Kleene's Theorem

Assignment 2

COMSATS University Islamabad
Sahiwal Campus

Usama Sarwar

FA17-BS(CS)-090-B

Mr. Yawar

Automata Theory

March 29, 2020

Kleen's Theorem

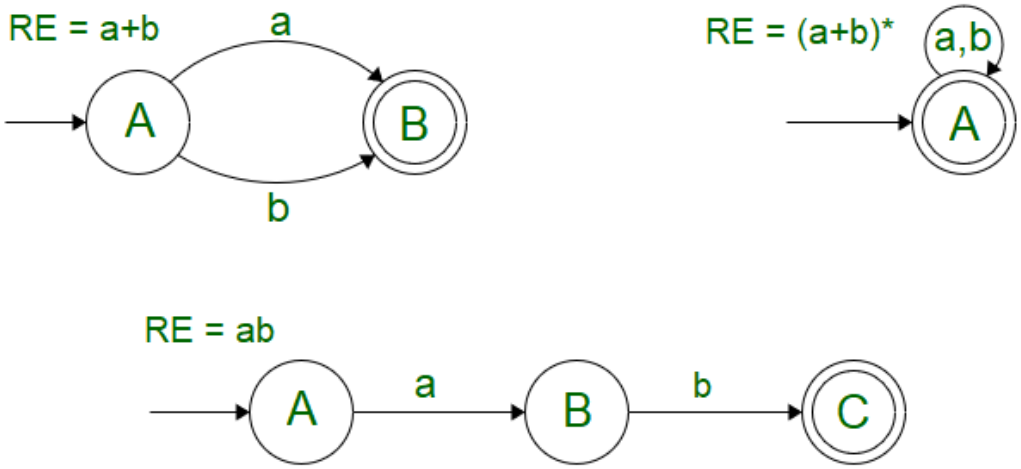
"It states that any regular language is accepted by an FA and conversely that any language accepted by an FA is regular."

1. Part I

A language is said to be regular if it can be represented by using a Finite Automata or if a Regular Expression can be generated for it. This definition leads us to the general definition that; For every Regular Expression corresponding to the language, a Finite Automata can be generated.

For certain expressions like: (a+b), ab, (a+b)* ; It's fairly easier to make the Finite Automata by just intuition as shown below. The problem arises when we are provided with a longer Regular Expression. This brings about the need for a systematic approach towards FA generation, which has been put forward by Kleene in **Kleene's Theorem – I**

For any Regular Expression r that represents Language L(r), there is a Finite Automata that accepts same language.



To understand Kleene's Theorem-I, Let's take in account the basic definition of Regular Expression where we observe that ϕ, ε and a single input symbol "a" can be included in a

KLEEN'S THEOREM

Regular Language and the corresponding operations that can be performed by the combination of these are:

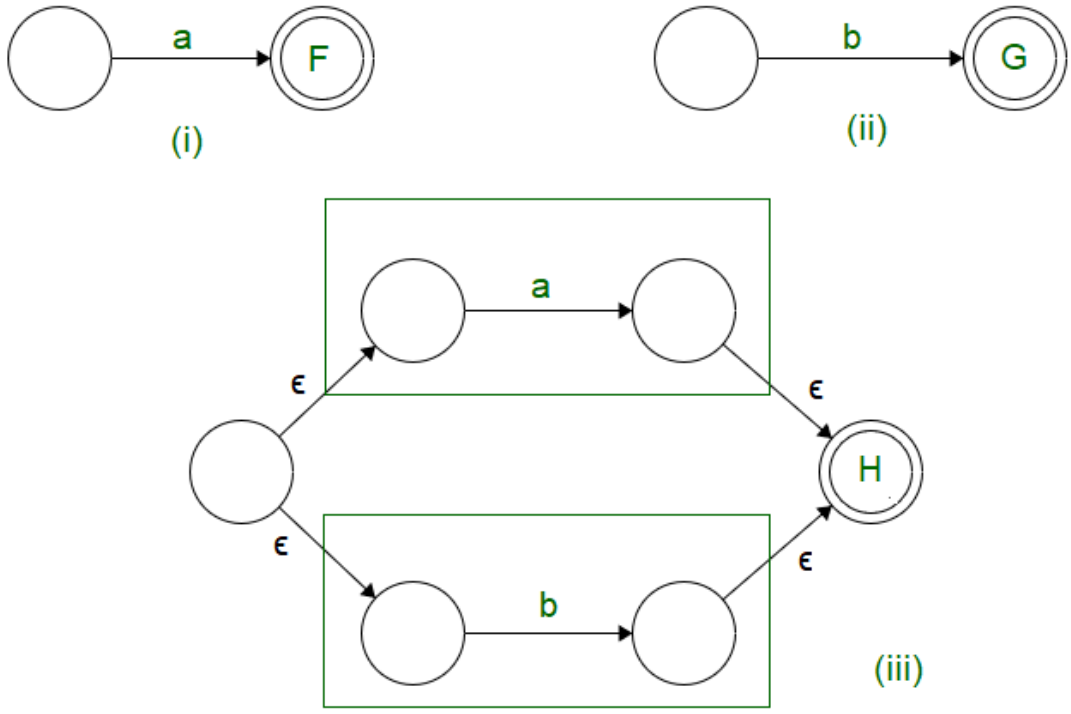
Say, r_1 and $r_2$be two regular expressions. Then,

