The language of a DFA

$M = (Q, \Sigma, q_0, F, \delta)$

states alphabet start state final state transitions

The language accept a set of string:

$\hat{\delta}(q_0, x) \in F$ This means a transition $\hat{\delta}$ accepts a string $x \in \Sigma^*$ if $q_0 \xrightarrow{x} q$ where $q \in F$.

Start state — a string — Final state

$\hat{\delta}$ is a function $\hat{\delta}: Q \times \Sigma^* \to Q$

☆ Define $\hat{\delta}$

$\hat{\delta}(q, x) = \delta(q, x)$ when $x \in \Sigma$ 当x不是string

$\hat{\delta}(q, \underset{\text{string}}{ax}) = \delta(\hat{\delta}(q, a), x)$ where $a \in \Sigma^*$ $x \in \Sigma$

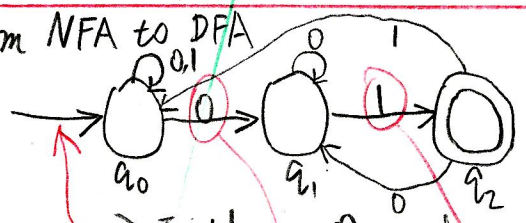
The language recognized by M is the language

$L(M) = \{ x \in \Sigma^* \mid M \text{ accepts } x \}$ over the alphabet Σ.

Another definition of $\hat{\delta}$ key notation $\cup$

$$\hat{\delta}(q, ax) = \bigcup_{q' \in \hat{\delta}(q, a)} \delta(q', x)$$

From NFA to DFA



Ending with 0, 1 NFA

State \ Inputs	0	1
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

DFA

Given an NFA, A language of A denoted of language in the set. $\{a \mid a \in \Sigma^* \text{ and } \_\_\_\_\_\}$

$\hat{\delta}(q_0, x) \in F$

Define $\hat{\delta}$:

$\hat{\delta}: \underbrace{Q \times \Sigma^*}_{\text{The input can be } \phi} \to \underbrace{P(Q)}_{\text{The result is a set of states.}}$ this means $q \xrightarrow{a}$ $\left.\begin{matrix} \xrightarrow{x} \\ \xrightarrow{x} \\ \xrightarrow{x} \\ \xrightarrow{x} \end{matrix}\right\}$ do their Union

How many states in $P(Q)$? $2^{|Q|}$

e.g. $Q = \{a, b, c\}$ $2^{|Q|} = 2^3 = 8$ $P(Q) = \{\phi, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$

Review for CSE 355 – DFA, NFA, εNFA

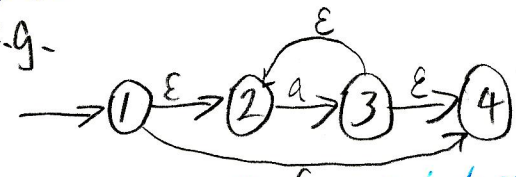
εNFA$(\Sigma, Q, q_0, \delta, F)$ DFA also could be represented by $M=(Q, \Sigma, \delta, q_0, F)$

Σ is the alphabet of input symbols. The set of all string accepted by a DFA / NFA / εNFA M is denoted by $L(M)$. We also say that the language $L(M)$ is accepted by M.

---

e-closure of a state is the set of all states, including S itself, that you can get to via e-transitions.

e.g.



e-closure(1) $=\{1, 2, 4\}$ include itself

e-closure(3) $=\{3, 2, 4\}$

How to define the εNFA

εNFA$(\Sigma, Q, q_0, \delta, F)$ we want to define δ

$\because$ ε and Σ can by accepted by the εNFA $\Rightarrow \delta: Q \times \Sigma \cup \{\varepsilon\} \to P(Q)$ NFA $\to P(Q)$ DFA $\to Q$

input a given sequence of symbols may get different results.

How to define DFA:

$\delta: Q \times \Sigma \to Q$

How to define NFA:

$\delta: Q \times \Sigma \to P(Q)$

$P(Q)$ means Power(Q)

e.g. $S=\{x, y, z\}$

$P(S)=\{\{\}, \{x\}, \{y\}, \{z\}, \{x,y\}, \{x,z\}, \{y,z\}, \{x,y,z\}\}$

We know that $\delta: Q \times \Sigma \to Q$ we want to define $\hat{\delta}: Q \times \Sigma^* \to Q$ $(\hat{\delta}(q, x))$

$\hat{\delta}(q, \varepsilon) = q$ for $|x|=0$

$\hat{\delta}(q, ay) = \hat{\delta}(\delta(q, a), y)$

Prove: $\hat{\delta}: Q \times \Sigma^* \to Q$

Basis: $\hat{\delta}(q, \varepsilon) = q$ for $|x|=0$

Induction: suppose $x = ay$ (y is a string, a is a symbol)

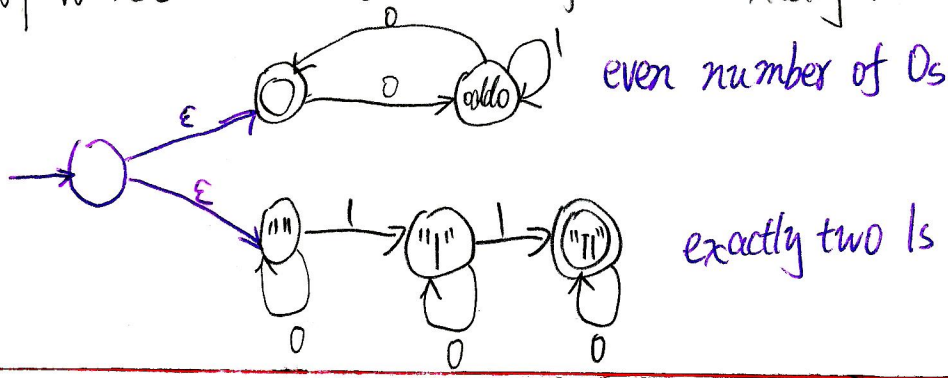
$$\hat{\delta}(q, ay) = \hat{\delta}(\delta(q, a), y)$$

Notice that if $x=a$, we have

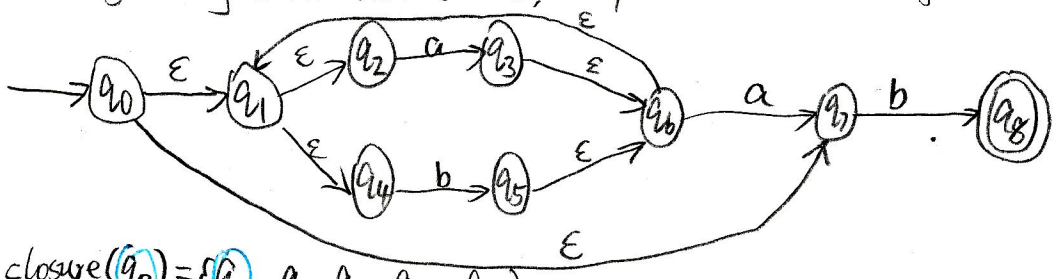
$\hat{\delta}(q, a) = \delta(q, a)$ since $a = a\varepsilon$ and $\hat{\delta}(\delta(q, a), \varepsilon) = \delta(q, a)$

CSE355 NFA

{w | w has an even number of 0s or exactly two 1s} with six states.



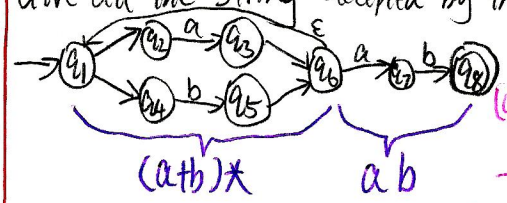
Consider the following ε-NFA, Σ={a,b}. Compute the ε-closure for each state.



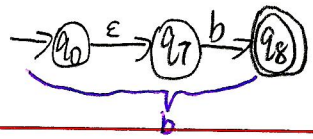
ε-closure(q_0) = {q_0, q_1, q_2, q_4, q_7}
ε-closure(q_1) = {q_1, q_2, q_4}
ε-closure(q_2) = {q_2}
ε-closure(q_3) = {q_1, q_2, q_4, q_3, q_6}
ε-closure(q_4) = {q_4}
ε-closure(q_5) = {q_1, q_2, q_4, q_5, q_6}
ε-closure(q_6) = {q_1, q_2, q_4, q_6}
ε-closure(q_7) = {q_7}
ε-closure(q_8) = {q_8}

e-closure of a state is the set of all states, include s itself, that you can get via ε-transitions.

Give all the string accepted by the ε-NFA with four character or fewer



b → 1 possible
(a+b)(a+b)*ab (length=4 ⇒ 2^2=4 possibles; length=3 ⇒ 2 possible)
Total = 7 possibles



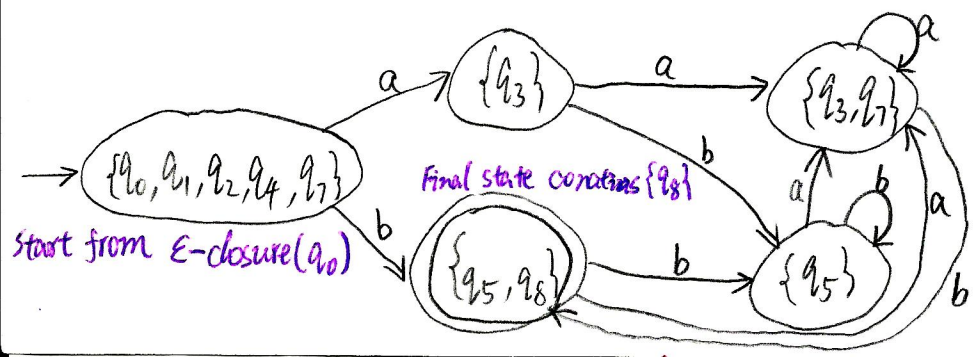
∴ b+(a+b)*ab
b, aab, bab, baab, bbab, aaab, abab

Convert the ε-NFA to a DFA

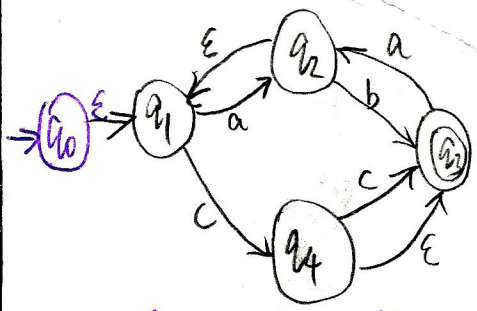
ε-closure(q_0) = {q_0, q_1, q_2, q_4, q_7} first step ε-closure(q_0)

	a	b
{q_0, q_1, q_2, q_4, q_7}	{q_3}	{q_5, q_8}
{q_3}	{q_3, q_7}	{q_5}
{q_5, q_8}	{q_3, q_7}	{q_5}
{q_3, q_7}	{q_3, q_7}	{q_5, q_8}
{q_5}	{q_3, q_7}	{q_5}

(all possible symbols)



Review CSE355 NFA to DFA

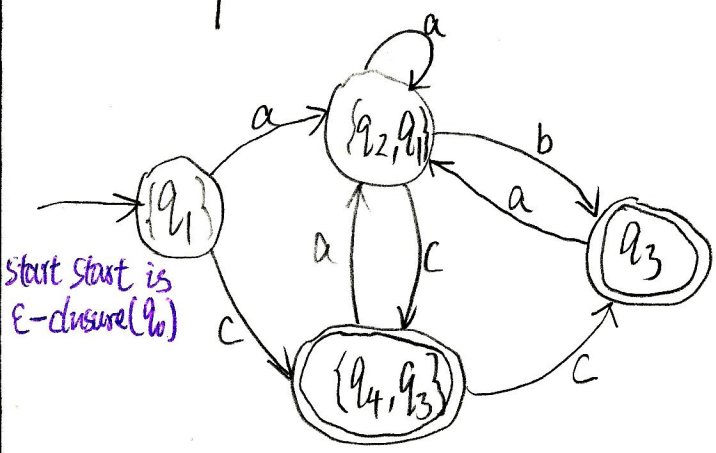


adding q_0 to change NFA to ε-NFA

ε-closure(q_0) = $\{q_1\}$

	a	b	c	all possible Σ
$\{q_1\}$	$\{q_2, q_1\}$	—	$\{q_4, q_3\}$	
$\{q_2, q_1\}$	$\{q_2, q_1\}$	$\{q_3\}$	$\{q_4, q_3\}$	
$\{q_4, q_3\}$	$\{q_2, q_1\}$	—	$\{q_3\}$	
$\{q_3\}$	$\{q_2, q_1\}$	—		

key: e.g. (q_4, a) means all possible states after input "a", so not include $\{q_3\}$



start start is ε-closure(q_0)

Final states are all states which contains $\{q_3\}$

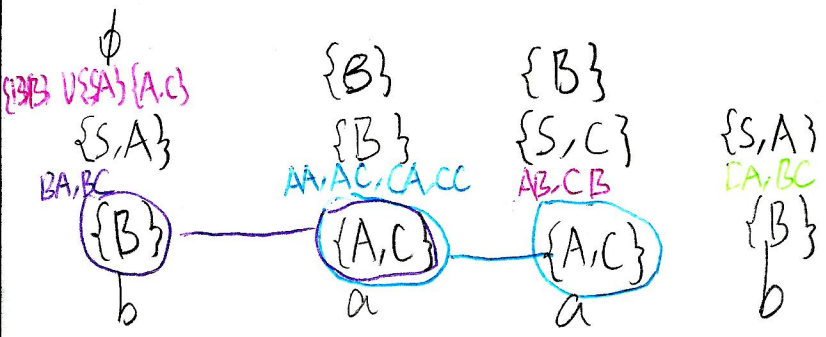
Review For CSE 355 Test 3

CYK Algorithm

CNF grammar G

$S \to AB \mid BC$
$A \to BA \mid a$
$B \to CC \mid b$
$C \to AB \mid a$

w is baaba Is baaba in $L(G)$?



$(X_{2,i}, X_{i+1,j})(X_{2,i+1}, X_{i+2,j})$

$\{A,C\}$ A,C can $\to a$
a B can $\to b$

$(X_{2,i}, X_{i+1,j})$

Review for CSE 355 — DFA, NFA — ε NFA

We know $q_0 \in Q$, $\delta: Q \times \Sigma \to Q$ and $F \subseteq Q$. — $\delta \to Q$ not $P(Q) \Rightarrow$ don't need $\cup$ (union)

From $\Sigma \to \Sigma^*$

We need a DFA, language of A, denoted by L_A is $\{x : x \in \Sigma^* \ \& \ \hat{\delta}(q_0, x) \in F\}$

$\hat{\delta}: Q \times \Sigma^* = Q$ — Answer

$\hat{\delta}(q, \varepsilon) = q$ — basis, suppose it is true

$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, \boxed{x}), \boxed{a})$ — $x \in \Sigma^*$, $a \in \Sigma$ — We want to put sth to make LHS = RHS

$\hat{\delta}(q, a) = \delta(q, a)$

We assumed $\hat{\delta}(q, \varepsilon) = q$ is true, and according to $\delta: Q \times \Sigma \to Q \Rightarrow \delta(q, \varepsilon) = q$

So, $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ — Let $xa = a \Rightarrow x = \varepsilon$

$\Downarrow$

$\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \varepsilon), a) \Rightarrow \hat{\delta}(q, a) = \delta(q, a)$

We know $q_0 \in Q$, $\delta: Q \times \Sigma \to P(Q)$ and $F \subseteq Q$ — $\delta \to P(Q) \Rightarrow$ need $\cup$

We need a NDFA, the language of A, denoted by L_A is $\{x : x \in \Sigma^* \ \& \ \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$ — just means we can reach F

$\hat{\delta}: Q \times \Sigma^* = P(Q)$

$\hat{\delta}(q, \varepsilon) = q$

$\hat{\delta}(q, xa) = \bigcup \delta(q', \boxed{a})$ — We want to put sth to make LHS = RHS

$\boxed{} \to q' \in ?$

$\Downarrow$

$\hat{\delta}(q, \varepsilon a)$

$q' \in \hat{\delta}(q, \boxed{x})$ — we only know $\hat{\delta}(q, \varepsilon) = q$

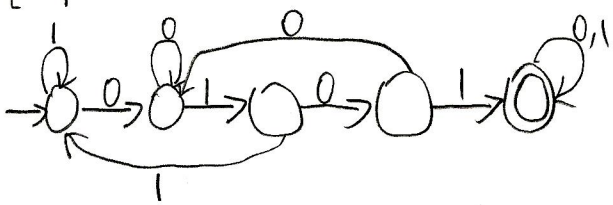
So, Let $xa = a \Rightarrow \boxed{x = \varepsilon}$

Finally, we get: $\hat{\delta}(q, a) = \bigcup_{q' \in q} \delta(q', a) = \delta(q, a)$

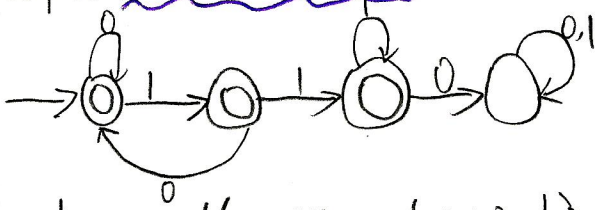
Review for CSE 355 DFA, NFA, ε NFA

Give the state diagrams of DFAs for the following languages.

