use halt to implement new problem
halt reduced to new problem -> use new problem to implement halt
(new problem reduced from halt)