1. $r_1 + r_2$ is a regular expression too, whose corresponding language is L(r_1) U L(r_2)
2. $r_1 . r_2$ is a regular expression too, whose corresponding language is L(r_1).L(r_2)
3. r_1* is a regular expression too, whose corresponding language is L(r_1)*

We can further use this definition in association with Null Transitions to give rise to a FA by the combination of two or more smaller Finite Automata (each corresponding to a Regular Expression).

Let S accept L = {a} and T accept L = {b}, then R can be represented as a combination of S and T using the provided operations as:

$$R = S + T$$

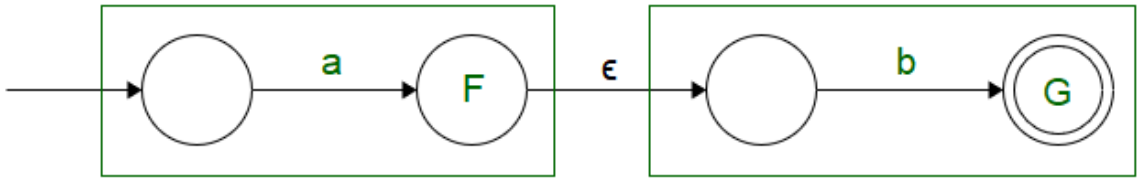


KLEEN'S THEOREM

We observe that,

1. In case of union operation, we can have a new start state, from which, null transition proceeds to the starting state of both the Finite State Machines.
2. The final states of both the Finite Automata's are converted to intermediate states. The final state is unified into one which can be traversed by null transitions.

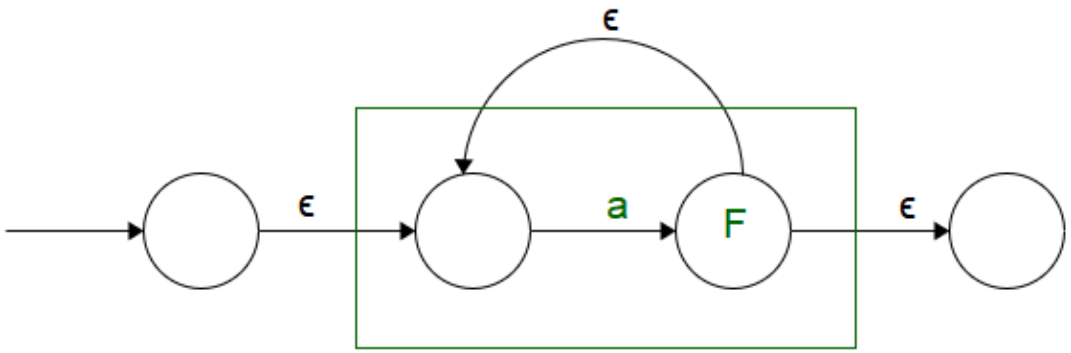
$$R = S.T$$



We observe that,

1. In case of concatenation operation we can have the same starting state as that of S, the only change occurs in the end state of S, which is converted to an intermediate state followed by a Null Transition.
2. The Null transition is followed by the starting state of T, the final state of T is used as the end state of R.