{w | w contains the substring 0101}

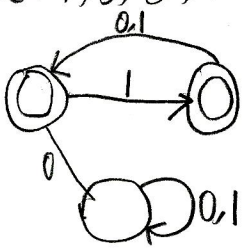


{w | w does not contains the substring 110} means all states are final, except which accepts "110"



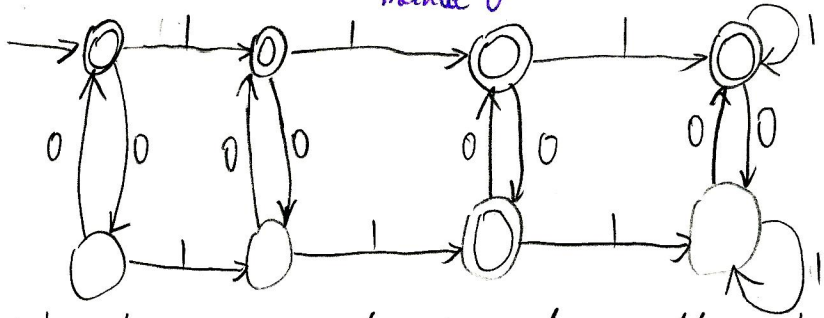
{w | every odd position of w is 1} consider ε

odd: 1, 3, 5, 7

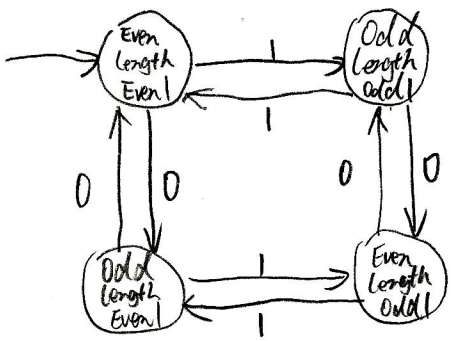


{w | w has an even number of 0's or exactly two 1s}

include "0"



{w | w has an even length and an odd number of 1s}

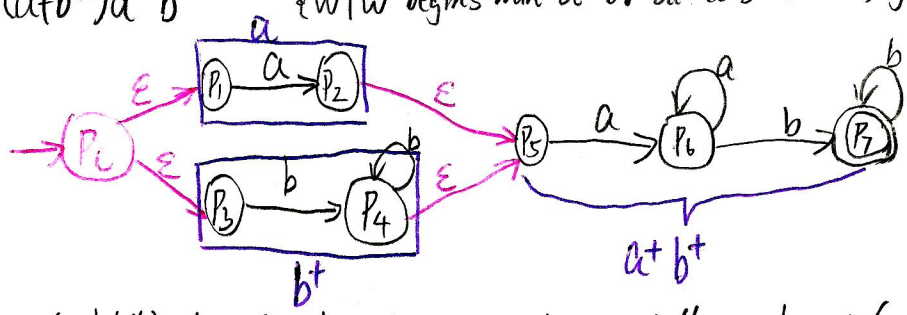


CSE 355 Regular expressions

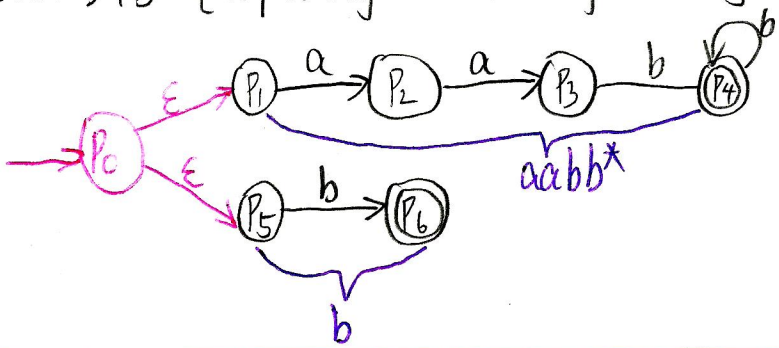
Regular expression to ϵNFA

have at least one or more

$(a+b^+)a^+b^+$ {w | w begins with a or at least one b, follows at least one a, follows at least one b}



$a(abb^*)+b$ {w | w begins with aa follows by at least one b, or contains exactly one b}



Write regular expressions for the following languages.

{w | w does not end with 11} Consider ϵ

$0^*+(0+1)^*00+(0+1)^*01+(0+1)^*10$

all 0 or ϵ — all possible conditions without 11

{w | w has an even number of 0s or exactly two 1s}

could have no 0

$(1+01^*0)^*+0^*10^*10^*$

even num of 0s — exactly two 1s

0 is a even num

{w | every odd position of w is 1} odd: 1,3,5,7.... "1" is in w.

$1(0+11)^*+(10+11)^*$ or $(1(0+1))^*(1+\epsilon)$

"1"

{w | w begins with 10, ends with 01, and contains the substring 00} $|10|+|01|+|00|=6$ need to define all conditions under length 6.

$10(1+0)^*00(1+0)^*01+100(0+1)^*01+10(0+1)^*001+1001$

conditions which length less than 6

{w | w either begins or ends (or both) with 01}

$01(0+1)^*+(0+1)^*01$

CSE355 Pumping lemma

Let L be a regular language, Then there exist a number n, such that for all string $|w| \geq n$, w can be written as xyz, such that

(1) $xy^iz \in L$ for $i \geq 0$

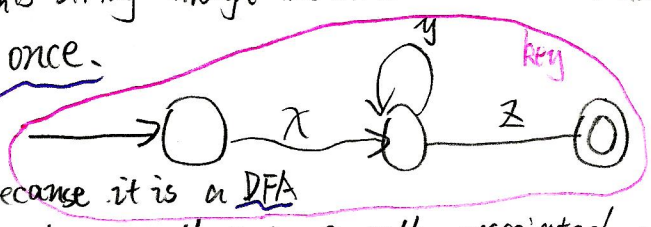
(2) $|xy| \leq n$

(3) $|y| \geq 1$

Proof:

Since L in a Regular Language, it must have a DFA, Let the number of states in that DFA be n, now consider any string w in L such that $|w| \geq n$.

Lets pass this string through the DFA, since the DFA has n states, this string must visit same state more than once.



$v \neq \varepsilon$ because it is a DFA

xz in L because there is a path associated with uw from initial state to a final state

uv^nw in L due to the same reason as above.

Use Pumping Lemma.

$L = \{0^n 1^m 0^n \mid m, n > 0\}$

Suppose L is $\boxed{\text{regular}}$, then it must satisfy the pumping lemma, Let $\boxed{p}$ be the number in the pumping lemma.

Now consider $www = 0^p 1^p 0^p$, We can break www to xyz such that $\boxed{|xy| \leq p}$, y thus contain only 0's, Let it be 0^k, $k > 0$ Suppose www is $\underbrace{\boxed{000}}_{x}\underbrace{\boxed{111}}_{y}\underbrace{\boxed{000}}_{z}$ ☆ $p=3$ ☆ $\Rightarrow xy^2z = \underbrace{0000}_{4\text{ zeros}}111\underbrace{000}_{3\text{ zeros}} \Rightarrow$ not in L

We have $xy^2z = 0^{p+k}1^p0^p$. Since $k > 0$, xy^2z is not in L.

L does not satisfy the pumping lemma. Hence, L cannot be a regular language.

$L = \{www \mid \text{where } w \in \{0,1\}^*\}$

Suppose L is regular. then it must satisfy the pumping lemma, Let p be the number in the pumping lemma.

Now, consider $www = \overbrace{\boxed{0^p1^p}}^{w}\overbrace{\boxed{0^p1^p}}^{w}\overbrace{\boxed{0^p1^p}}^{w}$. We can break www to xyz such that $|xy| \leq p$, y thus contain only 0's. Let it be 0^k, $k > 0$.

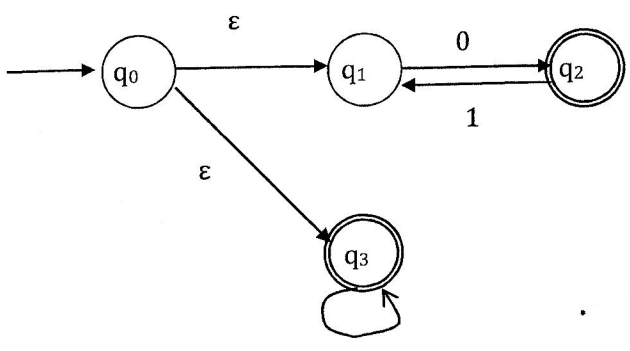
We have $xy^2z = 0^{p+k}1^p0^p1^p0^p1^p$ since $k > 0$, xy^2z is not in L.

L does not satisfy the pumping lemma. Hence, L cannot be a regular language.

CSE 355 Quiz 8 - 2/5/2013

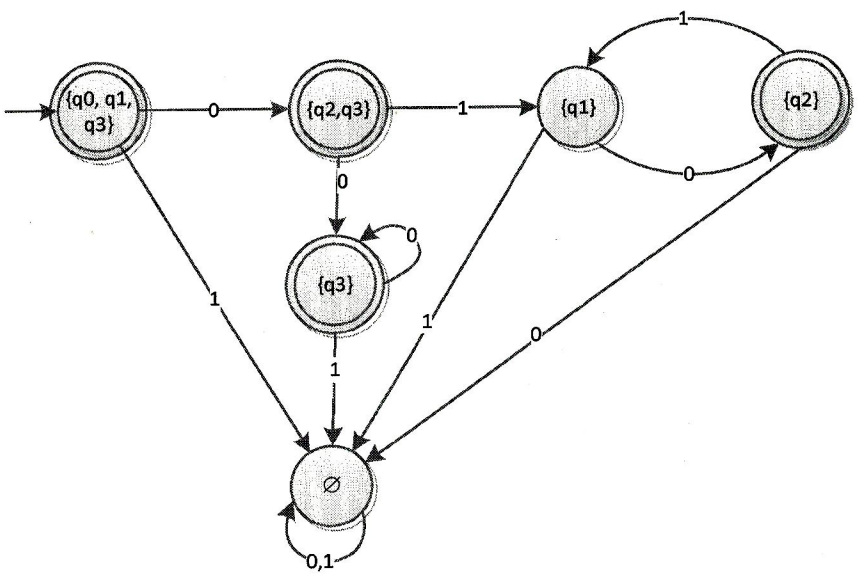
SOLUTION

Q1: Consider the following ε-NFA.
Convert it to its equivalent DFA by drawing its diagram and specifying a transition table for its δ.



$\varepsilon - CLOSE(q0) = \{q0, q1, q3\}$

	0	1
{q0, q1,q3}	{q2,q3}	∅
{q2,q3}	{q3}	{q1}
{q3}	{q3}	∅
{q1}	{q2}	∅
{q2}	∅	{q1}



Q2: Write a regular expression for each of the followings:

a). The set of strings over {a,b,c} in which all the a's precede the b's, which in turn precede the c's. It is possible that there are no a's, b's, or c's.

a*b*c*

b). The set of strings of length two or more over {a,b} in which all the a's precede the b's.

a*aab*+ a*abb*+a*b*bb

c). The set of strings over {a,b,c} that begin with a, contain exactly two b's, and end with cc.

a(a+c)*b(a+c)*b(a+c)*cc

Review for Test 1

Given the ECLOSE for each of the states of the following ε-NFA. Then convert the ε-NFA to a DFA.

S_1 can go to S_2, S_3 without any input

	ε	a	b	c
→ S_1	$\{S_2, S_3\}$	ϕ	$\{S_2\}$	$\{S_3\}$
S_2	ϕ	$\{S_1\}$	$\{S_3\}$	$\{S_1, S_2\}$
* S_3	ϕ	ϕ	ϕ	ϕ

$\cup$

ε-close$(S_1) = \{S_1, S_2, S_3\}$

ε-close$(S_2) = \{S_2\}$

ε-close$(S_3) = \{S_3\}$

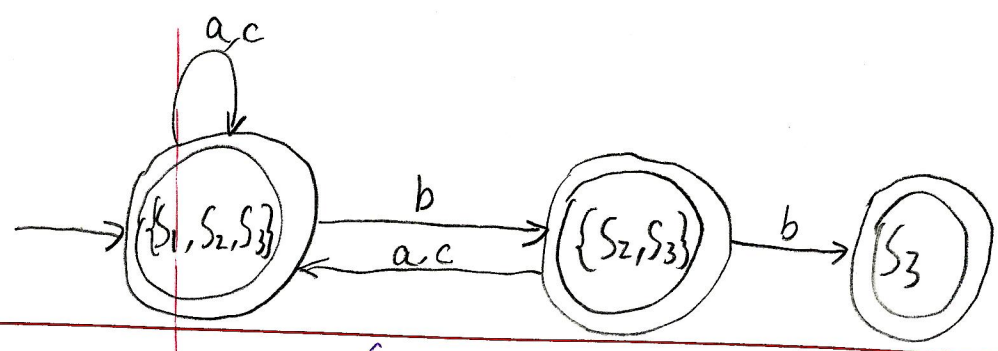
e-close always contains itself

$\delta(S_1, a) \Rightarrow \delta(S_1, \varepsilon) \cup \delta(S, a) = \{S_2, S_3\}$

$\delta(S_2, a) \cup \delta(S, \varepsilon)$

	a	b	c
$\{S_1, S_2, S_3\}$	$\{S_1, S_2, S_3\}$	$\{S_2, S_3\}$	$\{S_1, S_2, S_3\}$
$\{S_2, S_3\}$	$\{S_1, S_2, S_3\}$	$\{S_3\}$	$\{S_1, S_2, S_3\}$
$\{S_3\}$	—	—	—

$\delta(S_2, a) = S_1 \cup \delta(S_1, \varepsilon) = \{S_2, S_3\}$ // S_1 can go to $\{S_2, S_3\}$ without any inputs



Key:

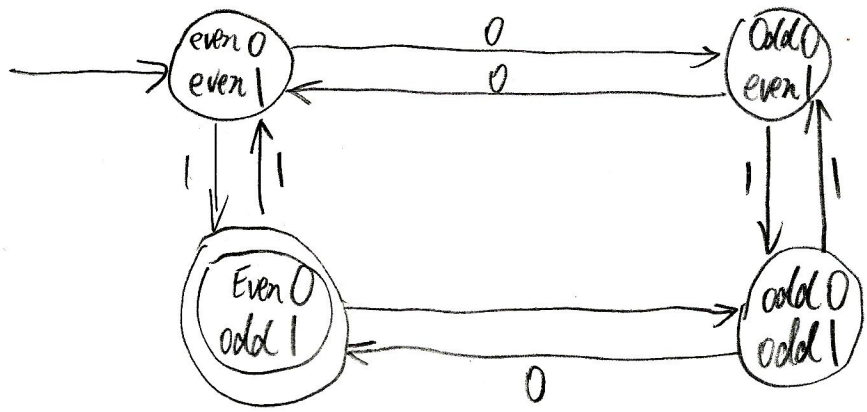
	ε
→ S_1	$\{S_2, S_3\}$

means, if we meet S_1 in transition table, then $\{S_2, S_3\}$ need to be add into the item.

question 4

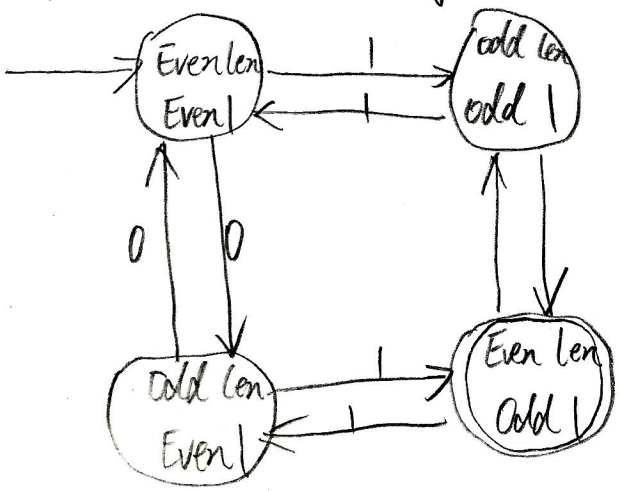
Review for Test 2

(5) Write a DFA for the following language: The language consisting of strings of 0s and 1s with even number (zero is considered an even number) of 0s and odd number of 1s.



