Ch. 4 – Properties of Regular & Non-Regular Languages

§1. Right-linear Grammars & NFAs

Def. Let $\mathcal{L}(REX)$ be the class of all languages that can be described by regular expressions, $\mathcal{L}(RLG)$ those that can be generated by right-linear grammars, and $\mathcal{L}(NFA)$ (resp. $\mathcal{L}(DFA)$) those that can be recognized by NFAs (resp. DFAs).

Our main aim in this chapter is to first show that $\mathcal{L}(RLG) = \mathcal{L}(NFA) = \mathcal{L}(OAS\text{-}NFA) = \mathcal{L}(REX)$.

$$\mathcal{L}(NFA) = \mathcal{L}(DFA)$$

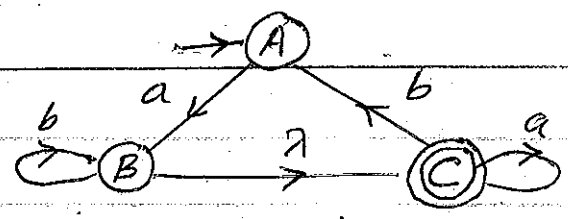
We have already seen that $\mathcal{L}(DFA) = \mathcal{L}(NFA)$ because every DFA is a special NFA, so $\mathcal{L}(DFA) \subseteq \mathcal{L}(NFA)$; and every NFA is equivalent to a DFA, so $\mathcal{L}(NFA) \subseteq \mathcal{L}(DFA)$.

Prop. 1 Let $M = \langle Q, T, \Delta, q_0, \mathcal{A} \rangle$ be an NFA. Then we can find an RLG $G = \langle V, T, \{S\}, \mathcal{P} \rangle$ such that $L(G) = L(M)$.

Proof: Let $V = Q$ and $S = q_0$. The productions in $\mathcal{P}$ are constructed from M as follows.
For each $B \in \mathcal{A}(M)$ in M, we get the production $B \to \lambda$.
Also each transition $A \xrightarrow{a} B$ or $A \xrightarrow{\lambda} B$ in M, we get the production $A \to aB$ or resp. $A \to B$.

From the construction of G we can see that $w \in L(M)$ $\Leftrightarrow$ w can lead you from q_0 to an accepting state in M $\Leftrightarrow$ there is a derivation of w, from S, in G $\Leftrightarrow$ $w \in L(G)$. So $L(G) = L(M)$.

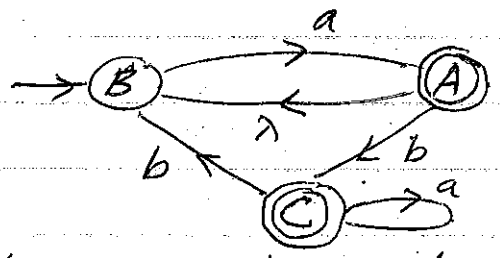
Ex. 1a Let M be the NFA shown on the right.



Then an equivalent RLG G will be as shown below.

$\rightarrow A,\quad A \rightarrow aB,\ B \rightarrow bB,\ B \rightarrow C,\ C \rightarrow aC,\ C \rightarrow bA,\ C \rightarrow \lambda.$

Let us consider the string $\varphi = abba$. In M, φ is accepted as follows: $\rightarrow A \xrightarrow{a} B \xrightarrow{b} B \xrightarrow{b} B \xrightarrow{\lambda} C \xrightarrow{a} C$.
In G, φ can be derived from A as follows.
$\rightarrow A \Rightarrow aB \Rightarrow abB \Rightarrow abbB \Rightarrow abbC \Rightarrow abbaC \Rightarrow abba.$

Ex. 1b Let $M =$ 

. Then an equiv. RLG will be as shown below.

$\rightarrow B,\quad B \rightarrow aA,\ A \rightarrow \lambda,\ A \rightarrow B,\ A \rightarrow bC,\ C \rightarrow \lambda,\ C \rightarrow bB,\ C \rightarrow aC.$

Notice that the number of productions in G will be the number of transitions of M plus the number of states in $\mathcal{A}(M)$.

Prop. 2. Let $G = \langle V, T, \{S\}, P\rangle$ be an RLG. Then we can find an NFA $M = \langle Q, T, \Delta, q_0, \mathcal{A}\rangle$ such that $L(M) = L(G)$.

Proof: Let $q_0 = S$. We will construct Q, $\mathcal{A}$ and Δ from the productions of G as follows. Q will start out being V – but more states will be added as we go along.

(a) For each production in G of the form $B \to \lambda$, let $B \in \mathcal{A}(M)$.

(b) For each production of the form $A \to B$ in G, add the transition $A \xrightarrow{\lambda} B$ to M

(c) For each production in G of the form $A \to a_1 \cdots a_n B$, make $n-1$ new states and add the transitions

$$A \xrightarrow{a_1} \bigcirc \xrightarrow{a_2} \bigcirc \xrightarrow{a_3} \cdots \bigcirc \xrightarrow{a_n} B \quad \text{to } M.$$

(d) Finally make a specially designated accepting state Z in $\mathcal{A}(M)$ and for each production in G of the form $A \to a_1 a_2 \cdots a_k$ make $(k-1)$ new states & add the transitions

$$A \xrightarrow{a_1} \bigcirc \xrightarrow{a_2} \cdots \bigcirc \xrightarrow{a_k} Z \quad \text{to } M.$$

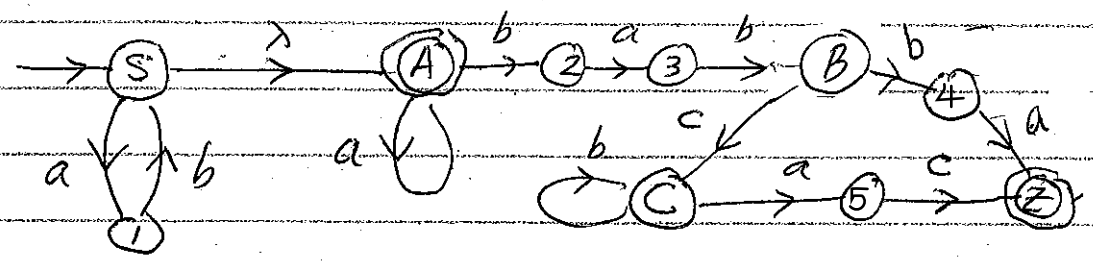
From the construction of M, we can see that $L(M) = L(G)$ because each derivation in G of a string φ will correspond to an acceptance track of φ in M.

Ex.2 Let G be the RLG shown below.

$S \to abS$, $S \to A$, $A \to babB$, $A \to aA$, $A \to \lambda$

$B \to cC$, $B \to ba$, $C \to ac$, $C \to bC$

Then an equivalent NFA M will be as below.



The derivation $S \Rightarrow abS \Rightarrow ab\lambda A \Rightarrow ab\lambda aA \Rightarrow ab\lambda a\lambda$ in G corresponds to the acceptance track

$\to S \xrightarrow{a} 1 \xrightarrow{b} S \xrightarrow{\lambda} A \xrightarrow{a} A$ in M.

Def. Let $M = \langle Q, T, \Delta, q_0, \mathcal{A} \rangle$ be an NFA and φ & ψ be strings from T^*. Recall that the <u>reaching set</u>

$\mathcal{R}(\varphi)$ = set of all states that can be reached via φ when starting at q_0.

We define the <u>back-tracking set</u> $\mathcal{B}(\psi)$ by

$\mathcal{B}(\psi)$ = set of all states from which you can start and reach an accepting state via ψ.

Qu.1 Given an RLG, G, how can we tell if it is ambiguous?