$$R = S^*$$



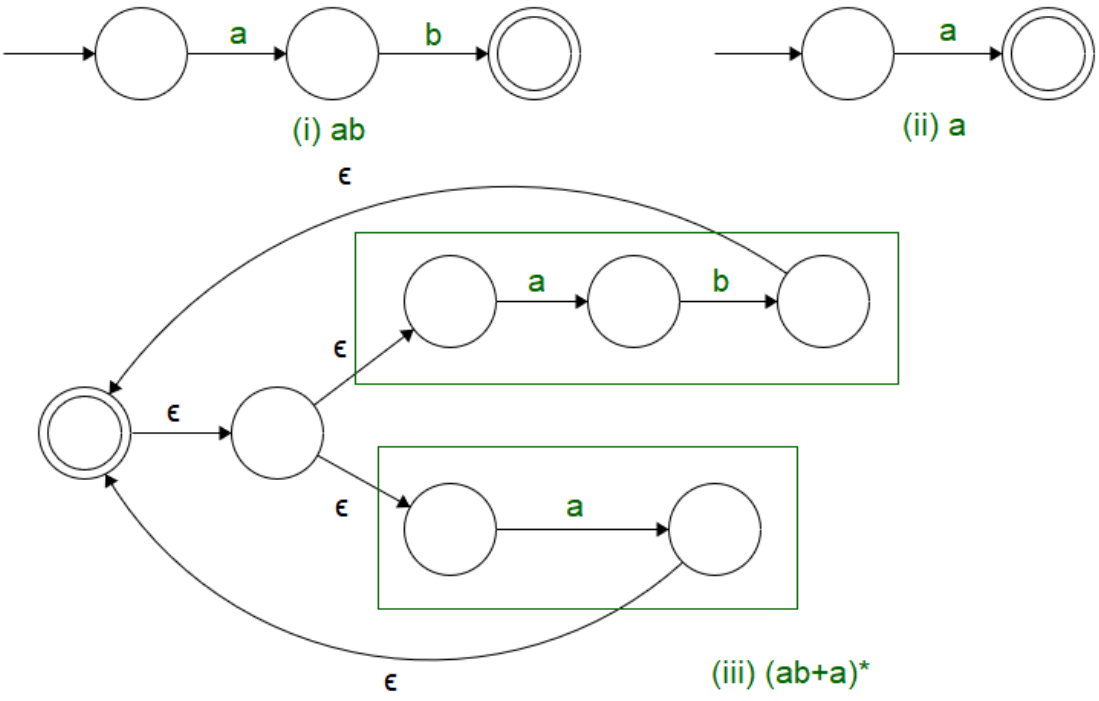
We observe that,

1. A new starting state is added, and S has been put as an intermediate state so that self looping condition could be incorporated.
2. Starting and Ending states have been defined separately so that the self looping condition is not disturbed.

Now that we are aware about the general operations. Let's see how Kleene's Theorem-I can be used to generate a FA for the given Regular Expression.

---

*Make a Finite Automata for the expression (ab+a)**



We see that using Kleene's Theorem – I gives a systematic approach towards the generation of a Finite Automata for the provided Regular Expression.

2. Part II

The converse of the part 1 of Kleene Theorem also holds true. It states that any language accepted by a finite automaton is regular. Before proceeding to a proof outline for the converse, let us study a method to compute the set of strings accepted by a finite automaton. Given a finite automaton, first relabel its states with the integers 1 through n, where n is the number of states of the finite automaton. Next denote by L(p, q, k) the set of strings representing paths from state p to state q that go through only states numbered no higher than k. Note that paths may go through arcs and vertices any number of times.