Related question:

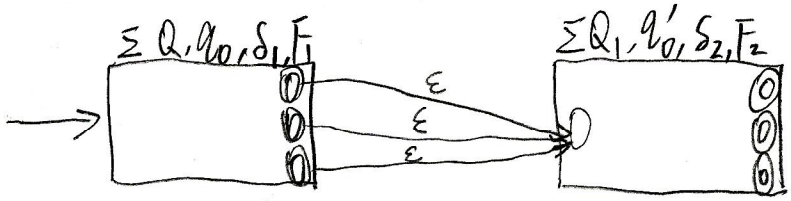
{w | w has an even length and an odd number of 1s}



Question 5

Review for Test 1

Let L_1 and L_2 be two regular languages having the same alphabet that have corresponding DFAs $(\Sigma, Q_1, q_0, \delta_1, F_1)$ and $(\Sigma, Q_2, q_0', \delta_2, F_2)$ respectively. Assume that Q_1 and Q_2 are disjoint. Using these two DFAs construct an ε-NFA (or just NFA) corresponding to the language $L_1.L_2$, where $L_1.L_2$ is defined as the set $\{xy \mid x \in L_1 \,\&\, y \in L_2\}$



$\Sigma \rightarrow$ same as before

$Q = Q_1 \cup Q_2$

$q_0 = q_0$

$\delta(x, \varepsilon) = q_0'$ where $x \in F_1$

$\delta(x, a) = \delta_1(x, a)$ where $x \in Q_1$ $a \in \Sigma$

$\delta(x, a) = \delta_2(x, a)$ where $x \in Q_2$ $a \in \Sigma$

Quiz 11 Question 11

Review for CSE 355

Prove: the number of edges in a tree of size n is $n-1$.

The defination of Tree:

If a graph $G(V,E)$ has any two of the following three properties, it is a tree.

1. G is connected
2. G has no cycles.
3. $|E| = |V| - 1$

We assume 1 and 2 are true, since we want to prove 3.

For $P(1)$:

When vertex $= 1 \Rightarrow$ Edge $= 0$ and vertex $= 2 \Rightarrow$ Edge $= 1$

So, adding one more vertex will result a tree withe one more Edge if vertex > 1.

For $P(k) \rightarrow P(k+1)$:

Assume vertex $= n \Rightarrow$ Edge $= n-1$ is true

We need to show that vertex $= n+1 \Rightarrow$ Edge $= n$

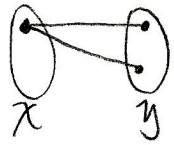
So, vertex $= n+1$ means adding one more vertex to a tree with vertex $= n$ which has $n-1$ edges.

Finally, vertex $= n+1 \Rightarrow$ Edge $= n-1+1 = n$

Eventually: $P(k+1)$ is true, under the assumption that $P(k)$ is true, so, vertex $= n \Rightarrow$ Edge $= n-1$

CSE 355

Function:



not a function

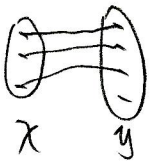
Total Function:

nothing left at left hand ⇒ Total

E.g. Prove $f(n)=n^2+4$ is total.

proof: Let $n \in N$. N is closed under multiplication and addition, so $f(n)=n^2+4 \in N$, Therefore f is Total.

1-1:



Every argument(x) is related to distinct value(y).

if $f(x)$ is monotone increasing ⇒ one to one function.

E.g. (a) $f(x)=x-1$

$\frac{df}{dx}f(x)=1>0$ ⇒ $f(x)$ is increasing ⇒ $f(x)$ is 1 to 1.

(b) $f(x)=x^2+1$

$\frac{d}{dx}f(x)=2x$ ⇒ $f(x)$ is not always increasing on Z ⇒ $f(x)$ is not one to one.

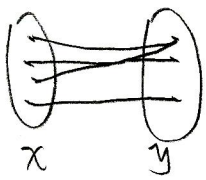
(c) $f(x)=x^3$

$\frac{d}{dx}f(x)=3x^2 \geq 0$ ⇒ $f(x)$ is increasing ⇒ 1 to 1.

(d) $f(x)=\lceil x/2 \rceil$

not 1 to 1, since $f(2.1)=\lceil 2.1/2 \rceil=2=f(2.2)$

On to



Every element of co-domin set is the image of some element in the domain set "A".

E.g. $f(x)=x^5+1$

Suppose $y \in R$, $f(x)=y=x^5+1$ ⇒ $x=\sqrt[5]{y-1}$ ⇒ The y still $\in R$ ⇒ onto

$f(x)=x^2+2x$

Suppose $y \in R$, $f(x)=y=x^2+2x$ ⇒ $x=\sqrt{y+1}-1$ ⇒ whe $y=-2$, x is not $\in R$, ⇒ -2 is not in the range of the function ⇒ not onto. 有偶次方一般不是Onto

---

$Card(X_1) \leq Card(X_2)$

if there exist a total 1-1 Function from X_1 to X_2.

$Card(X_1)=Card(X_2)$

if there is a total, 1-1 and onto function from X_1 to X_2.

To prove $Card(X_1)<Card(X_2)$ than we have to prove

$Card(X_1) \leq Card(X_2)$ and $Card(X_1) \neq Card(X)$ Total 1-1 but not onto

Review for CSE 355

上与 $\wedge$ 下或 $\vee$

De Morgan's Law

$\neg(p \wedge q) \equiv \neg p \vee \neg q$

$\neg(p \vee q) \equiv \neg p \wedge \neg q$

Equivalent

$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

Distributive property

$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

Sets:

If two sets are equal, we need to prove that both $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$

E.g. Prove $R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$

We need to show that $R \cup (S \cap T) \subseteq (R \cup S) \cap (R \cup T)$

and $(R \cup S) \cap (R \cup T) \subseteq R \cup (S \cap T)$

For $R \cup (S \cap T) \subseteq (R \cup S) \cap (R \cup T)$

Let $x \in R \cup (S \cap T)$

$\Rightarrow x \in R$ or $x \in (S \cap T)$

$\Rightarrow x \in R$ or ($x \in S$ and $x \in T$)

$\Rightarrow$ ($x \in R$ or $x \in S$) and ($x \in R$ or $x \in T$) // $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

$\Rightarrow x \in (R \cup S) \cap (R \cup T)$

For $(R \cup S) \cap (R \cup T) \subseteq R \cup (S \cap T)$

Let $x \in (R \cup S) \cap (R \cup T)$

$\Rightarrow$ ($x \in R$ or $x \in S$) and ($x \in R$ or $x \in T$)

$\because A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

$\therefore \Rightarrow x \in R$ or ($x \in S$ and $x \in T$)

$\Rightarrow x \in R \cup (R \cap T)$

Finally, we could say that $R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$

Prove $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

For $P(1)$:

$1 = \frac{1(1+1)}{2} = 1$

For $P(k) \to P(k+1)$:

Assume: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ is true

We need to show that $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$

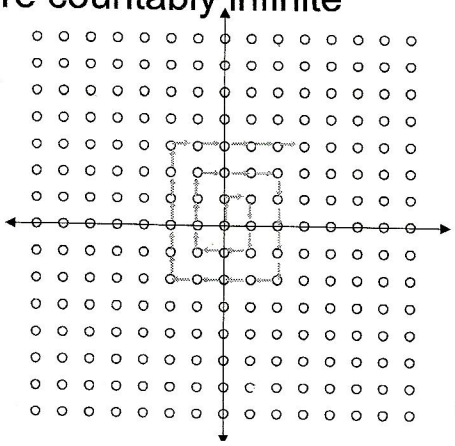
left hand side $= \frac{n(n+1)}{2} + (n+1)$

$= \frac{n(n+1) + 2(n+1)}{2} = \frac{n^2 + n + 2n + 2}{2} = \frac{n^2 + 3n + 2}{2} = \frac{(n+1)(n+2)}{2}$

Finally, $P(k+1)$ is true, under the assumption that $P(k)$ is true, so, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Showing ordered pairs of integers are countably infinite

A one-to-one correspondence



Showing a set is countably infinite

- Done by showing there is a one-to-one correspondence between the set and the integers
- Examples
  - Even numbers
    - Shown two slides ago
  - Rational numbers
  - Ordered pairs of integers
    - Shown next slide

Show that the rational numbers are countably infinite

- First, let's show the positive rationals are countable
- See diagram:
- Can easily add 0 (add one column to the left)
- Can add negative rationals as well

1/1	2/1	3/1	4/1	5/1	..
1/2	2/2	3/2	4/2	5/2	..
1/3	2/3	3/3	4/3	5/3	..
1/4	2/4	3/4	4/4	5/4	..
1/5	2/5	3/5	4/5	5/5	..
⋮	⋮	⋮	⋮	⋮	⋱

infinite Sets.

$N = \{0, 1, 2, \cdots\}$

$Even = \{0, 2, 4, 6 \cdots\}$

$Odd = \{1, 3, 5, 7 \cdots\}$

$Even_8 \{8, 10, 12 \cdots\}$

$N \times N = \{(0,0), (0,1), (0,2) \cdots$
$(1,0), (1,1), (1,2) \cdots$
$\vdots$

$R_{1,0}$ = Ration num between 0 and 1. $= \{0.000\ldots1 \cdots 1\}$

$N \to N \times N \Rightarrow$ Onto

(0,0) (0,1) (0,2) (0,3) (0,4) ...
(1,0) (1,1) (1,2) (1,3 (1,4) ...

$Card(R) > Card(N)$

$Card(N) = Card(Even) = Card(Odd) = Card(N \times N)$

Prove that all natural numbers $n \geq 8$ can be write as a sum of $3n$ and $5n$.

P(1): $8 = 3 + 5$: hence 8 can be written as a sum of $3n$ and $5n$.

For $P(k) \to P(k+1)$:

Assume it is true for all $n \geq 8$ and $n \leq i$ 设上限

Consider $i+1$

$i+1$ can be written as $(i-2)+3$

But $i-2$ can be written as a sum of $3n$ and $5n$.

Hence, $i+1 = (i-2)+3$ can be written as $3n$ and $5n$.

Prove that $\sqrt{2}$ is an irrational num.

proof: Suppose $\sqrt{2}$ is an rational num. Then by defination of rational num. $\sqrt{2} = \frac{p}{q}$ where p and q are integers with no common divition. — ①

$2 = \frac{p^2}{q^2} \Rightarrow p^2 = 2q^2 \Rightarrow p^2$ is an Even num. $\Rightarrow$ P is an Even num $\Rightarrow$ P can be written as $2r$.

$\Rightarrow 2q^2 = (2r)^2 \Rightarrow 4r^2 = 2q^2 \Rightarrow q^2 = 2r^2 \Rightarrow q^2$ is even $\Rightarrow q$ can be written as $2nS$. ②

due to ① and ② $\Rightarrow$ Error, if P and q are integers with no common divitions $\Rightarrow$ P and q can't be all Even.

Finally, $\sqrt{2}$ is an irrational num.

Review for CSE 355

CFG Defination

Suppose you are given a Free Grammar consisting of $G(V,T,P,S)$ where V is a set of variables (also called non-terminals), T is a set of terminal symbols, P is a set of production rules of the form $X \to \alpha$, where $\alpha \in (V \cup T)^*$, and $S \to V$ is the start symbol. Define the language of G.

(a) Define when $\alpha X \beta \Rightarrow \alpha \gamma \beta$

(b) Now define $\alpha \overset{*}{\Rightarrow} \beta$

(c) Use $\alpha \overset{*}{\Rightarrow} \beta$ define the language of G denoted by $L(G)$.

Solution:

(a) $\alpha X \beta \Rightarrow \alpha \gamma \beta$ if $\alpha, \gamma, \beta \in (V \cup T)^*$ and $X \to \gamma$ is a rule in P.

(b) $\alpha \overset{*}{\Rightarrow} \beta$ if (i) $\alpha \Rightarrow \beta$ or (ii) $\alpha \overset{*}{\Rightarrow} \gamma$ and $\gamma \overset{*}{\Rightarrow} \beta$

(c) $L(G) = \{ w \in T^* \mid S \overset{*}{\Rightarrow} w \}$

CSE 355

Defination DFA

Let $P=(Q,\Sigma,\Gamma,\delta,q_0,F)$ be a PDA. The language of P, denoted by $L(P)$ can be defined as follows: $\{w \mid (q_0,w,\epsilon) \vdash^* (q,\epsilon,a)$, where $q \in F$ and $a \in (\Sigma \cup \Gamma)^*\}$; where a triplet of the form (q,a,β) represents a configuration where a is the remaining input string and β is the stack content; and $(q,a,\beta) \vdash^* (q',a',\beta')$ means that the configuration (q,a,β) can transition to (q',a',β') in one or more steps. Give an inductive definition of $(q,a,\beta) \vdash^* (q',a',\beta')$.

Base case: Input a change β to γ

$a,\beta \to \gamma$

$(q, aX, \beta Y) \vdash (q', X, \gamma Y)$ if $(q',\gamma) \in \delta(q,a,\beta)$ where $a \in \Sigma_\epsilon$; $X \in \Sigma^*$

$\beta, \gamma \in \Gamma_\epsilon$; $Y \in \Gamma^*_\epsilon$

$(q,X,Y) \vdash^* (q',X',Y')$ if $(q,X,Y) \vdash (q',X',Y')$

Input ϵ

Induction step:

$(q,X,Y) \vdash^* (q',X',Y')$ if $(q,X,Y) \vdash (q_0,X_0,Y_0)$ and $(q_0,X_0,Y_0) \vdash^* (q',X',Y')$

Review for CSE 355

Chomsky Normal Form:

A CFG is in Chomsky Normal Form if each rule is of the form

$A \to BC$

$A \to a$

Where
- a is any terminal
- ABC are variables
- B, C cannot be start variable

☆ However, $S \to \varepsilon$ is allowed.