Ans1. First convert the RLG, G, into an equivalent NFA M. Then find all the reaching sets and back-tracking sets of M. G will be ambiguous $\Leftrightarrow$ there is a reaching set $\mathcal{R}(\varphi)$ and a back-tracking set $\mathcal{B}(\psi)$ such that $|\mathcal{R}(\varphi) \cap \mathcal{B}(\psi)| \geq 2$. The string $\varphi\psi$ will have at least two leftmost derivations.

Qu.2 Given an ambiguous RLG, G, how can we find an equivalent un-ambiguous RLG, G'?

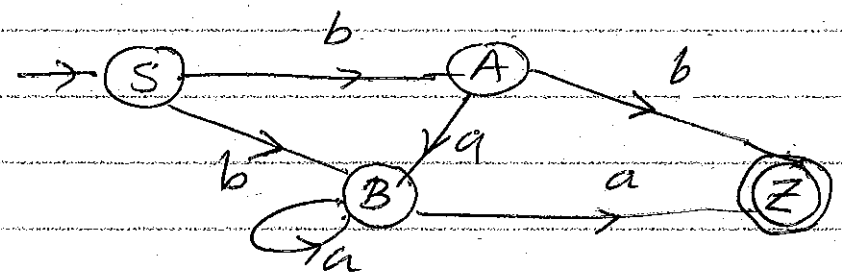
Ans.2 First convert the RLG, G, into an equivalent NFA. Then convert M into an equivalent DFA, M_D. Finally convert M_D into an equivalent RLG, G'. Then G' will be an un-ambiguous RLG which is equivalent to G. The ambiguity was removed in passing from the NFA to the DFA.

Ex.3 (a) Determine whether or not the RLG G below is ambiguous

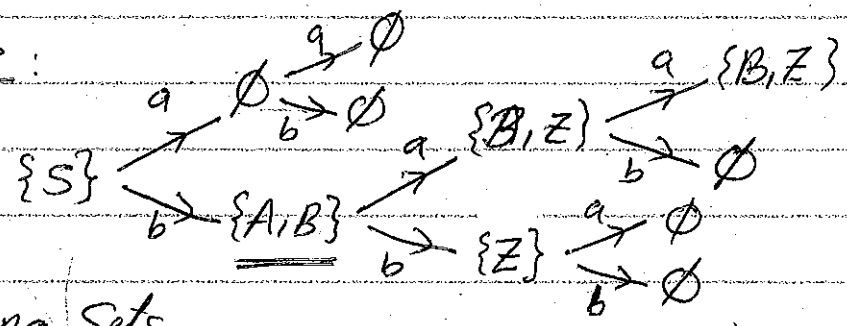
$S \to bA$, $S \to bB$, $A \to aB$, $A \to b$, $A \to aB$, $B \to aB$, $B \to a$.

(b) If G is ambiguous, find an equivalent un-ambiguous RLG, G'.

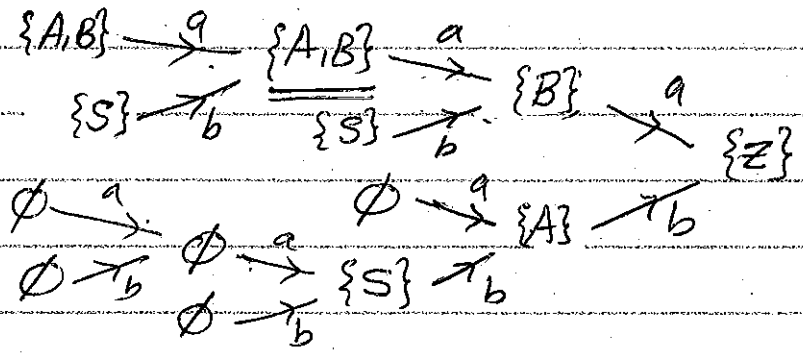
(a) M:



<u>Reaching Sets</u>:



<u>Back-tracking Sets</u>



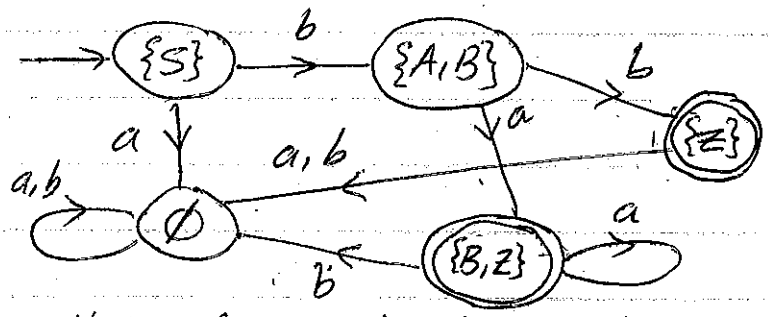
If we take $\varphi = b$ and $\psi = aa$, then we see that $\mathcal{R}(\varphi) = \{A, B\}$ and $\mathcal{B}(\psi) = \{A, B\}$. Since $|\mathcal{R}(\varphi) \cap \mathcal{B}(\psi)| = |\{A, B\} \cap \{A, B\}| = |\{A, B\}| = 2 \geq 2$, it follows that G is ambiguous. The string $\varphi\psi = baa$ will have at least 2 leftmost derivations in G. Indeed, we have:

$S \Rightarrow bA \Rightarrow baB \Rightarrow baa$ (first leftmost derivation)

& $S \Rightarrow bB \Rightarrow baB \Rightarrow baa$ (2nd leftmost derivation)

(b) We already found the reaching sets of M in part (a). So it is now easy to find the DFA, M_D.

M_D:



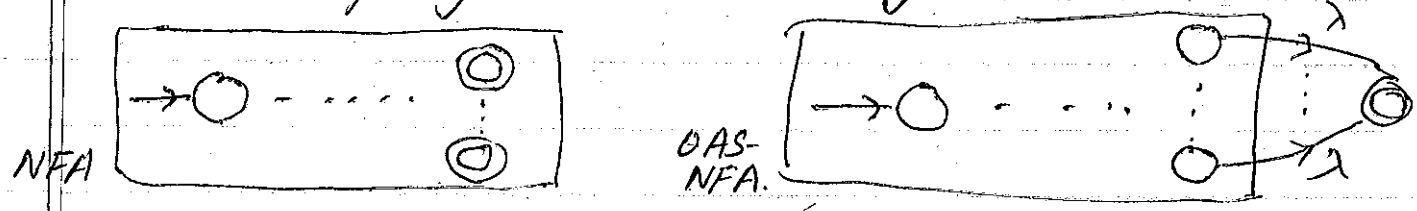
Now all we have to do is to convert M_D into an equivalent RLG, G'. To make this easier to understand, let $C = \{S\}$, $D = \{A,B\}$, $E = \emptyset$, $F = \{B,Z\}$ and $H = \{Z\}$. Then G' is

$\rightarrow C$, $C \rightarrow aE$, $C \rightarrow bD$, $D \rightarrow aF$, $D \rightarrow bH$, $E \rightarrow aE$, $E \rightarrow bE$, $F \rightarrow bE$, $F \rightarrow aF$, $H \rightarrow aE$, $H \rightarrow bE$, $F \rightarrow \lambda$, $H \rightarrow \lambda$; and we are done. Note that the string $\varphi\psi = baa$ has only one leftmost derivation in G' because there is only one path from $\{S\} = C$ to $\{B,Z\} = F$ via baa.

§2. Regular expressions & OAS-NFAs

Definition. An OAS-NFA is an NFA with only one accepting state which is different from the initial state.

It is easy to see that any NFA is equivalent to an OAS-NFA. All we have to do is to create a single new accepting state and add λ-transitions from the old accepting states to the single new one.



<u>Prop. 3</u> If E is a regular expression, then we can find an OAS-NFA M such that $L(M) = L(E)$.

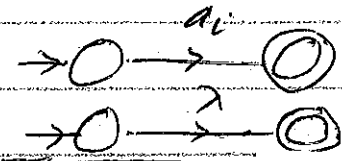
<u>Proof:</u> Recall that a regular expression over $\{a_1, \ldots, a_n\}$ is defined recursively as follows:

(a) $\underline{a_1}, \ldots, \underline{a_n}, \underline{\lambda}$ and $\emptyset$ are regular expressions.