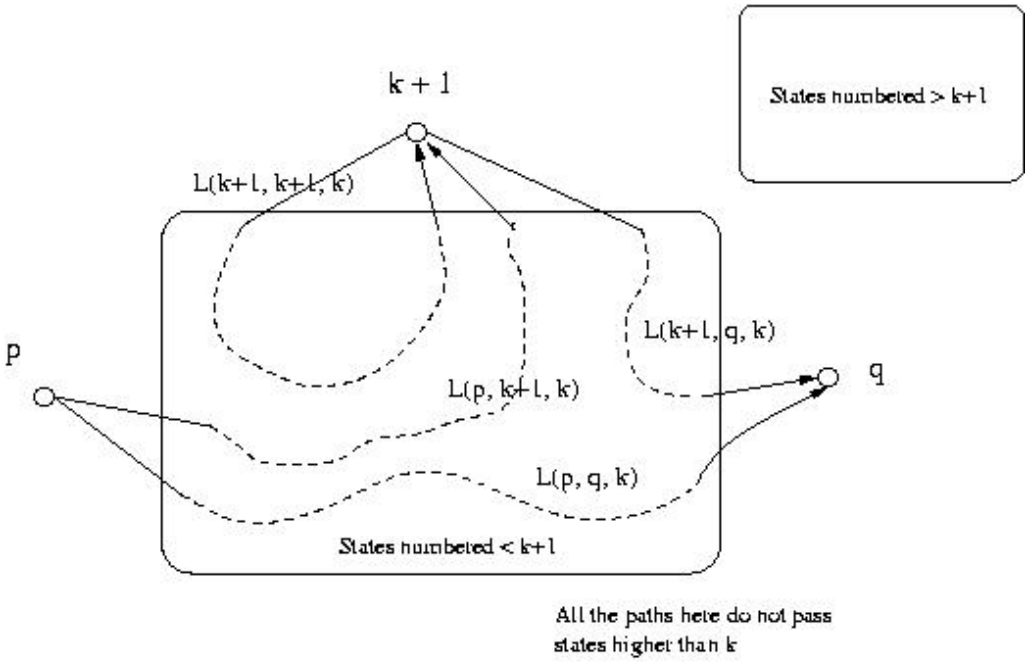
Then the following lemmas hold.

Lemma 1: L(p, q, k+1) = L(p, q, k) L(p, k+1, k)L(k+1, k+1, k)*L(k+1, q, k) .

What this lemma says is that the set of strings representing paths from p to q passing through states labeled with k+1 or lower numbers consists of the following two sets:

1. L(p, q, k) : The set of strings representing paths from p to q passing through states labeled with k or lower numbers.
2. L(p, k+1, k)L(k+1, k+1, k)*L(k+1, q, k) : The set of strings going first from p to k+1, then from k+1 to k+1 any number of times, then from k+1 to q, all without passing through states labeled higher than k.

See the figure below for the illustration.



KLEEN'S THEOREM

Lemma 2: L(p, q, 0) is regular.

Proof: L(p, q, 0) is the set of strings representing paths from p to q without passing any states in between. Hence if p and q are different, then it consists of single symbols representing arcs from p to q. If p = q, then is in it as well as the strings representing any loops at p (they are all single symbols). Since the number of symbols is finite and since any finite language is regular, L(p, q, 0) is regular.

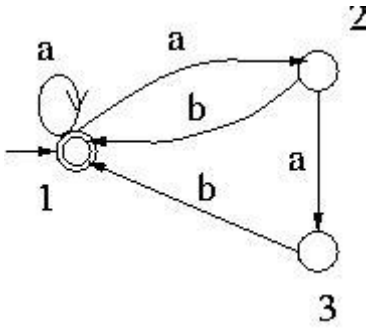
From Lemmas 1 and 2 by induction the following lemma holds.

Lemma 3: L(p, q, k) is regular for any states p and q and any natural number k.

Since the language accepted by a finite automaton is the union of L(q0, q, n) over all accepting states q, where n is the number of states of the finite automaton, we have the following converse of the part 1 of Kleene Theorem.

Theorem: Any language accepted by a finite automaton is regular.

Example: Let us find the language accepted by the following finite automaton using the lemmas.



Let us denote by r(p, q, k) the regular expression for the set of strings L(p, q, k).

Then the language accepted by this NFA is r(1, 1, 3).

By Lemma 1, r(1, 1, 3) = r(1, 1, 2) + r(1, 3, 2)r(3, 3, 2)*r(3, 1, 2) .

r(1, 1, 2):

r(1, 1, 2) = r(1, 1, 1) + r(1, 2, 1)r(2, 2, 1)*r(2, 1, 1)

r(1, 1, 1) = r(1,1,0) + r(1,1,0)r(1,1,0)*r(1,1,0) = a* , since r(1,1,0) = a + .

r(1, 2, 1) = r(1,2,0) + r(1,1,0)r(1,1,0)*r(1,2,0) = a+ , since r(1,2,0) = a .

r(2, 2, 1) = r(2,2,0) + r(2,1,0)r(1,1,0)*r(1,2,0) = ba+ + , since r(2,2,0) = and r(2,1,0) = b .

r(2, 1, 1) = r(2,1,0) + r(2,1,0)r(1,1,0)*r(1,1,0) = ba* .