E.g. Convert CFG to CNF

$A \to BAB \mid B \mid \varepsilon$

$B \to 00 \mid \varepsilon$

Step1: Add a new start variable S_0 and the rule $S_0 \to S$, where S is the start variable of G. ensures that start variable of the new grammar does not appear on right side.

$S_0 \to A$

$A \to BAB \mid B \mid \varepsilon$

$B \to 00 \mid \varepsilon$

Step2: Remove ε rules. A terminal X is nullable if:

(1) We have $X \to \varepsilon$ in the rules.

or (2) We have $X \to A_1 A_2 \cdots A_n$ and all of $A_1, A_2, A_3 \cdots A_n$ are nullable.

E.g. $A \to BC$, $B \to d \mid \varepsilon$, $C \to e \mid \varepsilon$ $\Rightarrow$ A, B, C are all nullable $\Rightarrow$

$S_0 \to A \mid \varepsilon$ (S_0 is nullable, since A, B, C are nullable.)

$A \to BC \mid B \mid C$ (All possible situations)

$B \to d$

$C \to e$ (For $X \to a$, remove ε)

E.g. $A \to B_1 B_2 B_3$ (B_2 and B_3 are nullable)

$\Downarrow$

$A \to B_1 B_2 B_3$

$A \to B_1 B_2 \mid B_1 B_3 \mid B_1$

E.g. B_1, B_2, B_3 are all nullable.

$A \to B_1 B_2 B_3$

$A \to B_1 B_2 \mid B_1 B_3 \mid B_2 B_3$

$A \to B_1 \mid B_2 \mid B_3$

Review for CSE355

CNF

Step2: Remove ε.

S_0, A, B are all nullable.

$S_0 \to A \mid \varepsilon$

$A \to BAB \mid BA \mid AB \mid BB \mid A \mid B$

$B \to 00$

Step3: Remove the unit rules.

rules We remove the unit rule $A \to B$. To do so, for each rule $B \to u$ (where u is a string of variables and terminals), we add the rule $A \to u$.

- E.g. if we have $A \to B$, $B \to aC$, $B \to CC$

  we add $A \to aC$, $A \to CC$

Possible pairs:

(S_0, S_0) (S_0, A) (S_0, B)

(A, A) (A, B)

(B, B)

(S_0, S_0) $S_0 \to \varepsilon$

(S_0, A) $S_0 \to BAB \mid BA \mid AB \mid BB \mid A \mid B$

(S_0, B) $S_0 \to 00$

(A, A) $A \to BAB \mid BA \mid AB \mid BB \mid A \mid B$

(A, B) $A \to 00$

(B, B) $B \to 00$

Summary:

$S_0 \to BAB \mid BA \mid AB \mid BB \mid 00 \mid \varepsilon$ Only S_0 can have ε

$A \to BAB \mid BA \mid AB \mid BB \mid 00$

$B \to 00$

CNF

Step 4: Convert to CNF

rules Suppose we have a rule $A \to u_1 u_2 \cdots u_k$, where $k > 2$ and each u_i is a variable or terminal. We replace this rule by

$A \to u A_1$, $A_1 \to u_2 A_2$, $A_2 \to u_3 A_3 \cdots$, $A_{k-2} \to u_{k-1} u_k$

$S_0 \to BX \mid BA \mid AB \mid BB \mid 00 \mid \varepsilon$

$X \to AB$

$A \to BX \mid BA \mid AB \mid BB \mid 00$

$B \to 00$

☆ According to CNF, $B \to 00$ is not allowed.

To remove a rule $A \to u_1 u_2$ with some terminal on the right side, we replace the terminal u_i by a new variable U_i and add the rule

$U_i \to u_i$

☆ After the change, the string on the right hand side of any rule is exactly a terminal or two variables.

E.g. $S \to aSb$

$\Downarrow$

$S \to aX$
$X \to Sb$ (Still not CNF)

$\Downarrow$

$S \to AX$
$A \to a$
$X \to SB$
$B \to b$ (This is CNF)

We have $B \to u_1 u_1$

So, $B \to ZZ$
$Z \to 0$ need to be added

Finally:

$S_0 \to BX \mid BA \mid AB \mid BB \mid ZZ \mid \varepsilon$

$X \to AB$

$A \to BX \mid BA \mid AB \mid BB \mid ZZ$

$B \to ZZ$

$Z \to 0$

Review for CSE 355

Context Free Grammer.

Write CFG For language.

$L1 = \{w \mid w \in \{0,1\}^*$, w has an odd length$\}$

Logic: $S \to xA$ xA $xxxA$ x appears 1, 3, 5, 7 ... times.

$A \to xS \mid \varepsilon$ xxS $xxxe$

Only A can return

Solution: $S \to \boxed{0A \mid 1A}$ Only require Length

$A \to \boxed{0S \mid 1S} \mid \varepsilon$

$L2 = \{w \mid w \in \{0,1\}^*$, w does not contain substring $110\}$.

When meet 11 the next should be 1 not 0.

$S \to 0S \mid 1A \mid \varepsilon$ 111

$A \to 0S \mid 1B \mid \varepsilon$

$B \to 1B \mid \varepsilon$

$L5 = \{w \mid w \in \{0,1\}^*$, every even position of w is 0's$\}$

Logic: same as L1:

do sth here

$S \to \boxed{0A \mid 1A \mid \varepsilon}$ requires position

$A \to \boxed{0S} \mid \varepsilon$ can be ε, since 0 is an even number.

$L8 = \{w \mid w$ is a string of a's and b's and w has even or zero number of b's$\}$.

Logic: $S \to xA \mid \varepsilon$ only S can return

$A \to xS$

xA $xx\varepsilon$ return

xxS

Even number of b's

Solution: $S \to bA \mid aS \mid \varepsilon$ all a's or ε

$A \to bS \mid aA$

Review for CSE355

Write CFG For Languages.

$L_9 = \{a^i b^j c^k \mid i=j \text{ or } j=k \text{ where } i,j,k \geq 0\}$

When $i=j$, nothing requirements for c^k.

$S_1 \to A_1 C_1$

$A_1 \to aA_1b \mid \varepsilon$ $a^i b^j$ $i=j$ i,j can be 0.

$C_1 \to cC_1 \mid \varepsilon$ c's, can have 0 c's.

When $j=k$:

$S_2 \to A_2 C_2$

$A_2 \to aA_2 \mid \varepsilon$ a's, can have 0 a's

$C_2 \to bC_2c \mid \varepsilon$ $b^j c^k$, $j=k$ j,k can be 0

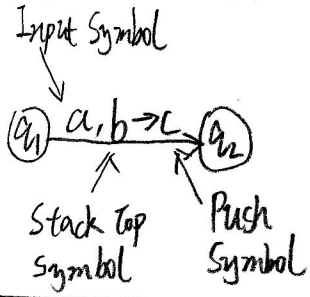
Combination:

$S \to S_1 \mid S_2$

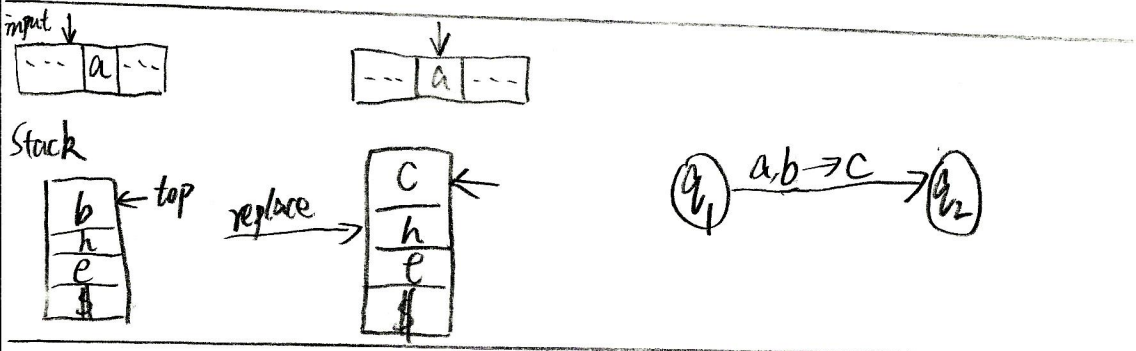
PDA

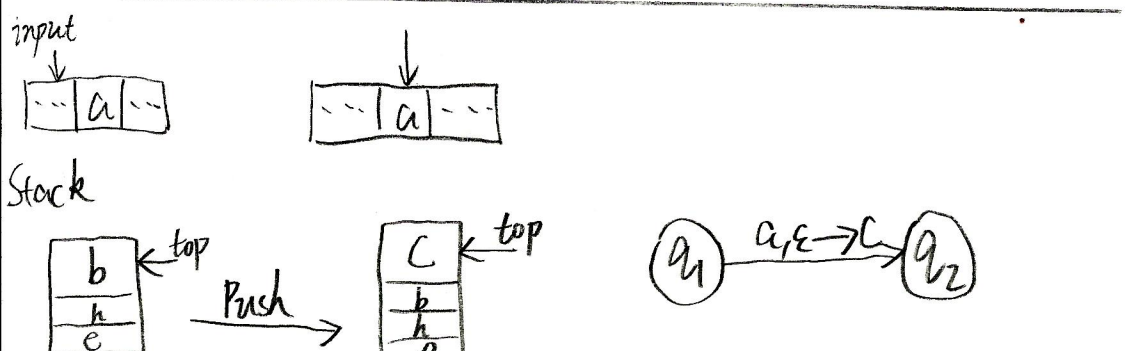
- A PDA is a sextuple $(Q, \Sigma, A, \delta, q_0, F)$, where:
  - Q is the set of states
  - Σ is the input alphabet
  - A is the alphabet for the stack (Γ in our book)
  - δ is the transition function
  - q_0 is the start state
  - F is the set of accepting states.

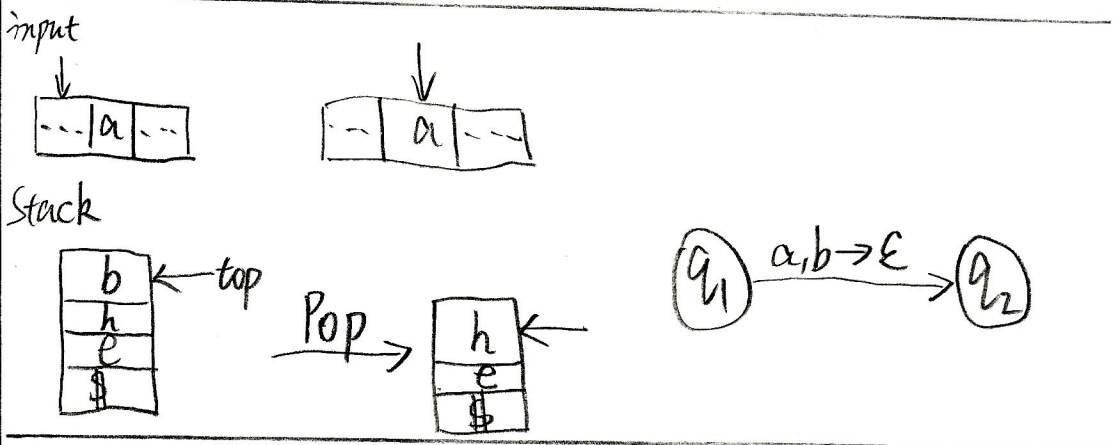
PDA

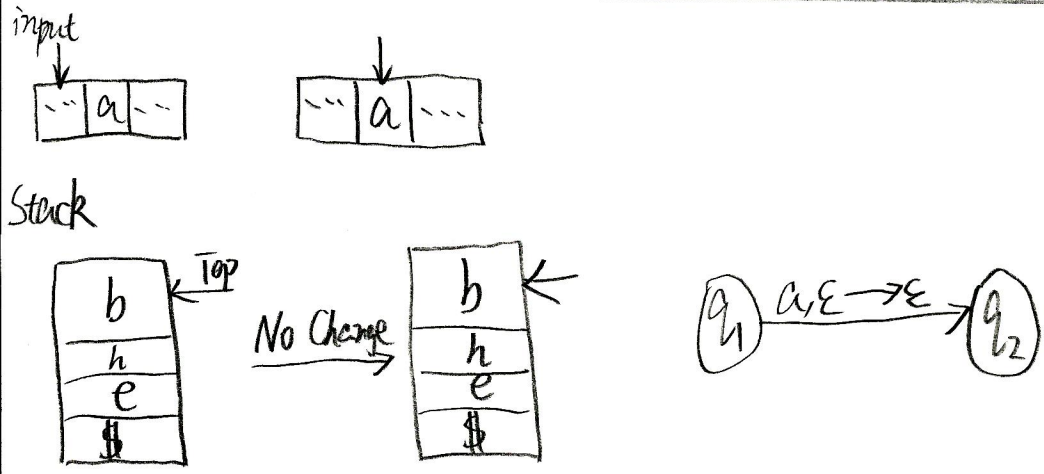


When input is a, replace b with c

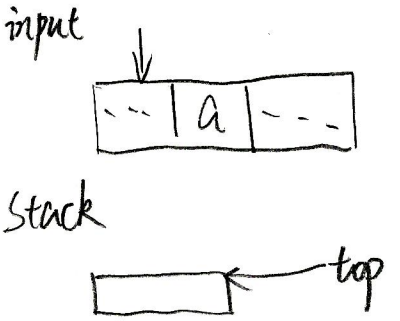








PDA



$\xrightarrow{\text{Pop}}$ Automation halts!

If the automation attempts to pop from empty stack then it helts and rejects input

---

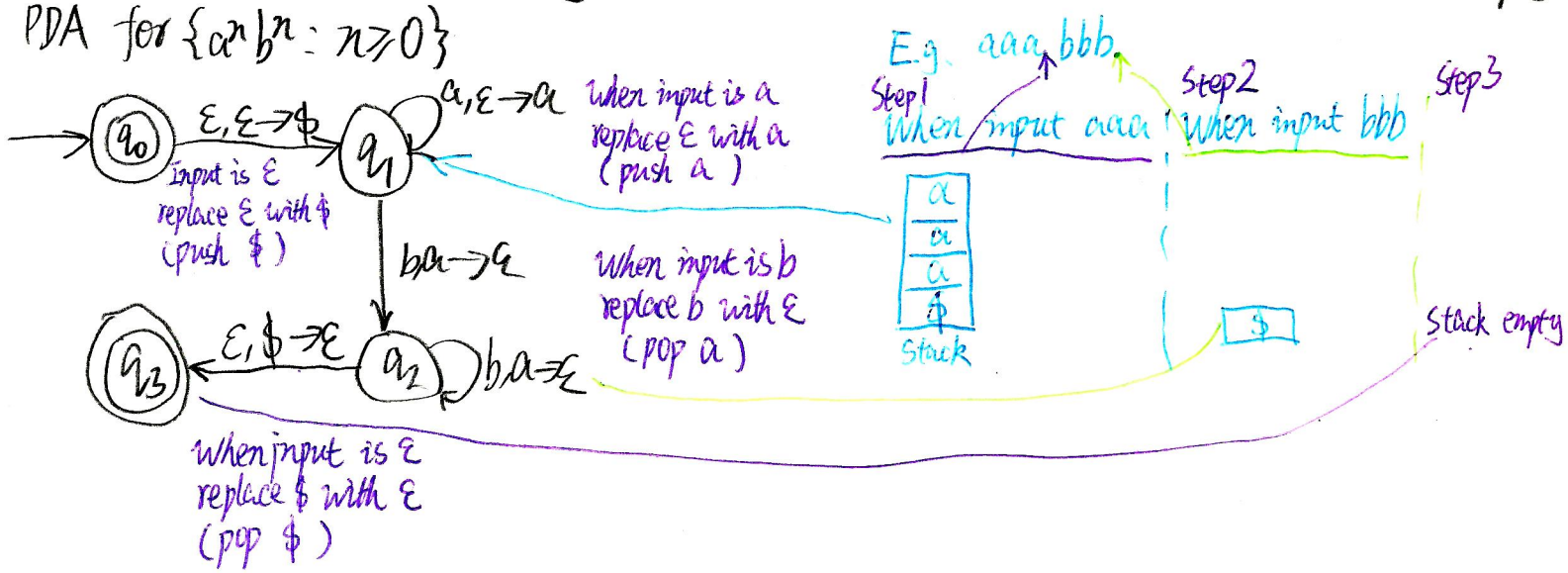
PDA Accept – Reject Status

- The PDA accepts when there exists a computation path such that:
  - The computation path ends in an accept state.
  - All the inputs is consumed
  - (no requirement for the stack)
- The PDA reject when all the patchs:
  - Either end in a non-accepting state
  - Or are incomplete (meaning that at some point there is no possible transition under the current input and stack symbols).