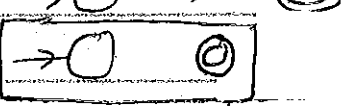
(b) If E_1 & E_2 are regular expressions then so are $(E_1 + E_2)$, $(E_1 . E_2)$, and (E_1^*).

To find an OAS-NFA for a reg. expr. E, it will suffice to show how to make OAS-NFAs for the reg. expr. in (a); and given OAS-NFAs for E_1 & E_2, to show how to make OAS-NFAs for $(E_1 + E_2)$, $(E_1 . E_2)$ and (E_1^*).

(a) OAS-NFA for $\underline{a_i}$:

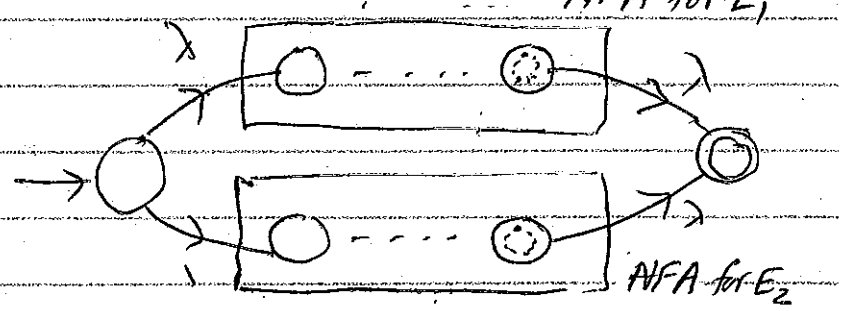


OAS-NFA for $\underline{\lambda}$:

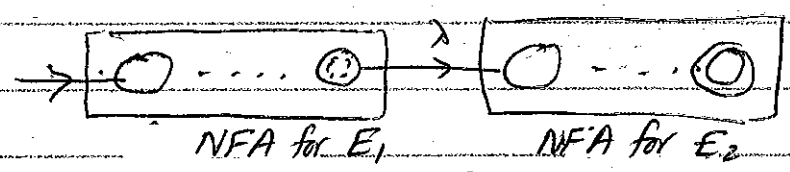


OAS-NFA for $\emptyset$:

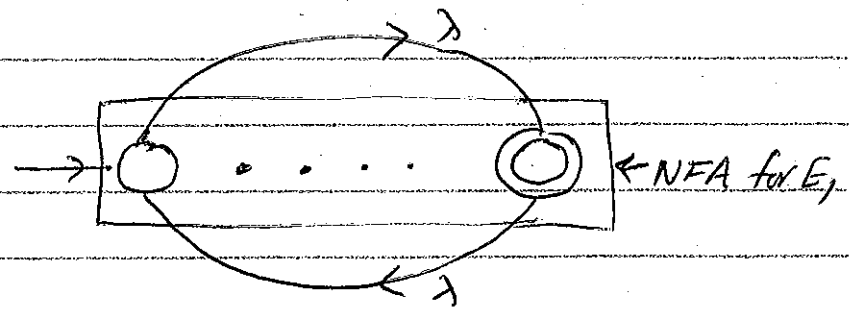
(b) OAS-NFA for $(E_1 + E_2)$:



OAS-NFA for $(E_1 . E_2)$:

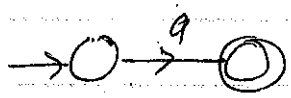


OAS-NFA for (E_1^*):

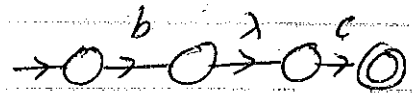


Ex.4 Find an OAS-NFA M such that $L(M) = \underline{a}^* \cdot (\underline{bc} + \underline{ca})^*$

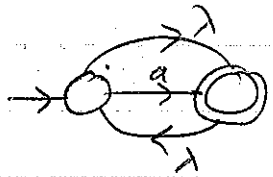
$\underline{a}$:



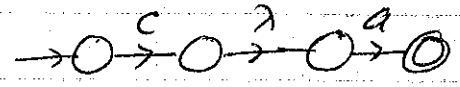
$\underline{bc}$:



$\underline{a}^*$:

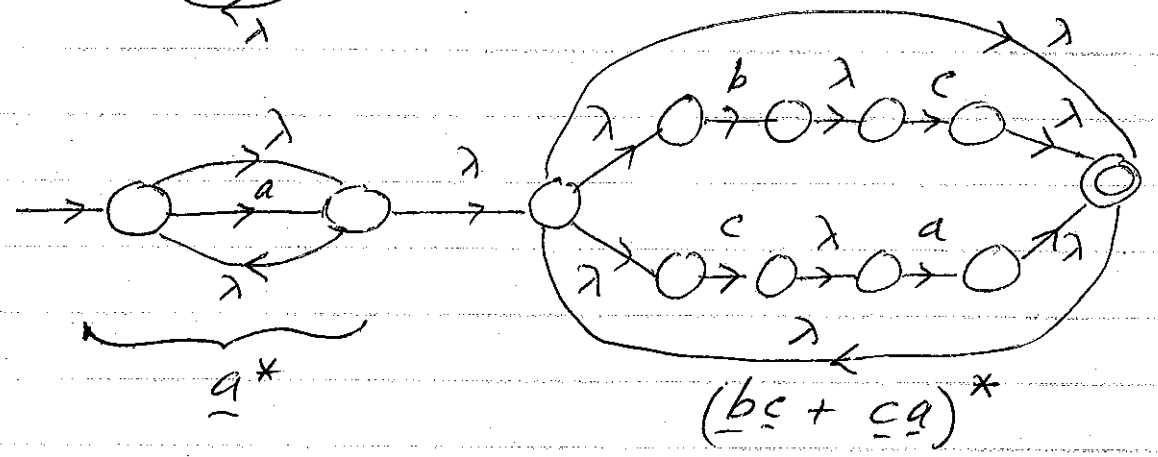


$\underline{ca}$:



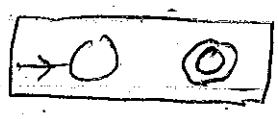
Ans:

M:

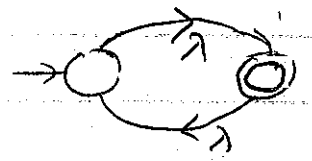


Ex.5 Find an OAS-NFA M such that $L(M) = (\emptyset^*)$

Ans: $\emptyset =$

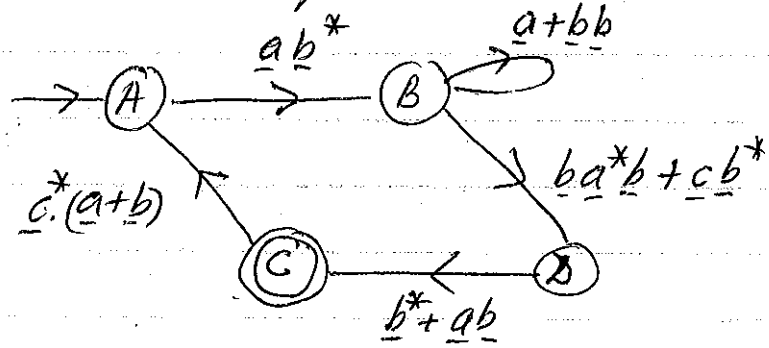


$(\emptyset^*)$:



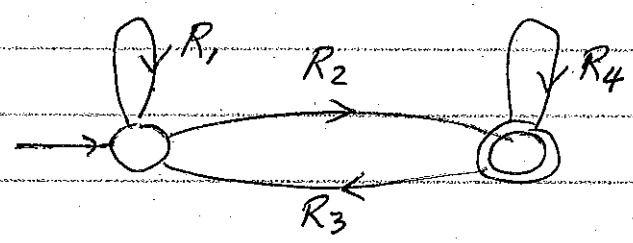
Def A <u>generalized finite acceptor</u> (GFA) is defined in the same manner as an NFA except that instead of having "a" or "λ"– transitions we can have any regular expression serve as a transition.