Hence r(1, 1, 2) = a* + a+(b a+)*b a* .

r(1, 3, 2):

r(1, 3, 2) = r(1, 3, 1) + r(1, 2, 1)r(2, 2, 1)*r(2, 3, 1)

r(1, 3, 1) =

r(2, 3, 1) = a

Hence r(1, 3, 2) = a+(b a+ +)*a

= a+(b a+)*a .

r(3, 3, 2):

r(3, 3, 2) = r(3, 3, 1) + r(3, 2, 1)r(2, 2, 1)*r(2, 3, 1)

r(3, 3, 1) =

r(3, 2, 1) = r(3,2,0) + r(3,1,0)r(1,1,0)*r(1,2,0) = ba+ , since r(3,2,0) = and r(3,1,0) = b .

Hence r(3, 3, 2) = + ba+(ba+ +)*a

= + (ba+)+a

r(3, 1, 2):

r(3, 1, 2) = r(3, 1, 1) + r(3, 2, 1)r(2, 2, 1)*r(2, 1, 1)

r(3, 1, 1) = r(3,1,0) + r(3,1,0)r(1,1,0)r(1,1,0) = ba*

Hence r(3, 1, 2) = ba* + ba+(ba+ +)*ba*

= (ba+)*ba* .

Hence r(1, 1, 3) = a* + a+(b a+)*ba* + (a+(ba+)*a)(+ (ba+)+a)*(ba+)*ba*.

This can be further simplified to (a + ab + abb)*, then to (a + ab)*. The detail is left as an exercise though it would be quite challenging.

In this example there is only one accepting state. If there are more accepting states, then r(p, q, n) must be found for each accepting state q, where p is the initial state and n is the number of states in the given finite automaton, and all the r(p, q, n)'s must be added together to get the regular expression for the language accepted by the automaton.

3. Part III

Closure Properties of Context-Free Languages

We show that context-free languages are closed under union, concatenation,

and Kleene star. Suppose $G1 = (V1, \Sigma1, R1, S1)$ and $G2 = (V2, \Sigma2, R2, S2)$.

Example: For $G1$ we have

$S1 \rightarrow aS1b$

$S1 \rightarrow .$

For $G2$ we have

$S2 \rightarrow cS2d$

$S2 \rightarrow .$

Then $L(G1) = \{anbn : n \geq 0\}$. Also, $L(G2) = \{cndn : n \geq 0\}$

KLEEN'S THEOREM

3.1.1 Union

$G = (V1 \cup V2 \cup \{S\}, \Sigma1 \cup \Sigma2, R, S)$ where $R = R1 \cup R2 \cup \{S \rightarrow S1, S \rightarrow S2\}$ and S is a new symbol.

Then $L(G) = L(G1) \cup L(G2)$.

Example:

$S1 \rightarrow aS1b$

$S1 \rightarrow .$

$S2 \rightarrow cS2d$

$S2 \rightarrow .$

$S \rightarrow S1$

$S \rightarrow S2$

Then $L(G) = \{anbn : n \geq 0\} \cup \{cndn : n \geq 0\}$.

3.1.2 Concatenation

$G = (V1 \cup V2 \cup \{S\}, \Sigma1 \cup \Sigma2, R, S)$ where $R = R1 \cup R2 \cup \{S \rightarrow S1S2\}$ and S is a new symbol.

Example:

$S1 \rightarrow aS1b$

$S1 \rightarrow .$

$S2 \rightarrow cS2d$

$S2 \rightarrow .$

$S \rightarrow S1S2$

Then $L(G) = \{ambmcndn : m, n \geq 0\}$.

3.1.3 Kleene star

$G = (V1 \cup \{S\}, \Sigma1, R, S)$ where $R = R1 \cup \{S \rightarrow , S \rightarrow SS1\}$ and S is a new symbol.

Example:

$S1 \rightarrow aS1b$

$S1 \rightarrow .$

$S \rightarrow$

$S \rightarrow SS1$

Then $L(G) = \{anbn : n \geq 0\}*$.