Is the stack empty?

- place a special symbol ($) at the bottom of the stack.
- Whenever we find the $ again we know that we reached the end of the stack.
- In order to accept a string there is no need for the stack to be empty.

A PDA for $\{a^n b^n : n \geq 0\}$



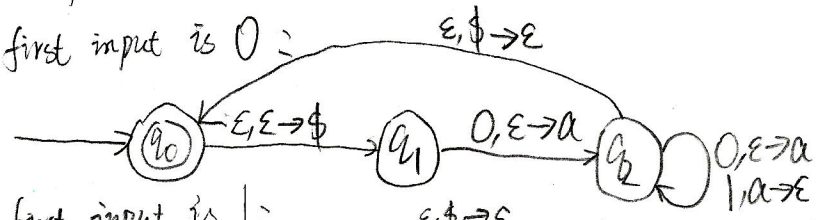
Review for CSE 355

Write PDA:

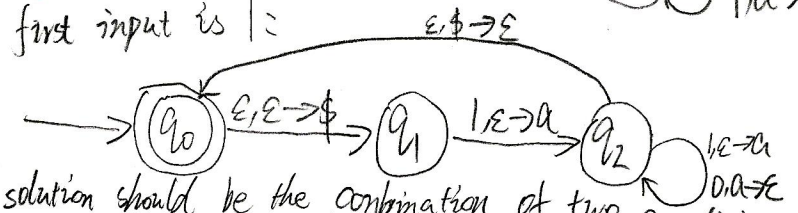
1. $L6 = \{w \mid w \in \{0,1\}^*$, w contains an equal number of 0's and 1's$\}$

Divide the problem into 2 conditions; first input is 0 or first input is 1

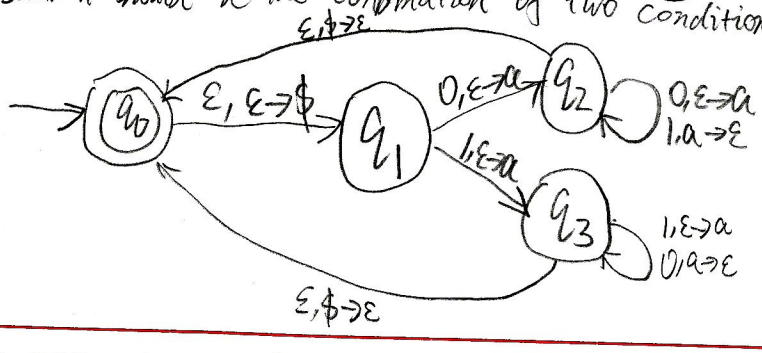
When first input is 0:



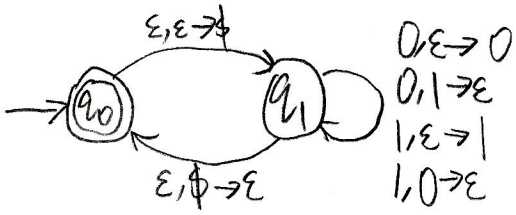
When first input is 1:



Then, The solution should be the combination of two conditions.

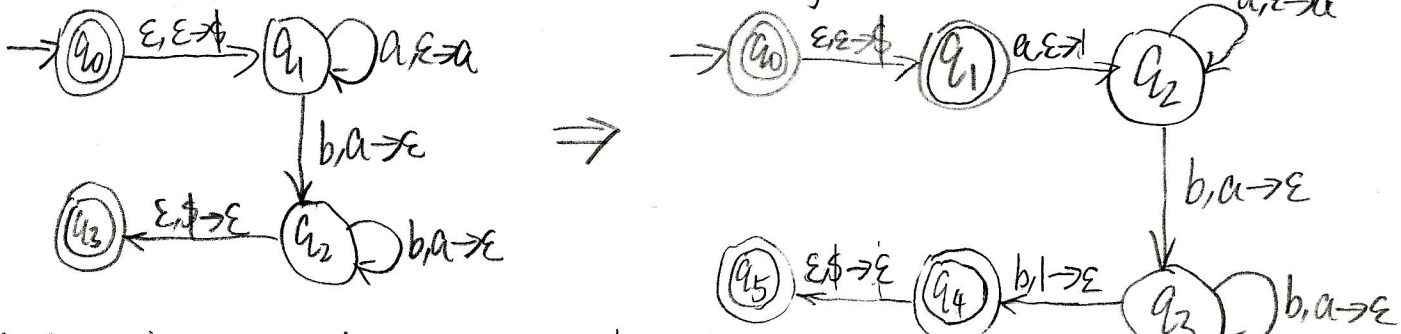


Solution 2: NPDA



2. $L7 = \{a^m b^n \mid m \neq n\}$

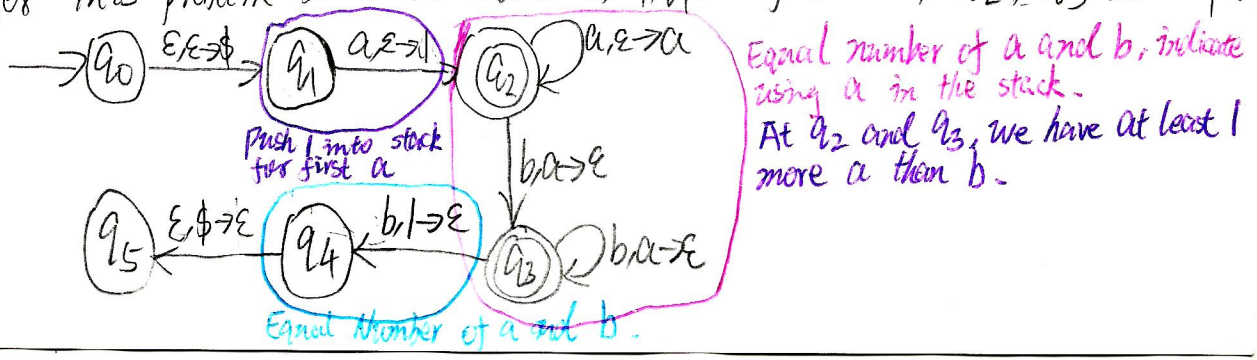
We know that the PDA for $\{a^n b^n : n \geq 0\}$ is



And we know that the PDA accepts when there exists a computation path such that:

1. The computation path ends in an accept state
2. All the inputs is consumed
3. No requirement for the stack

Thus, the solution for this problem should be make q_0, q_1, q_4 as reject status, q_2, q_3 as accept.



Review for CSE 355

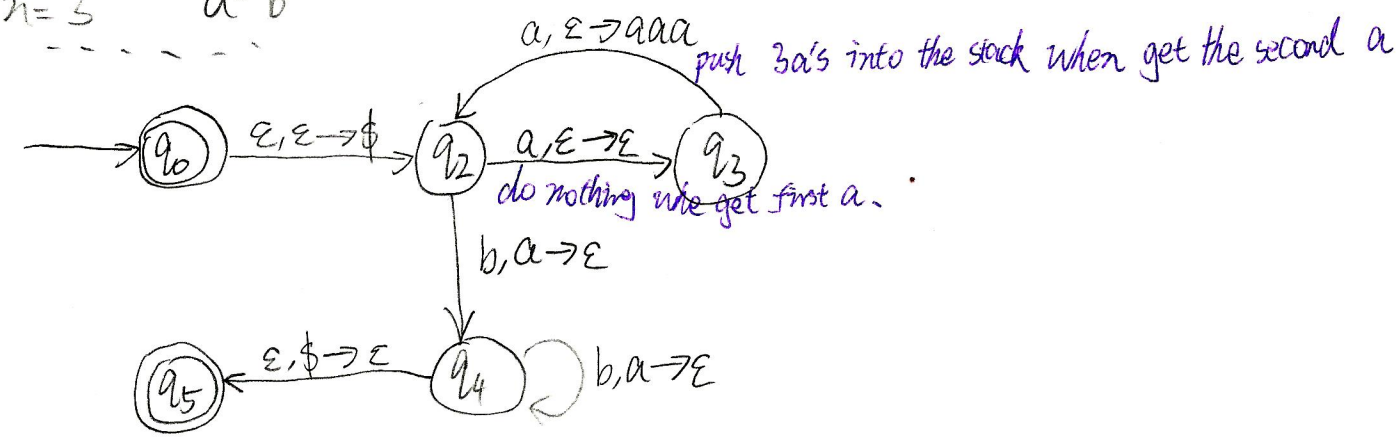
Write PDA:

$\{a^{2n} b^{3n}\}$

accept states:

$n=0$	ε
$n=1$	$a^2 b^3$
$n=2$	$a^4 b^6$
$n=3$	$a^6 b^9$

$\Rightarrow$ idea, when get 2 a's, replace 2 a's with 3 a's.



DFA to CFG

For example to get grammar: $\{ 0^n 1^n \mid n \geq 0 \} \cup \{ 0^n 1^n \mid n \geq 0 \}$

$G1 \qquad S_1 \rightarrow 0S_1 1 \mid \varepsilon$

$G2 \qquad S_2 \rightarrow 1S_2 0 \mid \varepsilon$

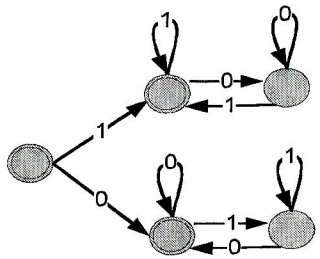
add this starting substitution rule

$S \rightarrow S_1 \mid S_2$

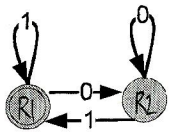
2. If the language is regular - create a DFA and then convert to CFG as follows:

a. Make a variable Ri for each state qi of the DFA.
b. Add the rule Ri →aRj to the CFG if there is a DFA transistion (with a) from state Ri to Rj.
c. Add the rule Ri → ε if qi is an accept state.
d. Make R0 the start variable - where q0 is the start state of DFA.

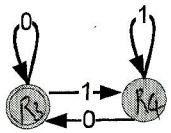
example:



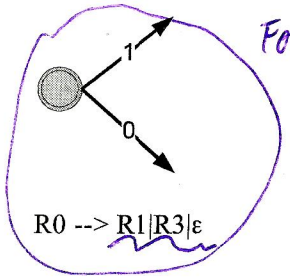
let's start with:



R1 --> 0R2|1R1|ε R1 is an accept state.
R2 --> 1R1|0R2



R3 --> 1R4|0R3|ε
R4 --> 0R3|1R4



For R0 (start)

R0 --> R1|R3|ε

Test out the resulting grammar.

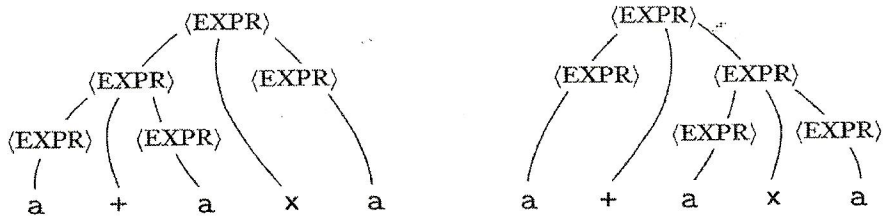
c. use this substitution rule R-->uRv

Example: 000111 S-->0S1|ε

==========

Ambiguous grammar - grammar generates the same string in multiple ways (has several different parse trees)

E → E+E | E x E | (E) | a



Two different parse tree for same string a+a x a

E-->ExE-->E+ExE-->a+ExE-->a+axE-->a+axa
E-->E+E-->a+E-->a+ExE-->a+axE-->a+axa

Generated ambiguously - 2 different parse trees, not 2 different derivations (same derivated string)

Leftmost derivation - leftmost variable is the one replaced
<<some CFL can be generated only my ambiguous grammars (inherently ambiguous)>>

CFG To PDA

转移函数定义如下。从初始化栈开始，把符号 $ 和 S 推入栈，实现步骤 1 的形式描述是：$\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$。然后进行步骤 2 主循环中的转移。

首先处理情况（a），这时栈顶是一个变元。令 $\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid A \to w$ 是 R 中的一条规则$\}$。

其次处理情况（b），这时栈顶是一个终结符。令 $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$。

最后处理情况（c），这时栈顶是空栈标记符 $。令 $\delta(\text{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$。

图 3-11 给出 P 的状态图。

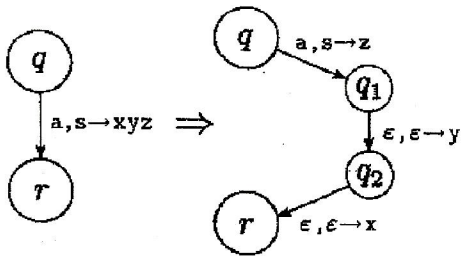


图 3-10　缩写 $(r, xyz) \in \delta(q, a, s)$ 的实现

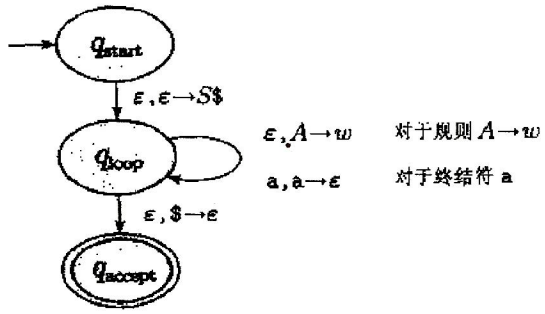


图 3-11　P 的状态图

例 3.14　利用在引理 3.13 中开发的过程，把下述 CFG G 转换成一台 PDA P_1。

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

转移函数如图 3-12 所示。

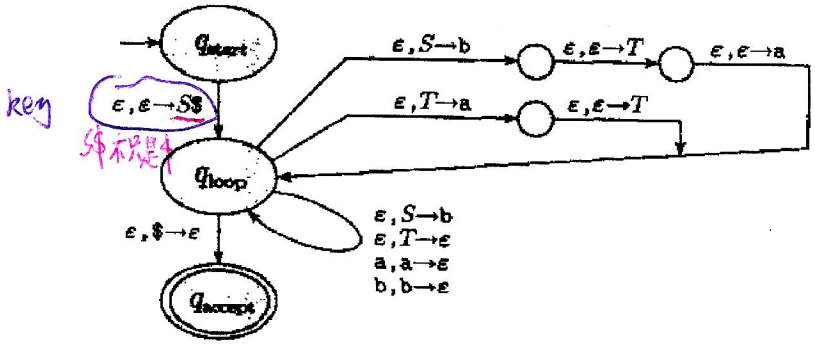


图 3-12　P_1 的状态图

下面证明定理 3.12 的反方向。对于正方向我们给出一个把 CFG 转换成 PDA 的过程。主要思想是设计自动机，使得它模拟文法。现在要给出一个以相反方式进行的过程：把 PDA 转换成 CFG。设计模拟自动机的文法。这项工作更困难，“程序设计”一台自动机比“程序设计”一个文法容易。

引理 3.15　如果一个语言被一台下推自动机识别，则它是上下文无关的。

证明思路　现有一台 PDA P，要构造一个 CFG G，它产生 P 接受的所有字符串。换句话说，如果一个字符串能使 P 从它的起始状态转移到一个接受状态，则 G 应该产生这个字符串。