Ex.6 Below is an example of a GFA



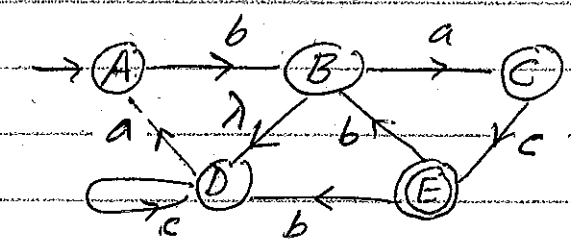
<u>Proposition 4</u>: Let M be an NFA. Then we can find a regular expression E such that $L(M) = E$.

<u>Sketch of Proof</u>: First we convert M into an OAS-NFA M_1. Then we view M_1 as a GFA and remove the other states, besides the initial state & accepting state, one at a time to get a sequence of equivalent simpler & simpler GFAs. At the end, we will get a GFA as shown below with only two states.



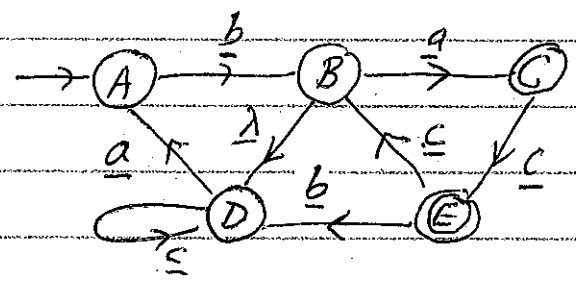
Then $L(M) = R_1^* R_2 (R_4 + R_3 \cdot R_1^* R_2)^*$.

<u>Ex 7</u> Find a regular expression for the language recognized by the NFA M below.



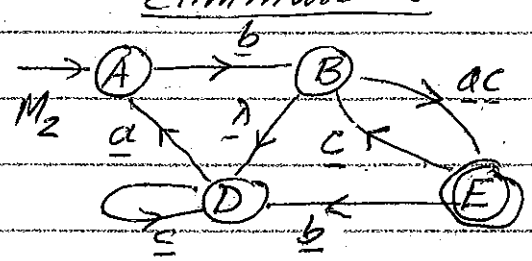
<u>Ans:</u> This NFA is an OAS-NFA, so we don't have to convert it into an OAS-NFA. To view M as a GFA, just add underlines under each transition

GFA M_1



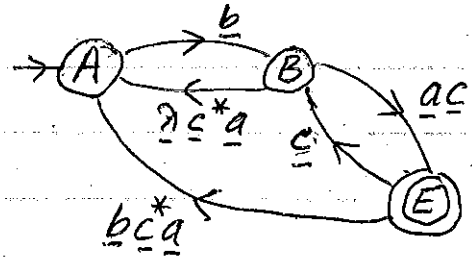
<u>Eliminate C</u>

M_2



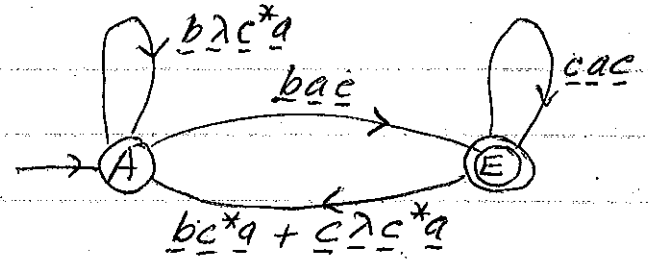
Ex7 Eliminate D

M_3:



Eliminate B

M_4:



So $R_1 = b\lambda c^* a = bc^*a$, $R_4 = cac$

$R_2 = bac$, $R_3 = bc^*a + c\lambda c^*a = (b+c).c^*a$

Hence $L(M) = R_1^* R_2 . (R_4 + R_3 R_1^* R_2)^*$

$= (bc^*a)^* . (bac) . (cac + (b+c) . c^*a . (bc^*a)^* bac)^*$.

Now from Propositions 1, 2, 3 & 4, we can deduce our first main result.

<u>Theorem 5</u> (Main theorem for Regular languages)

$\mathcal{L}(RLG) = \mathcal{L}(NFA) = \mathcal{L}(DFA) = \mathcal{L}(DAS\text{-}NFA) = \mathcal{L}(REX)$

§3. <u>Closure Theorems for regular languages.</u>

We will now explore some closure results about regular languages.

<u>Theorem 6</u>. (Closure theorem for Regular languages)

If L_1 & L_2 are regular languages, then so are also the following

(a) $L_1 \cup L_2$ (b) $L_1 \cdot L_2$ (c) L_1^*

(d) L_1^R (e) L_1^c (f) $L_1 \cap L_2$ (g) $L_1 - L_2$.

<u>Proof</u>: Since L_1 & L_2 are regular languages, then by definition, we can find regular expressions E_1 & E_2 such that $L(E_1) = L_1$ & $L(E_2) = L_2$

(a) Now $(E_1 + E_2)$ is a regular expression which describes $L_1 \cup L_2$. So $L_1 \cup L_2$ is a regular language.

(b) $(E_1 \cdot E_2)$ is a regular expression which describes $L_1 \cdot L_2$. So $L_1 \cdot L_2$ is a regular language.

(c) (E_1^*) is a regular expression which describes L_1^*. So L_1^* is a regular language.

(d) Since L_1 is a regular language, it follows from Theorem 5, the main theorem on reg. lang, that we can find an OAS-NFA M_1 such that $L(M_1) = L_1$. Now if we make the initial state of M_1 into the accepting, make the accept state of M_1 into the initial state, and reverse the direction of each transition, we will get an OAS-NFA M_1^R with $L(M_1^R) = L_1^R$. So by Theorem 5, L_1^R is a regular language.

(e) First convert E_1 into an equivalent NFA M_1. Then con convert M_1 into an equivalent DFA, M_D. Now form the complement machine M_D^c by switching the non-accepting states into accepting states & vice versa. Then convert M_D^c into an equiv. regular expression E^c. We will then have

$$L(E^c) = L(M_D^c) = L(M_D)^c = L(M_1)^c = L(E_1)^c = L_1^c$$

So E^c will be a regular expression which describes L_1^c. Hence L_1^c is a regular language.

(f) We know that $(L_1 \cap L_2)^c = L_1^c \cup L_2^c$ by De Morgan's Law. So $L_1 \cap L_2 = (L_1 \cap L_2)^{cc} = (L_1^c \cup L_2^c)^c$. But L_1^c & L_2^c are regular lang. by part (e). So $(L_1^c \cup L_2^c)$ is regular by part (a). Thus $L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$ will be regular by part (e).

(g) $L_1 - L_2 = L_1 \cap L_2^c$. By part (e), L_2^c is regular; so by part (f) $L_1 - L_2 = L_1 \cap L_2^c$ will a regular language.

Another closure result about regular languages.

Def. Let T_1 and T_2 be alphabets. We say that the function $h: T_1^* \to T_2^*$ is a <u>homomorphism</u> if $h(\alpha\beta) = h(\alpha)h(\beta)$ for any $\alpha, \beta \in T_1^*$.

<u>Fact 7</u>: If $h: T_1^* \to T_2^*$ is a homomorphism, then we must have $h(\lambda) = \lambda$.