Review For CSE 355

Convert CFG to PDA

1. For all terminals, we will have $d, d \to \varepsilon$ d is a terminal
2. For all transition rule $X \to \beta$ we have $\varepsilon, X \to \beta$ X is non terminal, β is terminal

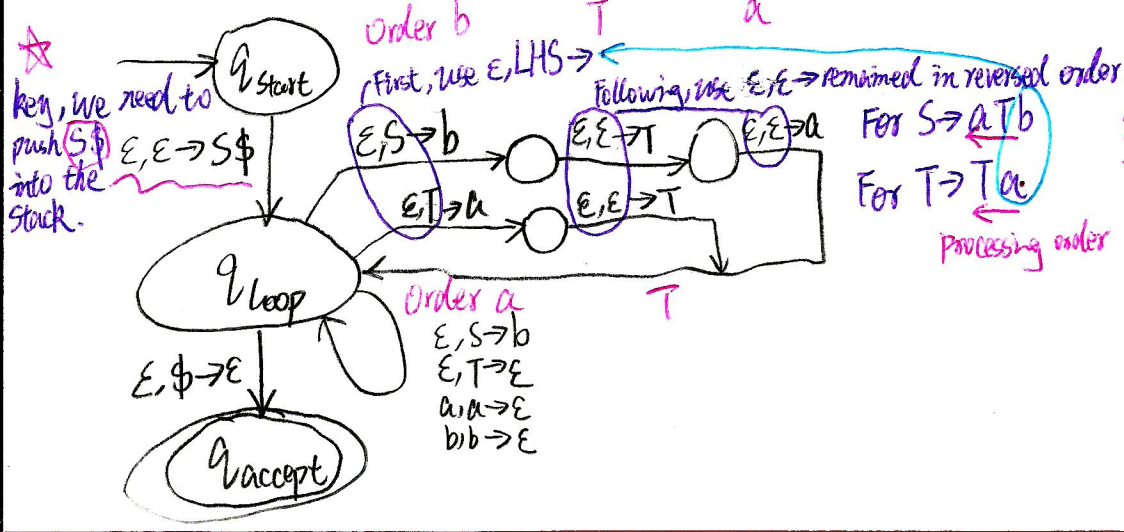
Example 1:

$S \to aTb \mid b$
$T \to Ta \mid \varepsilon$

$\Rightarrow$

$S \to aTb$
$S \to b$ rule 2 $\varepsilon, S \to b$
$T \to Ta$
$T \to \varepsilon$ rule 2 $\varepsilon, T \to \varepsilon$

non Terminals are a and b:
According to rule 1:
$a, a \to \varepsilon$
$b, b \to \varepsilon$



Since we need to push rules into the stack, so we need to use reversed order.

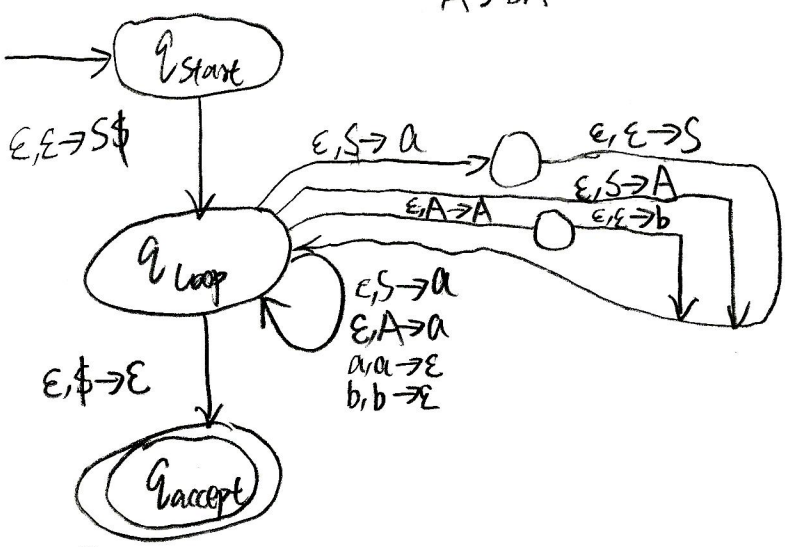
Example 2:

$S \to Sa \mid A \mid a$
$A \to a \mid bA$

$\Rightarrow$

$S \to Sa$
$S \to A$
$S \to a$ rule 2 $\varepsilon, S \to a$
$A \to a$ rule $\varepsilon, A \to a$
$A \to bA$

non Terminals a and b
$a, a \to \varepsilon$
$b, b \to \varepsilon$



Ambignous?

For "aaa" Find different syntax trees:

$S \to Sa \to Saa \to aaa$

$S \to Sa \to Saa \to Aaa \to aaa$

Convert CFG to PDA

Example 3:

$S \to AB \mid \varepsilon$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$

$\Rightarrow$

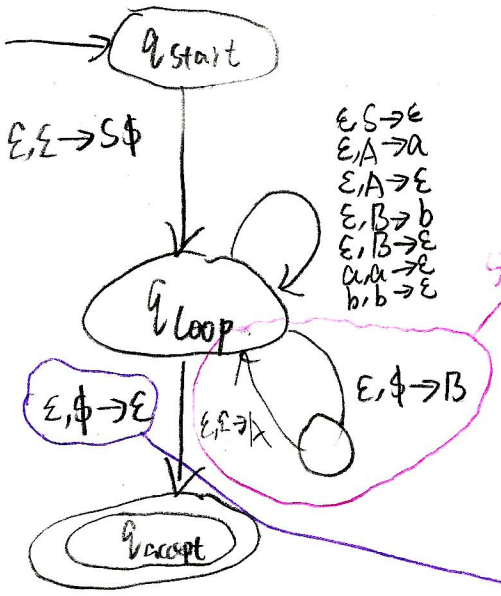
$S \to AB$
$S \to \varepsilon$
$A \to a$
$A \to \varepsilon$
$B \to b$
$B \to \varepsilon$

rule 2

$\varepsilon, S \to \varepsilon$
$\varepsilon, A \to a$
$\varepsilon, A \to \varepsilon$
$\varepsilon, B \to b$
$\varepsilon, B \to \varepsilon$

non Terminals

$a, a \to \varepsilon$
$b, b \to \varepsilon$



simple

Different Presentation:

$\delta(q_0, \varepsilon, \varepsilon) \to \{(q_1, S\$)\}$
$\delta(q_1, \varepsilon, S) \to \{(q_1, AB)\}$
$\delta(q_1, \varepsilon, S) \to \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, A) \to \{(q_1, a)\}$
$\delta(q_1, \varepsilon, A) \to \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, B) \to \{(q_1, b)\}$
$\delta(q_1, \varepsilon, B) \to \{(q_1, \varepsilon)\}$
$\delta(q_1, a, a) \to \{(q_1, \varepsilon)\}$
$\delta(q_1, b, b) \to \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, \$) \to \{(q_f, \varepsilon)\}$ $\to \varepsilon, \$ \to \varepsilon$

means transfer from state q_1 to q_f, when input is ε, change \$ to ε.

pump out

Ambiguous?

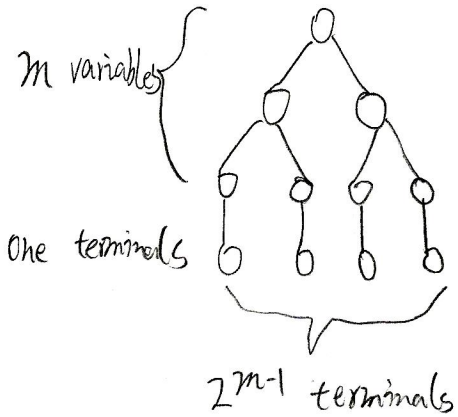
For "a b" Find different syntax trees:

$S \to AB \to aB \to ab$
$S \to AB \to Ab \to ab$

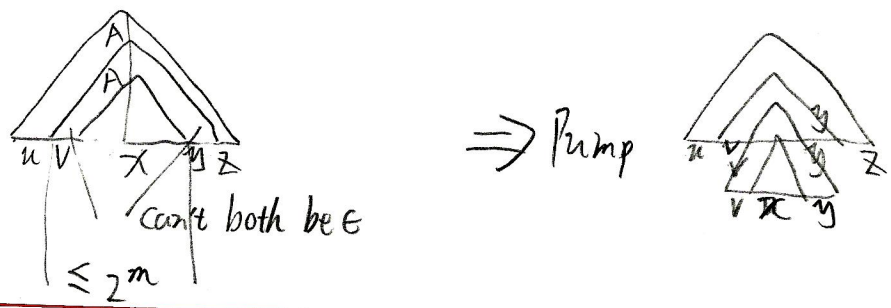
CSE 355

Proof of Lemma:

If the parse tree of a CNF grammar are of length $\leq m+1$, then the longest yield has length 2^{m-1}, as in:



Now, Let us find parser tree for $m+1$ variables.



State the pumping lemma for CF.

If the language L is CF then exist a number $p > 0$ such that for all string w, $|w| \geq p$, we can write $w = uvxyz$ such that:

$$uv^ixy^iz \in L \quad \forall i \geq 0$$

$$|vy| > 0$$

$$|vxy| \leq p$$

Review for CSE 355

Pumping Lemma

$\{a^x b^y c^z \mid z > x,\ z > y\}$

1st. suppose that L is CF

If the language L is context free then there exists a number $p > 0$ such that for all string w, $|w| > p$, we can write $w = uvxyz$ such that

$$uv^i x y^i z \in L \quad \forall i \geq 0$$
$$|vy| > 0$$
$$|vxy| \leq p$$

$a^n b^n c^{n+1}$

Consider the string $w = a^n b^n c^{n+1}$

$a^n b^n$

Case 1: if vxy does not contain c's, v or y must contains at least one a's or b's $\Rightarrow$ uv^4xy^4z has more a's than c's or more b's than c's. $\Rightarrow uv^4xy^4z \notin L$

vy contains a, vy contains b, vy contains ab — can't be empty

$|vy| > 0 \Rightarrow$ we actually considered 3 situations: $2^2 - 1$

Case 2: if vxy contains b's and c's, y does not contain c's $\Rightarrow$ v contains b's. Similar to case 1, $uv^4xy^4 \notin L$

$b^n c^{n+1}$ vy contains b's

Case 3: if vxy contains b's and c's, y contains c's $\Rightarrow uv^0xy^0z$ has the same number (or less) of c's as a's $\Rightarrow uv^0xy^0z \notin L$

$b^n c^{n+1}$ contains at least 1 c's removed at least 1 c's $\Rightarrow |a's| \geq |c's|$

$i \geq 0 \Rightarrow i$ can be 0

Case 4: if vxy contains only c's. Similar to case 3, uv^0xy^0z has the same number (or less) of c's as a's $\Rightarrow uv^0xy^0z \notin L$

c^{n+1} at least removed 1 c's

There is no division of string w that can be pumped. Therefore, L is not context free.

$\{s_1 0 s_2 0 \ldots 0 s_k \mid k \geq 2,$ each $s_i \in \{a,b\}^*$, and $s_i = s_j$ for some $i \neq j\}$

If the language L is CF, then there exists a number $p > 0$ such that for all string w, $|w| > p$ we have $w = uvxyz$ and

$$uv^i xy^i z \in L \quad \forall i \geq 0$$
$$|vy| > 0$$
$$|vxy| \leq p$$

$a^p b^p 0 a^p b^p$ or $a^p b^p$

contains 0's

go across 0's but v, y do not contain 0's.

Consider $w = s_1 0 s_2$, $s_1 = s_2 = a^p b^p$. $\Rightarrow a^p b^p 0 a^p b^p$

Case 1: if vxy is totally in s_1 or s_2, v or y should contains at least a or b or ab $\Rightarrow uv^2xy^2z \Rightarrow$ break the equality of s_1 and s_2 $\Rightarrow uv^2xy^2z \notin L$

Case 2: if v or y contains 0's, uv^2xy^2z has more then one 0's $\Rightarrow uv^2xy^2z \notin L$.

w have only one 0

case 3: if vxy goes across 0's but both v and y do not contain 0's. We have either v contains only b's y contains only a's or both of them are true. In any of these subcases, s_1 and s_2 of uv^2xy^2 are not the same any $\Rightarrow uv^2xy^2z \in L$

There is no division of string w that can be pumped. Therefore, L is not CF.

Review for Test 2

1. Give a DFA $=(\Sigma, Q, q_0, F, \delta)$ when do we say that two states q and q' are equivalent to A.

if q', q are equivalent, which means input same thing, can get to final state.

Solution: $\{a \mid \hat{\delta}(q, a) \in F\} = \{a \mid \hat{\delta}(q', a) \in F\}$

2. Components of a CFG.

- Start symbol
- Nonterminals
- Terminals
- Rules.

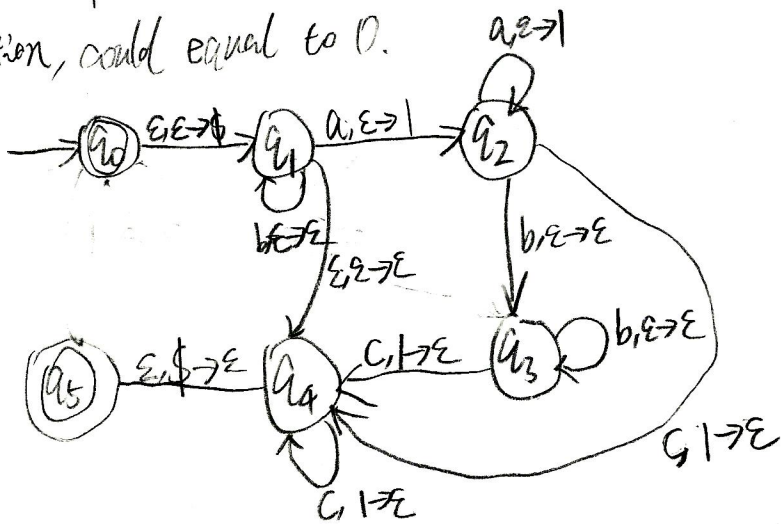
3. $S \to 0A0 \mid 1B1$

$A \to 0A0 \mid 1A1 \mid 1 \mid 11$

$B \to 0B0 \mid 1B1 \mid 0 \mid 00$

4. Write a push down automata that accepts the language $\{a^n b^k c^n \mid n, k \geq 0\}$

Attention, could equal to 0.



Review for Test 2

5. For a deterministic PDA the transition function δ is a function from what to what?

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \{Q \times \Gamma\} \cup \{\phi\}$$

For deterministic:

$$\left.\begin{array}{l} -\ \delta(q, a, b) \\ \delta(q, q, \varepsilon) \\ \delta(q, \epsilon, b) \\ \delta(q, \varepsilon, \varepsilon) \end{array}\right\}$$ at most one of them is non empty.

input x replace z with β

6. Find δ for PDA: $(s, xa, z\gamma) \vdash (s', a, \beta\gamma)$

$$s \xrightarrow{x,\ z \to \beta} s'$$

$$(s', \beta) \in \delta(s, x, z)$$

7. PL for CFL

If the language L is CF then there exists a number $p > 0$, such that for all string w, $|w| > p$, we can write $w = uvxyz$ such that

$$uv^ixy^iz \in L \quad \forall i \geq 0$$
$$|vy| > 0$$
$$|vxy| \leq p$$

10. $S \to aBCdE$
$B \to bb$
$C \to cc$
$E \to ee$

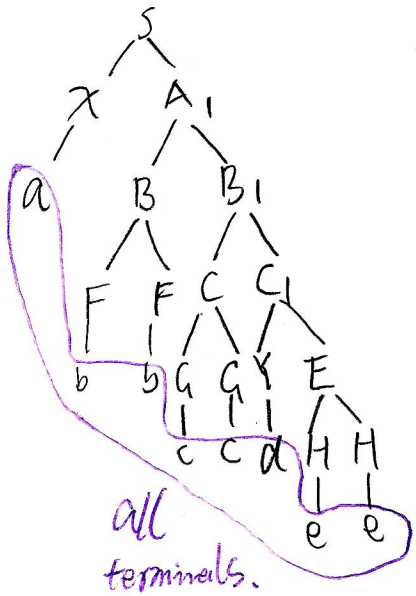
$\Rightarrow$

$S \to aA_1$
$A_1 \to BB_1$
$B_1 \to CC_1$
$C_1 \to dE$

$\Rightarrow$

$S \to XA_1$
$X \to a$
$A_1 \to BB_1$
$B_1 \to CC_1$
$C_1 \to YE$
$Y \to d$ allowed
$B \to FF$
$F \to b$
$C \to GG$
$G \to c$
$E \to HH$
$H \to e$

$E \to$ Terminal Terminal $\Rightarrow$ not allowed



Proplity of CFL

Show that if L_1 and L_2 are regular languages then $L_1 \cap L_2$ is regular.