<u>Proof</u>: We know that $h(\alpha.\beta) = h(\alpha).h(\beta)$ for any $\alpha, \beta \in T_1^*$. So in particular since $\lambda = \lambda.\lambda$, we get $h(\lambda) = h(\lambda.\lambda) = h(\lambda).h(\lambda)$. From this it follows that $h(\lambda) = \lambda$ – because if $h(\lambda) \neq \lambda$, then we cannot have $h(\lambda) = h(\lambda).h(\lambda)$.

It can also be shown that h is completely determined by $h(c_1), h(c_2), \ldots, h(c_n)$ where $T_1 = \{c_1, c_2, \ldots, c_n\}$. Now if E is a regular expression over T_1, then $h(E)$ can be defined as the same expression E except that each character $\underline{c_i}$ is replaced by $\underline{h(c_i)}$. For example, let $T_1 = \{a, b, c\}$ & $T_2 = \{0, 1\}$. Put $h(a) = 10$, $h(b) = 11$ & $h(c) = 00$. If $E = \underline{a}^*.(\underline{b} + \underline{ac})^*$, then we will have that $h(E) = (\underline{10})^*.(\underline{11} + \underline{1000})^*$.

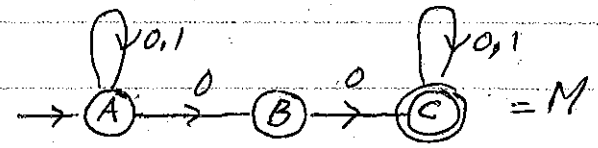
<u>Prop. 8</u>: If L is a regular language on T_1 and $h: T_1^* \to T_2^*$ is a homo-morphism, then $h[L] = \{h(\varphi) : \varphi \in L\}$ is also a regular language.

<u>Proof</u>: If E is a reg. expr. for L, then $h(E)$ is a reg. expr. for $h[L]$.

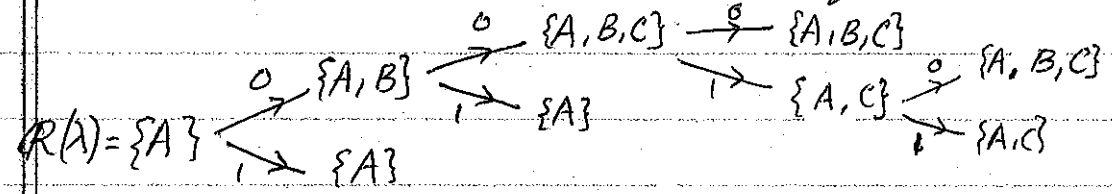
Ex 8 Let $L = \{\varphi \in \{0,1\}^* : \varphi \text{ contains the string } 00\}$. Find a regular expressions which describes L and a reg. expression which describes $L^c = \{\varphi : \varphi \text{ does not contain } 00\}$

(a) $E = (0+1)^*.00.(0+1)^*$ is a regular expression which describes L.

(b) First we convert E into an equiv. NFA, M.

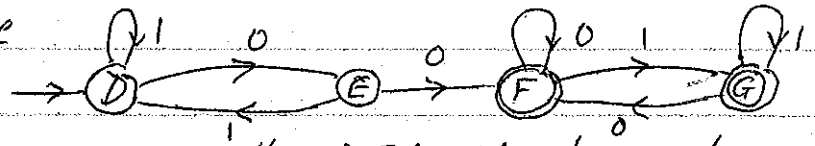


(c) Then we convert M into an equiv. DFA, M_D.



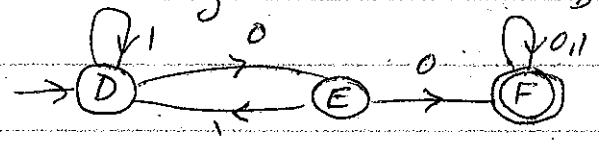
Let $D = \{A\}$, $E = \{A,B\}$, $F = \{A,B,C\}$ and $G = \{A,C\}$

Then M_D will be



We can also minimize the DFA M_D to get

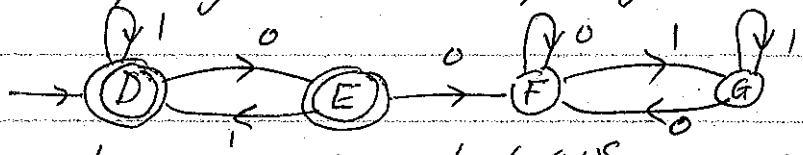
$(M_D)_{min} =$



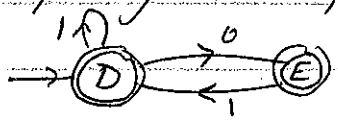
because G is indistinguishable from F.

But we don't need to do this.

(d) Now switch the accepting & non-accepting states to get $(M_D)^c =$

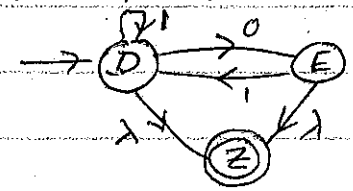


(e) Then find the regular expression which $(M_D)^c$ recognizes. Since F & G are non-accepting states, we only need to look at the NFA

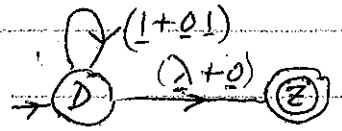


. So convert this NFA into an OAS-NFA

(f)



. Now get rid of E to obtain the GFA



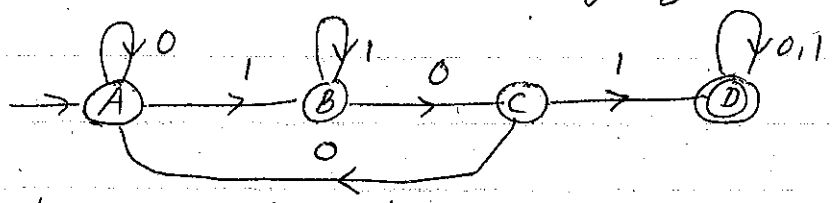
(g) Final Answer is: $E^c = (1+01)^*.(\lambda+0)$.

Ex. 9 Let $L = \{\varphi \in \{0,1\}^* : \varphi \text{ contains the string } 101\}$. Find regular expressions which describe L and L^c

(a) $(\underline{0}+\underline{1})^* . \underline{101} . (\underline{0}+\underline{1})^*$ is a regular expression which describes L.

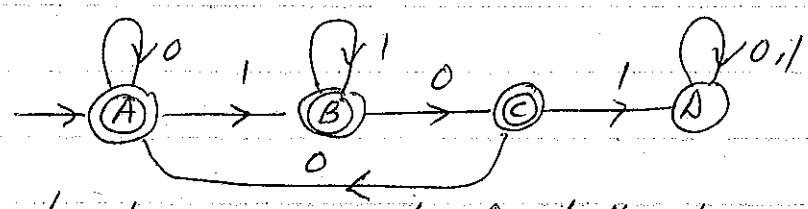
(b) To find a regular expression which describes L^c, we first find a DFA M which recognizes L.

$M =$



We did this because it takes too many steps to go through an NFA.

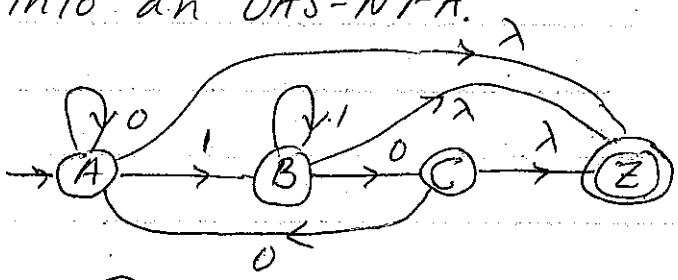
(c) Now $M^c =$



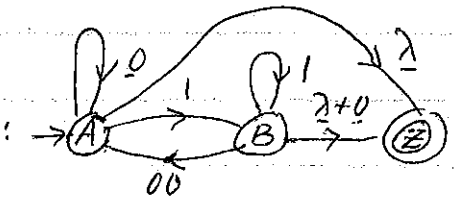
So all we have to do now is to find the language which M^c recognizes (as a regular expression).