If L_1 and L_2 are RL, then there are DFAs for them.

L_1: $M_1(Q_1, \Sigma_1, \delta_1, q_1, F_1)$

L_2: $M_2(Q_2, \Sigma_2, \delta_2, q_2, F_2)$

Construct $M = (Q, \Sigma, \delta, q, F)$

1. $\Sigma = \Sigma_1 \cap \Sigma_2$
2. $q_0 = (q_1, q_2)$
3. $F = (S_1, S_2)$ $\quad S_1 \in F_1 \quad S_2 \in F_2$
4. $Q = (S_1, S_2)$ $\quad S_1 \in Q_1 \quad S_2 \in Q_2$
5. $\delta((S_1, S_2), a) = (\delta_1(S_1, a), \delta_2(S_2, a))$ for $a \in \Sigma$ and $(S_1, S_2) \in Q$

M recognizes $L_1 \cap L_2$

Q2: Show that if L_1 is a CFL and L_2 is a RL then $L_1 \cap L_2$ is CF.

$\because$ L_1 is CFL $\Rightarrow$ there is a PDA for it. $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, F_1)$

$\because$ L_2 is RL $\Rightarrow$ there is a DFA for it. $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$

We want to Construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q, F)$

$\because$ we need $L_1 \cap L_2$ $\Rightarrow$

1. $\Sigma = \Sigma_1 \cap \Sigma_2$
2. $q_0 = (q_1, q_2)$
3. $F = \{(S_1, S_2) \mid S_1 \in F_1, S_2 \in F_2\}$
4. $Q = \{(S_1, S_2) \mid S_1 \in Q_1, S_2 \in Q_2\}$
5. $\delta((q, s), a, \alpha)$ contains $(q', \delta_2(s, a)), \gamma)$ if $\delta_1(q, a, \alpha)$ contains (q', γ) and

   $\delta((q, s), \varepsilon, \alpha)$ contains $((q', s), \gamma)$ if $\delta_1(q, \varepsilon, \alpha)$ contains (q', γ)

   where $s \in Q_2$, $a \in \Sigma$, $q, q' \in Q_1$ $\quad \alpha, \gamma \in \Gamma_\varepsilon$;

M recognizes $L_1 \cap L_2$.

Relationship between Regular, CF, Decidable, NPC

Regular (Write DFA/NFA/RE/ε-NFA) $O(n)$
Not Regular (Show it using PL)
Content Free (Write CFG/PDA) $O(n^3)$
Not CF (Does not satisfy PL for CFL)
Decidable (TM/any kind of program is Ok) $O(n^x)$ no limited
Not Decidable (reducing the know undecidable to this problem)
NP-Complete (① in NP ② show every problem in NP can be reduced to this problem) or show a known NPC problem can be reduced to this problem.)

How to show undecidable:
Suppose F_3 is undecidable
To show F_2 is undecidable F_3 [call F_2]

DFA: $A=(\Sigma, Q, q_0, F, \delta)$ 5 Item
- Σ: a finite set of input symbols
- Q: a finite set of states
- q_0: a start state
- F: a set of accept states ($F \subseteq Q$)
- δ: a transition Function ($\delta: Q \times \Sigma \to Q$)

Components of CFG:
- Start symbol
- Non terminals
- Terminal
- Rules

TM: $M=(Q, \Sigma, \Gamma, \delta, q_0, q_f, q_{reject})$ 7 Item
- Q: States
- Σ: Input alphabet
- Γ: Tape alphabet
- δ: Transition Function
- q_0: Initial state
- q_f: Final state
- q_{reject}: reject state.

PDA: $M=(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ 7 Items (optional) 6
- Q: is a finite set of states
- Σ: is a finite set of input alphabet
- Γ: is a finite set of stack alphabet
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ transition
- $q_0 \in Q$ start state
- $Z \in \Gamma$ init stack symbol
- $F \in Q$ accepting states.

Summary of NP / NPC

What is NP:

A class of such problems have the property that they can be solved using a non deterministic TM in polynomial number (in size of the input) of transitions. These problems are said to belong to the NP.

More simply, a problem is in the class NP if there is a polynomial time algorithm for verifying the correctness of a given potential solution.

What is NPC:

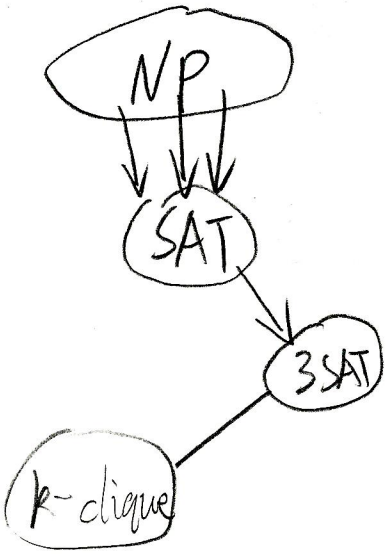
The class NPC is a subset of the class NP, with the property that if we can solve any problem in that class NPC in polynomial time then we can solve all the problems in NP in polynomial time.

How to show NPC:

2 ways:

A: show the problem is in NP, and show that all problems in NP can be polynomially reduced to the problem.
a general non deterministic TM

B. Show the problem is in NP, and show that a known NPC can be polynomially reduced to the problem.



Satisfiability is NPC

To prove a problem is NPC we need to show

① It is in NP

② every problem in NP can be polynomially reduced to it.
or show a known NP-complete can be polynomially reduce to A.

Prove satisfiability is in NP:

Given a truth assignment to all the propositions, we need to check one by one if each of the conjunct is true or not. This can be done in linear (thus polynomial) in the size of the input formula. Hence, Satisfiability is in NP.

To show satisfiability is NP-complete we now need to show that every problem in NP can be expressed as the satisfiability of a propositional formula. Since any problem in NP can be solved using a non-deterministic TM, all we need is to express to operation of a TM as formula.

Suppose the non-deterministic TM has $\delta(q, X) = \{(q', Y', L), (q'' Y'', R)\}$. This can be expressed by a formula $A \Rightarrow B \vee C$ where the proposition A means that current state is q and the tape symbol is X. B means that the current state is q', the tape symbol to one step right is Y'. C mean that current state is q'', the tape symbol to the left is Y''.

B and C means the state after transfer.

Summary

Prove SAT is NPC.

① SAT general form: $F = (X_1 \vee X_2 \cdots \vee X_n)$
Give a truth assignment to F, we can check F one by one in linear time (polynomial time) $\Rightarrow$ SAT is in NP.

② Any problem in NP can be solved by TM. $\Rightarrow$ all we need is to express operation of a TM as formula.

Suppose $\delta(q, X) = \{(q', Y', L), (q'', Y'', R)\}$. This can be expressed by formula

$A \Rightarrow B \vee C$

Prove 3-SAT is NP-complete

① Show 3-SAT is in NP

② Show 3-SAT is NP-complete — give a polynomial reduction of a known NP-Complete problem to 3-SAT.

For ①: e.g we are given $(a_1 \vee a_2 \vee a_4) \wedge (\bar{a}_2 \vee a_3 \vee \bar{a}_4) \wedge (\bar{a}_1 \vee a_4 \vee a_2) \wedge (a_4 \vee a_3 \vee \bar{a}_1)$

Give a truth assignment to all the proposition, we need to check one by one if each of the conjunt is true or not. This can be done in time linear (thus polynomial) in size of input formula. Hence, 3-SAT is in NP.

Assignment:

a_1	T
a_2	F
a_3	T
a_4	T

---

②. Show 3-SAT is NP-complete

We know that SAT Problem is NP complete $\Rightarrow$ We need to polynomial reduce SAT to 3SAT.

e.g. $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6)$ in F, Form that we obtain the following:

$$(x_1 \vee x_2 \vee Y_1) \wedge (x_3 \vee \bar{Y}_1 \vee Y_2) \wedge (x_4 \vee \bar{Y}_2 \vee Y_3) \wedge (x_5 \vee x_6 \vee \bar{Y}_3).$$

In general form a conjunt of form $F = (x_1 \vee \ldots \vee x_n)$ we obtain the 3-SAT formula

$$F' = (x_1 \vee x_2 \vee Y_1) \wedge (x_3 \vee \bar{Y}_1 \vee Y_2) \wedge \ldots \wedge (x_{n-2} \vee \overline{Y_{n-4}} \vee Y_{n-3}) \wedge (x_{n-1} \vee x_n \vee \overline{Y_{n-3}})$$

We want to show that Ⓐ if F satisfiable $\Rightarrow$ F' satisfiable and Ⓑ if F' satisfiable $\Rightarrow$ F satisfiable

For Ⓐ: When we convert $(x_1 \vee \ldots \vee x_n)$ to a 3-Sat formula, the 3-Sat formula has $n-2$ conjuncts with $n-3$ new variables. Any truth assignment of the $n-3$ new variables will make all but one conjunt true. Thus if F is satisfiable then the propositional literal that is true in F makes one of the conjuncts in F' true and the remaining $n-3$ conjuncts can be made true by the appropriate assignment of the remaining $n-3$ new variables. Here F is satisfiable implies F' is satisfiable.

Prove 3-SAT is NP-complete

For (B): Suppose F' is satisfiable. Then all its conjuct must be true. Since the truth assignment of the $n-3$ new variables will make all but one conjuct true the remaining conjuct must be made true by the assignment to one of its original (or non-new) literals. These assignments will make F true. Hence, if F' is satisfiable F would satisfiable.

Summary

If we want to prove 3-SAT is NP-complete, we nee to show 2 thing:

(1): Show 3-SAT is in NP

(2): show 3-SAT is NPC — give a polynomial reduction of a know NPC problem to 3-SAT.

For (1): we give a truth assignment to the general form of 3-SAT, we need linear time (also polynomial time) to check the assignment makes the 3-SAT to be true, Thus, the 3-SAT is in NP.

For (2): We neet to transfer general SAT form to 3-SAT form in polynomial time.

This transfer can be done by:

$$F=(X_1 \vee X_2 \vee \dots \vee X_n)$$

$$\Rightarrow F'=(X_1 \vee X_2 \vee Y_1) \wedge (X_3 \vee \bar{Y}_1 \vee Y_2) \wedge \dots (X_{n-2} \vee \bar{Y}_{n-4} \vee Y_{n-3}) \wedge (X_{n-1} \vee X_n \vee \bar{Y}_{n-3})$$

Which is in polynomial time.

Now we need to show (A) if F satisfiable $\Rightarrow$ F's satisfiable and (B) if F'satisfiable $\Rightarrow$ F satisfiable.

For (A) if we set any truth assignment of the $n-3$ new variables, $\Rightarrow$ all conjoints but one will be true, if F is satisfiable $\Rightarrow$ F will make one conjunct in F's to be true, thus F is satisfiable implies F's is satisfiable.

all Y_x to be true can't guarantee this conjunct to be true

if F is true, means at least one X_x is true. and In F', X_{n-1}, X_n can be any thing in F, since F' was ~~form~~ fromed by ourself. We assign one of X_{n-1} and X_n to be X_x in $F \Rightarrow (X_{n-1} \vee X_n \vee \bar{Y}_{n-3})$ should be also true. $\Rightarrow$ if F is satisfy the F' is also satisfiable.

For (B) if F' is true $\Rightarrow$ all $n-3$ new variable Y_x will make all but one conjunct to to true, thus this conjunct should be made true by the original literal's, since $F=(X_1 \vee X_2 \vee \dots \vee X_n) \Rightarrow$ one literal in F is true $\Rightarrow F$ is satisfiable

So, If F'satisfiable, then F satisfiable.

K-clique is NPC

The k-clique problem is: On an input of graph G and number k, find if G has a clique of size more than k or not.

① Show k-clique is in NP:

Suppose a graph has n vertices, given a set of k vertices, we need to check if each pair of those vertices are connected by an edge. This can be done in time $O(k^2)$, $k<n \Rightarrow$ can be done in $O(n^2)$. Thus, can be done in polynomial time $\Rightarrow$ K clique is in NP.

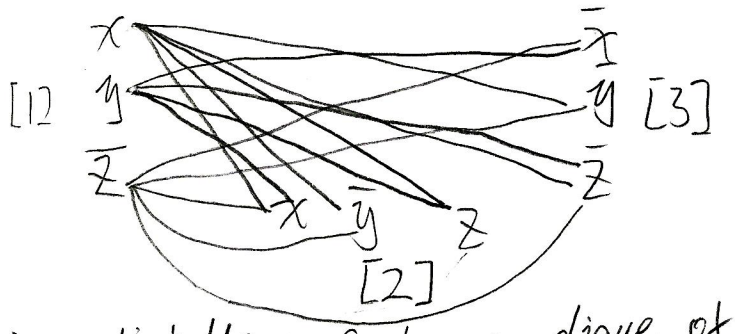
② Show a known NPC problem can be reduced to K-clique in polynomial time. — show 3-SAT can be reduced to k-clique in polynomial time.

e.g. $F = (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \ldots$

nodes will be $x[1], y[1], \bar{z}[1], x[2], \bar{y}[2], z[2], \bar{x}[3], \bar{y}[3], \bar{z}[3]$ ...

The edges of the graph are between nodes if they come from different conjoint and they are not complement.

Suppose nodes of the graph are denoted as $p[i]$ and $\bar{q}[j]$ for positions p in F_i or the $\bar{q}$ in F_j.



Ⓐ if F is satisfiable $\Rightarrow$ G has a clique of size k or larger.

If F is satisfiable $\Rightarrow$ must be a propositional literal in each of the F_i that is true. The nodes corresponding to these literals have edges between each pair as they are not complement of each other and they come from different F_is. These set of nodes make the vertices of the k-clique.

k-clique is NPC

(B) if G has a clique of size k or larger $\Rightarrow$ F is satisfiable.

$\Rightarrow$ the vertices of k clique must correspond to one propositional literal in each F_i.

Because no two of them can from the same F_i as by our construction we do not allow edges between such nodes.

and we do not allow those nodes have completement.

Thus, we can have a truth assignment, that makes the propositional literals corresponding to the node true, and these will make each F_i true, thus making F true and hence F would be satisfiable.

Summary

1. Show k-clique is in NP
2. Show k-clique is NPC — give a polynomial reduction of a known NPC problem to k-clique.

For 1: Suppose a graph has n vertices, give a set of k-vertices, we need to check if each pair of those vertices are connected by an edge. This can be done in $O(k^2)$

$\because k < n \Rightarrow$ Can be done in $O(n^2)$, Thus can be done in polynomial time $\Rightarrow$ k clique is in NP.

For 2: We need to give a polynomial reduction of 3-SAT to k-clique. and show that G has a k-clique $\Leftrightarrow$ F is satisfiable. (F is 3-SAT)

Construction 3-SAT to k-clique:

Let F be a 3-SAT formula.