(d) We can ignore D and convert the rest of M^c into an OAS-NFA.

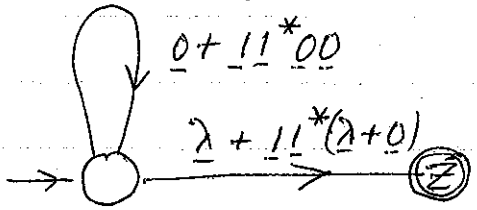
M_1:



M_2:



M_3:



$$E^c = (\underline{0}+\underline{11}^*\underline{00})^* . (\underline{\lambda}+\underline{11}^*(\underline{\lambda}+\underline{0}))$$
$$= (\underline{0}+\underline{11}^*\underline{00})^* (\underline{\lambda}+\underline{11}^*+\underline{11}^*\underline{0}).$$

Note that $L^c = \{\varphi \in \{0,1\}^* : \varphi \text{ does not contain } 101 \text{ as a substring}\}$.

(15) §4. Decidability problems involving Regular Languages

Def. A decision problem is a problem with unknown parameters which can take denumerablymany values, such that for all values of the parameters, we have a YES or NO answer.

Ex. 10 (Primality Decision Problem). The problem is: Is n a prime number? Here n is the unknown parameter and for each value of n, the answer is either YES or NO. The values the parameter n can take are $\{0, 1, 2, 3, \ldots\} = \mathbb{N}$.

Def. A decision problem is algorithmically decidable if there is a fixed algorithm which takes as input the possible values of the unknown parameters and produces the correct answer.

Ex. 11 The Primality Decision Problem is algorithmically decidable because we can find an algorithm which can decide whether or not n is prime. Just take n as input and try to see if any integer between 1 & $\sqrt{n}$ divides n. If none of the integers from 2 to $\lfloor \sqrt{n} \rfloor$ divides n, then n is prime. Otherwise n is not prime.

Theorem 9 (Decidability Theorem for Reg. Lang.) Let L_1 & L_2 be regular languages. Then it is algorithmically decidable whether or not

(a) $L_1 = \emptyset$ (b) $L_1 \subseteq L_2$ (c) $L_1 = L_2$ (d) L_1 is infinite.

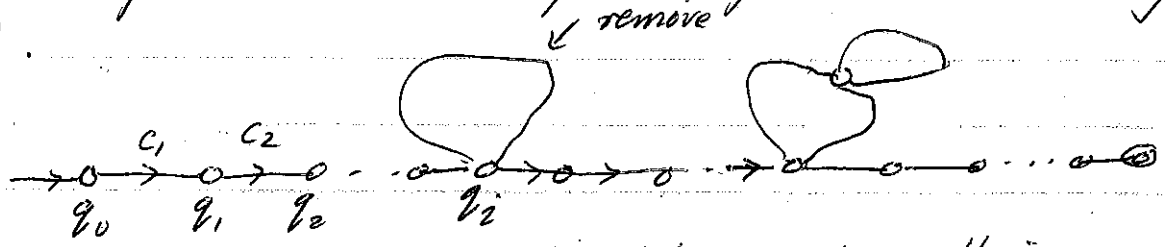
Before we prove Theorem 9, let us first point out that the parameters are a pair of regular languages, L_1 & L_2. There are only denumerably many values of these two parameters because the number of regular languages is denumerable. We need two Lemmas to help us in Theorem 9.

Def. An NFA is said to be <u>λ-free</u> if there are no λ-transitions that can take you from one state to a different state.

<u>Lemma 10</u>: Let M be a λ-free NFA with N states. Then $L(M) \neq \emptyset \Leftrightarrow M$ accepts a string φ_0 of length $\leq N-1$.

<u>Proof</u>: ($\Leftarrow$): Suppose M accepts a string φ_0 of length $\leq N-1$. Then $L(M) \neq \emptyset$ because $\varphi_0 \in L(M)$.

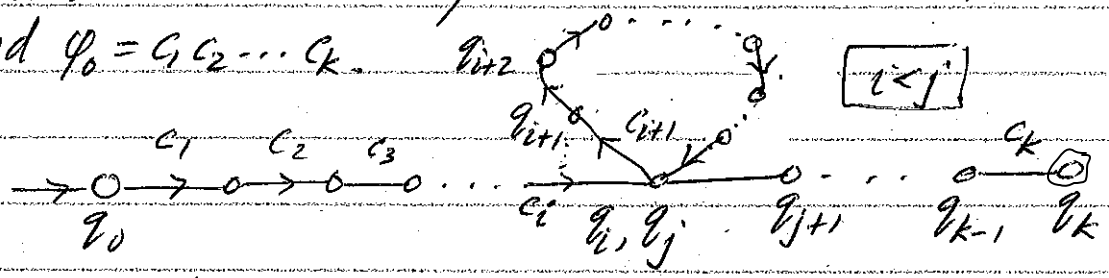
($\Rightarrow$) Now suppose $L(M) \neq \emptyset$. Then we can find a string φ_1 such that M accepts φ_1. Consider any acceptance track of φ_1. If we remove all the loops in this acceptance track of φ_1, we will end up with an acceptance <u>path</u> of a string φ_0.



Since the length of the longest possible path in a graph with N states is $N-1$, we must have $|\varphi_0| \leq N-1$. So M accepts a string φ of length $\leq N-1$.

<u>Lemma 11</u>: Let M be a λ-free NFA with N states. Then $L(M)$ is infinite $\Leftrightarrow$ M accepts a string φ_0 with $N \leq |\varphi_0| \leq 2N-1$.

<u>Proof</u>: ($\Leftarrow$) Suppose M accepts a string φ_0 with $N \leq |\varphi_0| \leq 2N-1$. Now consider any acceptance track of φ_0 in M. Since it takes $|\varphi_0|+1$ states to process φ_0 and $|\varphi_0| \geq N$, this acceptance track must have a loop as shown below. Let $k = |\varphi_0|$ and $\varphi_0 = c_1 c_2 \cdots c_k$.



If we let $x = c_1 c_2 \cdots c_i$, $y = c_{i+1} \cdots c_j$ & $z = c_{j+1} \cdots c_k$ then we can see that for each $r \geq 0$, xy^rz will be accepted by M. Since $|y| = j-i \geq 1$, it follows that all these strings xy^rz will be different from each other. So $L(M)$ will be infinite.

($\Rightarrow$) Suppose $L(M)$ is infinite. Then M must accept a string φ_1 with $|\varphi_1| \geq N$ — because the total no. of strings with length $\leq N-1$ is finite. Now if $|\varphi_1| \leq 2N-1$, then we take φ_0 to be φ_1 and M will accept a string φ_0 with $N \leq |\varphi_0| \leq 2N-1$. If $|\varphi_1| > 2N-1$, then consider any acceptance track of φ_1. If we delete the loops in this acceptance track of φ_1, one at a time, we will eventually get a string φ_0 which is accepted by M and which satisfies $N \leq |\varphi_0| \leq 2N-1$. So M will accept a string φ_0 with $N \leq |\varphi_0| \leq 2N-1$.

Proof of Theorem 9

(a) Since L_1 is a regular language, it follows from Theorem 5; that we can find a DFA (or λ-free NFA) M_1 such that $L(M_1) = L_1$. Now from Lemma 10, we know that $L(M_1) \neq \emptyset \Leftrightarrow M_1$ accepts a string φ_0 of length $\leq N-1$. So check all paths of length $\leq N-1$ from q_0 in M. If one of these paths leads to an accepting state, then $L(M_1) = L_1 \neq \emptyset$. If none of these paths lead to an accepting state, then $L_1 = \emptyset$. So we have an algorithm which can decide whether or not $L_1 = \emptyset$.