Let $C_1, C_2 \dots C_k$ be the clauses in F

Let $x_{j,1}, x_{j2}, x_{j3}$ be the literals of C_j.

1. For each literal $x_{j,q}$ we create a distinct vertex in G representing it.
2. G contains all edges, expect
   A. vertices in same clause.
   B. vertices are complements.

k-clique is NPC

Show if G has a k-clique ⇒ F is satisfiable

If G has a k-clique,

1. the k-clique must a vertex from each clause.
2. also, no vertex will be the negation of the others in the clique.

Thus, by setting the corresponding literal to True, F will be satisfied.

Show if F is satisfiable ⇒ G has a k-clique

If F is satisfiable, at least a literal in each clause is set to true in the satisfying assignment.

So, the corresponding vertices forms a clique, Thus G has a k-clique

Finally, since G can be constructed from F in polynomial time, so we have a polynomial time reduction from 3SAT to k clique.

Write TM

E1: $\{a^n b^n \mid n \geq 1\}$

题型 $a^n b^n c^n d^n$ ... key: $n \geq 0$ means TM should accept ⊔ (empty)

Basic idea:

Match a's with b's:

Repeat:

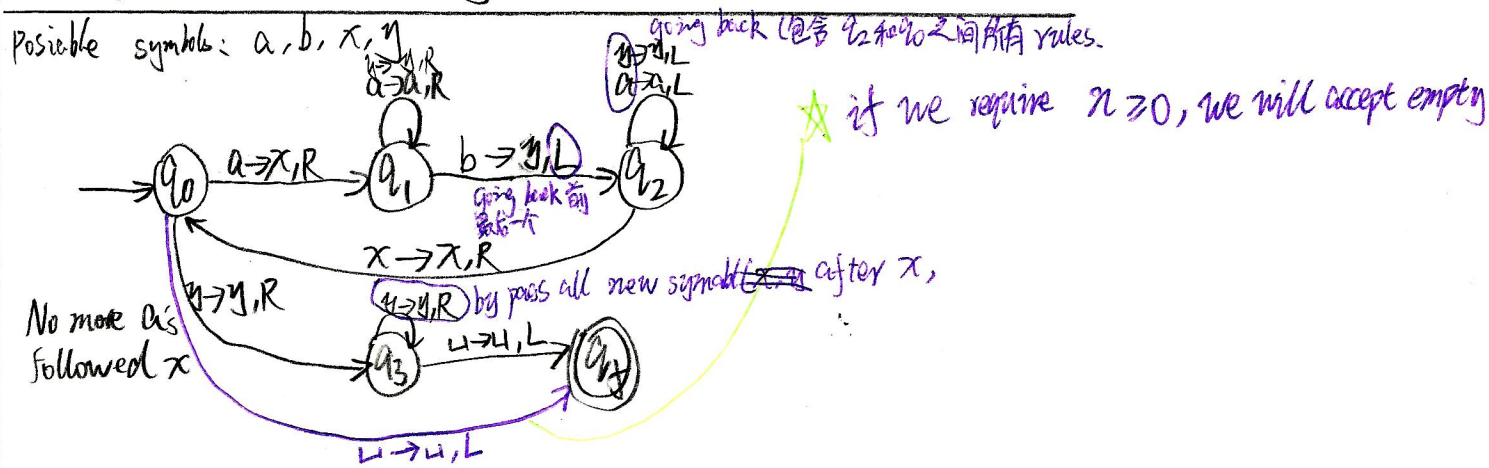
replace leftmost a with x

find leftmost b and replace it with y

Until there are no more a's or b's

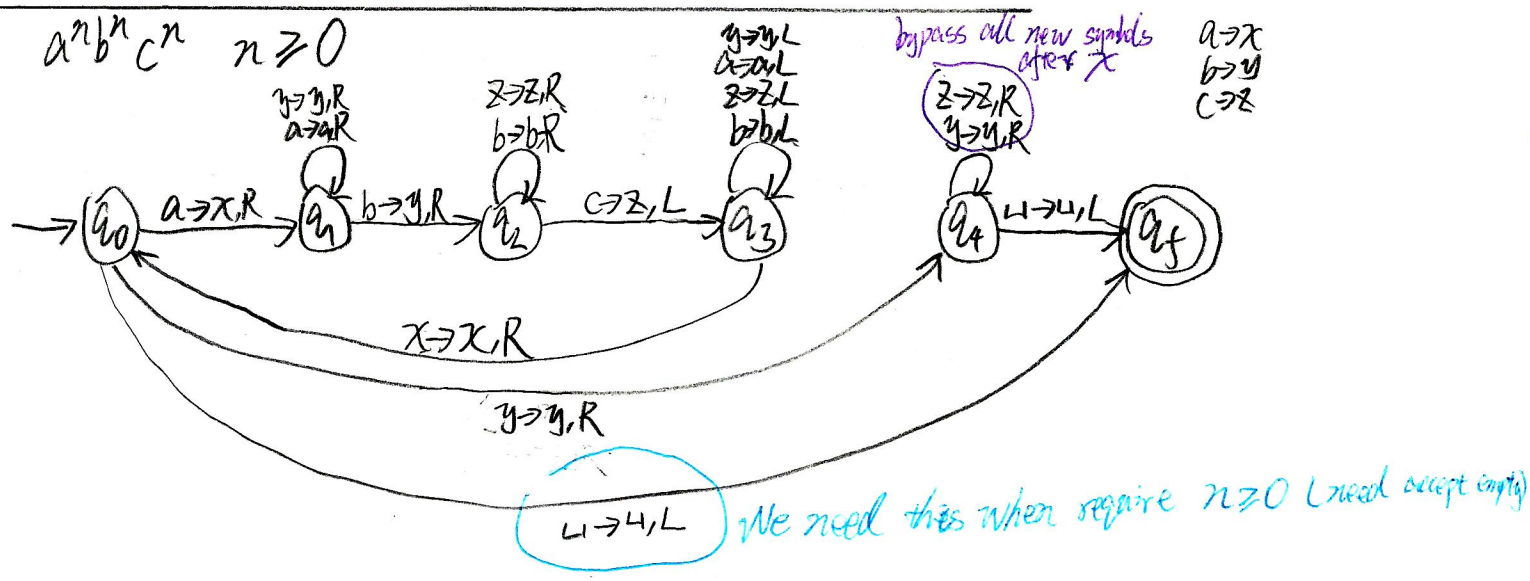
If there is a remaining a or b, reject.

Possible symbols: a, b, x, y



☆ if we require $n \geq 0$, we will accept empty

E2: $a^n b^n c^n$ $n \geq 0$



We need this when require $n \geq 0$ (need accept empty)

Turing Machines and Decidability

Notations.

$M(x)\uparrow$: Turing machine M on input x does not stop (or halt) in a final state.

$M(x)\downarrow$: TM on input x stops (or halt) in a final state.

Step 1: define $\boxed{\begin{aligned} f_3(x) &= 1 \text{ if } M_x(x)\uparrow \\ &= \text{undefined if } M_x(x)\downarrow \end{aligned}}$

Proof: ①. Suppose there is a TM M_i that can compute f_3

②. Compare $f_3(i)$ with $M_i(i)$, But, $f_3(i) \neq M_i(i) \Rightarrow M_i$ can not compute f_3.

Step 2: define $\boxed{\begin{aligned} f_2(x) &= 1 \text{ if } M_x(x)\uparrow \\ &= 0 \text{ if } M_x(x)\downarrow \end{aligned}}$

Proof: ①. Suppose there is a TM M_x that compute $f_2 \Rightarrow M_x$ also can compute f_3 as follow:

input (x)
if $M_x(x)=0$ then output 1
if $M_x(x)=1$ then go to infinate loop.

② But we know that f_3 is not computable
That means our assumption that f_2 is computable is wrong.
Hence, f_2 is not computable.

Step 3: $H_1 = \{x \mid M_x(x) \text{ halt}\}$ is not computable.

Proof: ①: Suppose it is computable, then there is a TM M that can compute it.
Using TM M we can compute f_2 as follow:

Input (x)
If $M(x)=1$ then output 0
If $M(x)=0$ then output 1

②. We know f_2 is not computable, so our assumption is wrong
$\Rightarrow \{x \mid M_x(x) \text{ halt}\}$ is not halts.

Step 4: $H = \{(i,x) \mid M_i(x) \text{ halts}\}$ is not recursive.

Proof: ①: Suppose H is computable, then there is a TM M for it
Using TM M we can compute f_2

Input (x)
If $M(x,x)=1$ then output 0
If $M(x,x)=0$ then output 1

② We know f_2 is not computable, so our assumption is wrong
$\Rightarrow \{(i,x) \mid M_i(x) \text{ halts}\}$ is not halts.

Turing Machine and Decidability

Summary

停机问题

$H_1 = \{x \mid M_x(x) \text{ halt}\}$ Question is will H_1 halt (computable?)?

TM x input is also x, output is also x, TM x 的输入就是自身

$H = \{(i, x) \mid M_i(x) \text{ halt}\}$

只要输出中含有输入的东西，问是不是能算，都是要证明不能算。

key and basic idea:

Suppose F_3 is undeciable / uncomputable

To show F_2 is undeciable, we just need $F_3[\text{call}(\cdot F_2]$ (F_2 可用来算 F_3)

---

key step:

①. define a undeciable function.

$f_3(x) = 1$ if $M_x(x)\uparrow$

$=$ undefined if $M_x(x)\downarrow$

Proof: Suppose there is a TM M_i that can compute f_3.

The input of M_i is another TM, also could be itself.

If M_i could not halt $\Rightarrow$ M_i input M_i, output 1, but if M_i could output 1, which means M_i could halt $\Rightarrow$ f_3 and M_i are differ on input i $\Rightarrow$ f_3 is not computable.

②. define $f_2(x) = 1$ if $M_x(x)\uparrow$

$= 0$ if $M_x(x)\downarrow$

Suppose there is a TM M_x that can compute f_2, $\Rightarrow$ M_x also can compute f_3 as follow

Input (x)
- if $M(x) = 0$ then output 1
- if $M(x) = 1$ then go to infinate loop

But we know f_3 is not computable $\Rightarrow$ f_2 is also not computable.

③. Final case.

$H_1 = \{x \mid M_x(x) \text{ halt}\}$ is not computable.

Proof: Suppose it is computable, then there is a TM M that can compute it.

Using TM M we can compute f_2 as follow

Input (x)
- if $M(x) = 1$ then output 0
- $= 0$ then output 0

But we know f_2 is not computable $\Rightarrow$ $\{x \mid M_x(x) \text{ halt}\}$ is not halt.

Decidability — Example.

function from $N \to \{2,1\}$ is not <u>decidable</u>.
这类题就是让把应个 function 定义成 not decidable.

$$f(N) \to \{2,1\}$$

<u>if $f(N)$ is computable</u>, then there must a TM M could compute $f(N)$

① define
$$f_3(x) = 1 \text{ if } M_x(x)\uparrow$$
$$= \text{undefined if } M_x(x)\downarrow$$

Proof: Suppose there is a M_i that can compute f_3, but f_3 and M_i are differ when input i,
So f_3 is not computable.

②. define
$$f_2(x) = 1 \text{ if } M_x(x)\uparrow$$
$$= 2 \text{ if } M_x(x)\downarrow$$

Suppose there is a TM M_x that can compute $f_2 \Rightarrow M_x$ also can compute f_3 as follow:

if input (x)
  if $M_x(x) = 1$ then go to infinate loop
  $= 2$ then output 1

But we know that f_3 is not computable $\Rightarrow f_2$ is not computable.

③. $f(N) \to \{2,1\}$
Suppose there is a TM M can compute $f(N) \Rightarrow M$ can be also used to compute f_2.
as follow:

Input (x)
  if $M(x) = 1$ then output 2
  $M(x) = 2$ then output 1

But we know f_2 is not computable $\Rightarrow f(N)$ is not computable.

CSE 355 – Introduction to theory of computation

Quiz 19

April 11th 2013; Take home SOLUTION

Q1: Write a Turing Machine that accepts the language $\{a^n b^n c^n \mid n \geq 0\}$.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, q_{reject})$$

Q : States
Σ : Input alphabet
Γ : Tape alphabet
δ : Transition Function
q_0 : Initial state
q_f : Final state
q_{reject} : reject state.

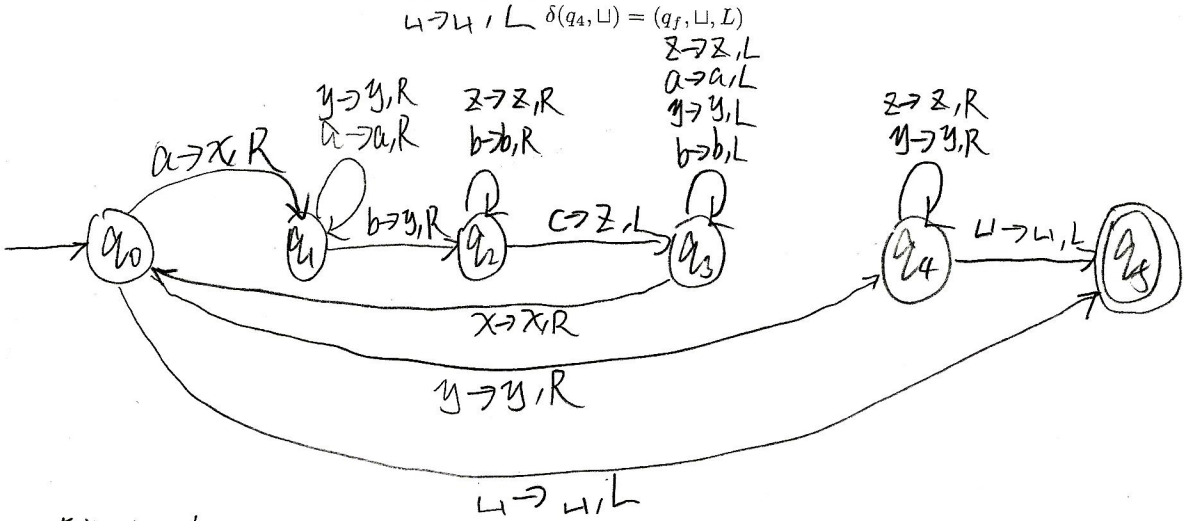
$a \to x, R$ $\quad \delta(q_0, a) = (q_1, x, R)$
$y \to y, R$ $\quad \delta(q_0, y) = (q_4, y, R)$
$\sqcup \to \sqcup\ L$ $\quad \delta(q_0, \sqcup) = (q_f, \sqcup, L)$

$a \to a, R$ $\quad \delta(q_1, a) = (q_1, a, R)$
$y \to y, R$ $\quad \delta(q_1, y) = (q_1, y, R)$
$b \to y, R$ $\quad \delta(q_1, b) = (q_2, y, R)$

$b \to b, R$ $\quad \delta(q_2, b) = (q_2, b, R)$
$c \to z, L$ $\quad \delta(q_2, c) = (q_3, z, L)$
$z \to z, R$ $\quad \delta(q_2, z) = (q_2, z, R)$

$b \to b, L$ $\quad \delta(q_3, b) = (q_3, b, L)$
$y \to y, L$ $\quad \delta(q_3, y) = (q_3, y, L)$
$a \to a, L$ $\quad \delta(q_3, a) = (q_3, a, L)$
$z \to z, L$ $\quad \delta(q_3, z) = (q_3, z, L)$
$x \to x, R$ $\quad \delta(q_3, x) = (q_0, x, R)$

$y \to y, R$ $\quad \delta(q_4, y) = (q_4, y, R)$
$z \to z, R$ $\quad \delta(q_4, z) = (q_4, z, R)$
$\sqcup \to \sqcup, L$ $\quad \delta(q_4, \sqcup) = (q_f, \sqcup, L)$



Idea:

每次replace一个

每次 replace leftmost a with x, b with y, c with z

Until there are no more a's, b's or c's.

If there is a remaining a, b or c reject.