(b) First observe that $L_1 \subseteq L_2 \Leftrightarrow L_1 - L_2 = \emptyset$. Since $L_1 - L_2$ is a regular language by Theorem 6(f), it follows that we can use the algorithm in part (a) to check whether or $L_1 - L_2 = \emptyset$. So we again have an algorithm to decide whether or not $L_1 \subseteq L_2$.

(c) Observe that $L_1 = L_2 \Leftrightarrow L_1 \subseteq L_2$ & $L_2 \subseteq L_1$. So use the algorithm in part (b) twice to check if both $L_1 \subseteq L_2$ & $L_2 \subseteq L_1$. So this gives us an algorithm to check whether or not $L_1 = L_2$.

(d) First find a λ-free NFA M_1 such that $L(M_1) = L_1$. Now $L(M_1)$ is infinite $\Leftrightarrow M_1$ accepts a string φ_0 with $N \leq |\varphi_0| \leq 2N-1$. So check all possible acceptance tracks in M with lengths between N and $2N-1$. If one track works, then $L(M_1) = L_1$ will be infinite. If none of the tracks works, then $L(M_1) = L_1$ will not be infinite.

§5. Non-regular languages & their properties.

Prop. 12: The set of all regular languages on $\{a,b\}$ is countable and consequently non-regular languages exist.

Proof: We know that a regular language on $\{a,b\}$ is one that can be described by a regular expression on $\{a,b\}$. Now a reg. expr. on $\{a,b\}$ is just a string based on the alphabet $\{\underline{a}, \underline{b}, \underline{\lambda}, \underline{\emptyset}, +, \cdot, *, (,)\} = T$. Since the set of all strings based on T is countable, it follows that REX, the set of all regular expressions on $\{a,b\}$ is also countable – because $REX \subseteq T^*$. Since we only have a countable number of reg. expr., the total no. of regular languages must also be countable.

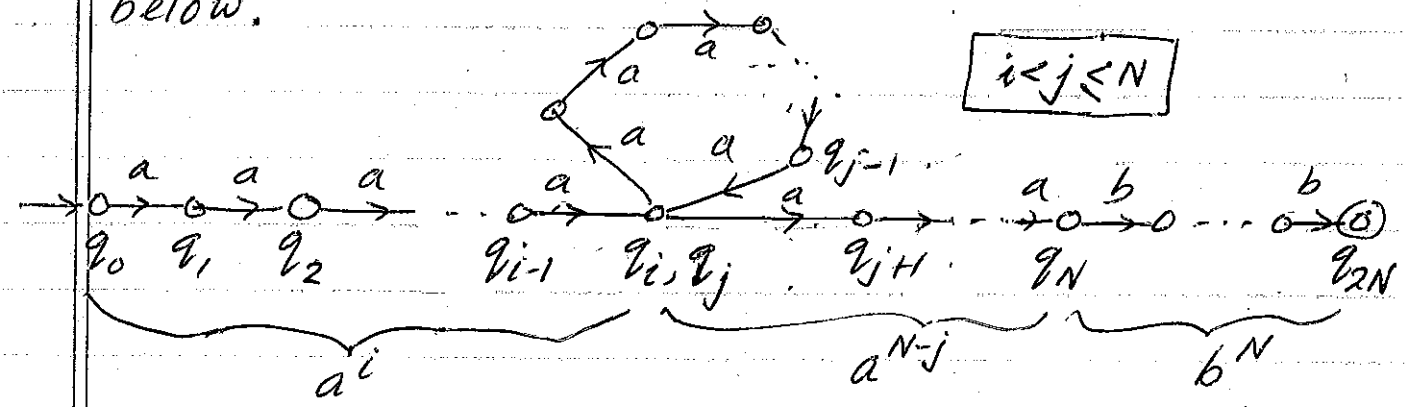
Finally the set of all languages on $\{a,b\}$ is uncountable, so there must exist languages that are not regular. These language are naturally called non-regular.

We will now give several concrete examples of non-regular languages.

Ex. 1 $L_1 = \{a^k b^k : k \geq 0\}$ is a non-regular language

Proof: Suppose L_1 is regular. Then we can find a λ-free NFA such that $L(M_1) = L_1$ by Theorem 5. Let N = number of states in M_1 and consider the string $a^N b^N$. Since $a^N b^N \in L_1$, M_1 will accept

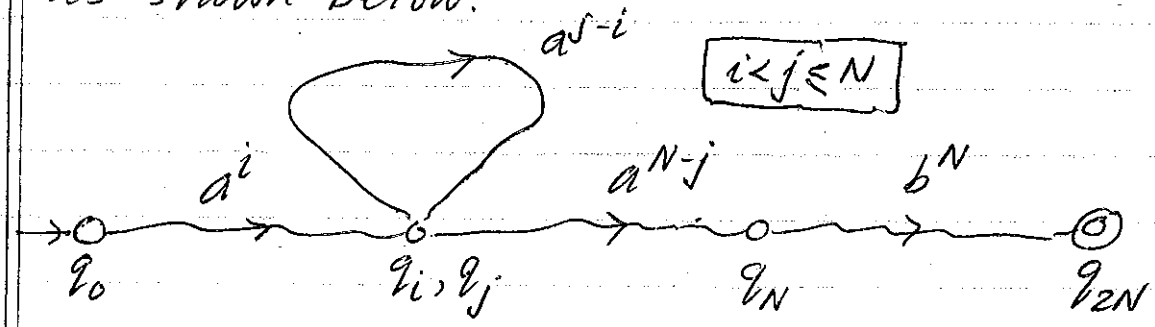
$a^N b^N$. Also since it takes $N+1$ states to process a^N, any acceptance track of $a^N b^N$ must have a loop as shown below.



Now if we skip the loop, we will see that M_1 accepts the string $a^i a^{N-j} b^N = a^{N-(j-i)} b^N$. Since $i<j$, $a^{N-(j-i)} b^N \notin L_1$. So we have a contradiction because $L(M_1) = L_1$ and M_1 accepted $a^{N-(j-i)} b^N$. Hence L_1 cannot be regular. So L_1 is non-regular.

<u>Ex. 2</u> $L_2 = \{a^k b^m : k \leq m\}$ is a non-regular language.

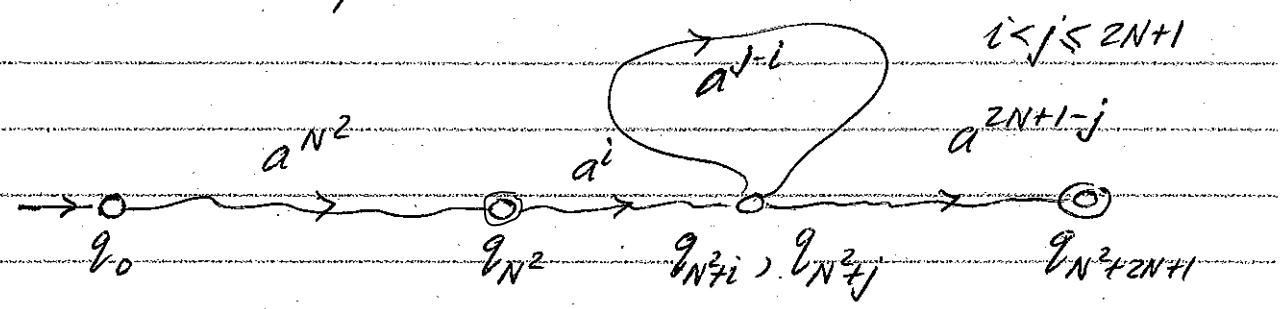
<u>Proof</u>: Suppose L_2 is regular. Then we can find a λ-free NFA M_2 such that $L(M_2) = L_2$. Let N = the number of states in M_2 and consider the string $a^N b^N$. Since $a^N b^N \in L_2$, M_2 will accept $a^N b^N$. Also since it takes $N+1$ states to process a^N, any acceptance track of $a^N b^N$ must contain a loop as shown below.



Now if we ride the loop twice, we will see that M_2 accepts the string $a^i a^{j-i} a^{j-i} a^{N-j} b^N = a^{N+(j-i)} b^N$. But $a^{N+(j-i)} b^N \notin L_2$ because $N+(j-i) \nleq N$ since $i<j$. This contradicts the fact that $L(M_2) = L_2$. Hence L_2 cannot be regular. So L_2 is non-regular

Ex.3 $L_3 = \{a^{k^2} : k \geq 0\}$ is a non-regular language.

Proof: Suppose L_3 is regular. Then we can find a λ-free NFA M_3 such that $L(M_3) = L_3$. Let N = the number of states in M_3 and consider the string $a^{(N+1)^2} = a^{N^2+2N+1}$. Since $a^{N^2+2N+1} \in L_3$, it will be accepted by M_3. Also since it takes $2N+2$ states to process the last $(2N+1)$ a's, any acceptance track of a^{N^2+2N+1} must have a loop as shown below.

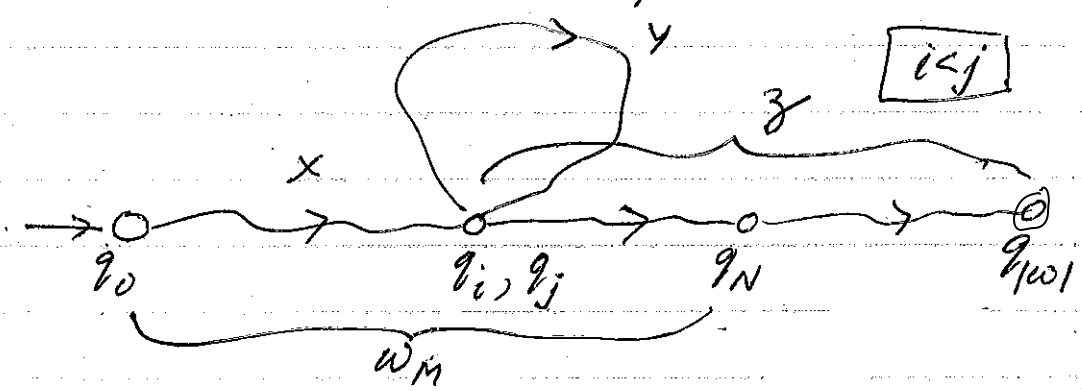


Now if we skip the loop, we will see that M_3 accepts the string $a^{N^2} \cdot a^i \cdot a^{2N+1-j} = a^{N^2+2N+1-(j-i)}$. But $N^2 < N^2+2N+1-(j-i) < N^2+2N+1 = (N+1)^2$, so it follows that $N^2+2N+1-(j-i)$ cannot be a perfect square, because it is strictly between two consecutive perfect squares. Hence $a^{N^2+2N+1-(j-i)} \notin L_3$. But this contradicts the fact that $L(M_3) = L_3$. Hence L_3 cannot be regular. So L_3 is a non-reg. lang.

We can summarize the idea in the last 3 examples in the following result.

<u>Prop. 12</u> (Pumping Lemma for Regular Languages)
Let L be any infinite regular language. Then we can find a positive integer N such that any $w \in L$ can be decomposed as $w = xyz$ with $|xy| \leq N$ & $|y| \geq 1$ such that $xy^rz \in L$ for each $r \in \mathbb{N} = \{0, 1, 2, 3, \ldots\}$

<u>Proof</u>: Since L is regular, we can find a λ-free NFA M such that $L(M) = L$. Put $N =$ the number of states in M. Now if w is any string in L with $|w| \geq N$, put $w_N =$ the string of the first N characters of w. Since $w \in L$, M will accept w. Also since M has only N states, and we need $N+1$ states to process w_M, any acceptance track of w must contain a loop as shown below



Now $|xy| \leq N$ because the loop will appear somewhere between q_0 & q_N. Also $|y| = j - i \geq 1$. If we ride the loop r times, we will see that M accepts xy^rz. So $xy^rz \in L$ for each $r \in \mathbb{N}$.

Ex4. Using the Pumping Lemma for Reg. lang, prove that $L_4 = \{\varphi \in \{a,b\}^* : 2n_a(\varphi) = n_b(\varphi)\}$ is non-regular.

Proof: Suppose L_4 is regular. Then by the Pumping Lemma, we can find an $N \geq 1$ such that any $w \in L_4$ with $|w| \geq N$ can be decomposed as $w = xyz$ with $|xy| \leq N$ & $|y| \geq 1$ such that $xy^rz \in L_4$ for all $r \in \mathbb{N}$.

Now consider the string $w = a^N b^{2N} \in L_4$. Since the first N characters of w are all a's, we must have $x = a^p$, $y = a^q$ and $z = a^{N-p-q} b^{2N}$ for some p & q with $0 \leq p \leq N-1$ & $q \geq 1$. But by the Pumping Lemma $xy^0z \in L_4$. So

$$a^p . \lambda . a^{N-p-q} b^{2N} = a^{N-q} b^{2N} \in L_4.$$

But this contradicts the fact that strings in L_4 have twice as many b's as a's. Hence L_4 must be non-regular.

We can also prove that certain languages are non-regular by using the Closure Theorem & previous results.

Ex.5 Show that $L_5 = \{a^k b^m : k \neq m\}$ is non-regular.

Proof: Suppose L_5 is regular. Then $a^*b^* - L_5$ will also be regular by Theorem 6(g). But

$$a^*b^* - L_5 = \{a^k b^m : k, m \geq 0\} - \{a^k b^m : k \neq m\} = \{a^k b^m : k = m\}$$
$$= \{a^k b^k : k \geq 0\} = L_1, \text{ which is non-reg.}$$

So this is a contradiction. Hence L_5 is non-regular.

We will now explore some of the properties of non-regular languages.

<u>Prop. 13</u>: If L is a non-regular language, then
(a) L^c is non-regular (b) L^R is non-regular

<u>Proof</u>: (a) Suppose L^c is regular. Then by the Closure Theorem (Theorem 6(e)), it follows that $(L^c)^c$ would also be regular. But $(L^c)^c = L$, so this contradicts the fact that L was non-regular. Hence L^c must also be non-regular.
(b) Suppose L^R is regular. Then by the Closure Theorem part (d), $(L^R)^R = L$ would also be regular. But this contradicts the fact that L is non-regular. Hence L^R must also be regular.

<u>Ex6</u> If L_1 & L_2 are non-regular languages, it is possible for (a) $L_1 \cup L_2$ to be regular, (b) $L_1 . L_2$ to be regular, and (c) L_1^* to be regular.

(a) Take $L_1 = \{a^k b^k : k \geq 0\}$ and $L_2 = \{a^k b^m : k \neq m\}$. Then L_1 & L_2 are both non-regular but $L_1 \cup L_2 = \underline{a}^* \underline{b}^*$ which is regular
(b) Take $L_1 = \{a^{k^2} : k \geq 0\}$ and $L_2 = \{a^m : m \text{ is not a square}\}$
$= L_1^c$. Then $L_1 . L_2 = \underline{aa}^*$ is regular because
$L_1 . L_2 = \{a^0, a^1, a^4, a^9, \ldots\} . \{a^2, a^3, a^5, a^6, a^7, a^8, a^{10}, \ldots\}$
$= \{a^2, a^3, a^4, a^5, \ldots\} = \{a^k : k \geq 2\} = \underline{aa}a^*$
(c) Take $L_1 = \{a^{k^2} : k \geq 0\}$. Then $L_1^* = \underline{a}^*$ which is regular bec. $\underline{a}^* = \{a\}^* \subseteq \{a^0, a^1, a^4, a^9, \ldots\}^* \subseteq \{a\}^*$.

END