Do some sample derivations.

3.1.4 Non-closure properties

Context-free languages are not closed under intersection or complement. This will be shown later.

3.1.5 Intersection with a regular language

The intersection of a context-free language and a regular language is contextfree. The idea of the proof is to simulate a push-down automaton and a finite state automaton in parallel and only accept if both machines accept.

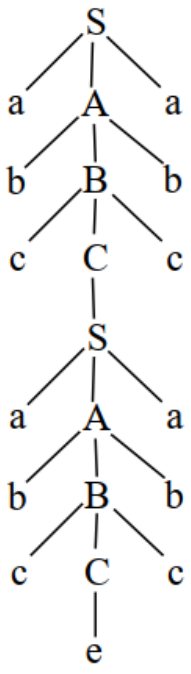
- Using this result one can show for example that the set of strings having equal numbers of *a* and *b* but no substring of the form *abaa* or *babb* is context-free; this would be very difficult to do using grammars.
- As another example, $\{anbn : n \geq 0\} \cap \{w \in \{a, b\}* : |w|$ is divisible by $3\}$ is context-free.

3.2 Example

Suppose a context-free grammar G is (V, Σ, R, S) where $\Sigma = \{a, b, c\}$, $V = \{S, A, B, C\}$, and R consists of the following rules:

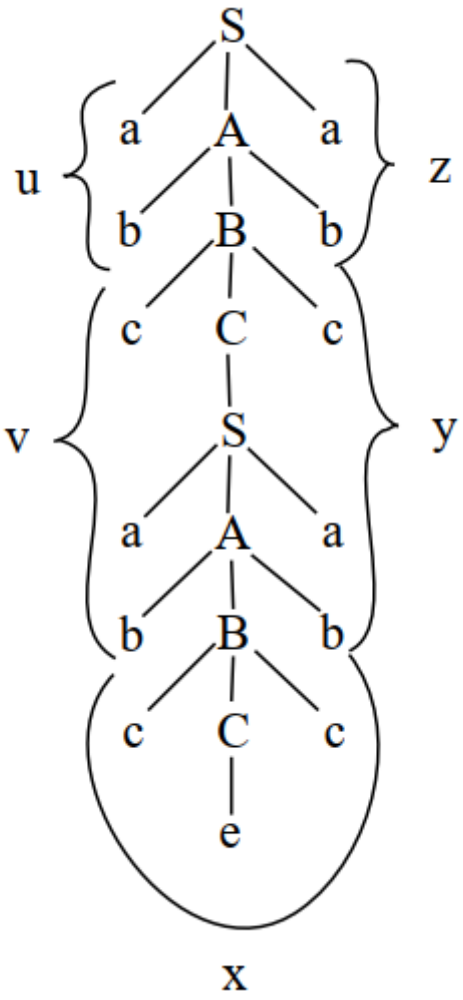
$S \rightarrow aAa$ $A \rightarrow bBb$ $B \rightarrow cCc$	$C \rightarrow S$ $C \rightarrow$

Then we have the following parse tree for the string *abcabccbacba* in $L(G)$:



KLEEN'S THEOREM

- Note that if a string is sufficiently long, a parse tree for the string will be very large, so it will have at least one very long path.
- This path will have some nonterminal appearing twice, just as the B (and other nonterminal) appear twice in this example.
- Now, using these two occurrences of B on the path, we can separate the string into substrings u, v, x, y, z as follows:



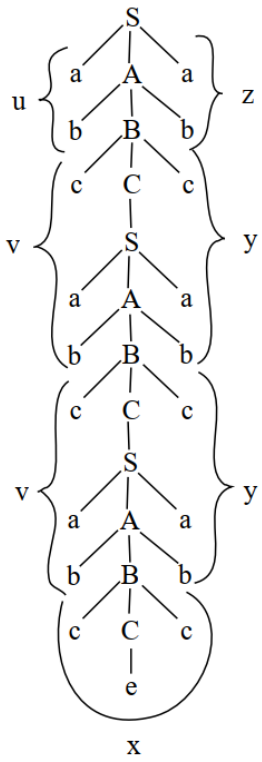
The string is divided according to what can be derived from the two

occurrences of B in the parse tree.

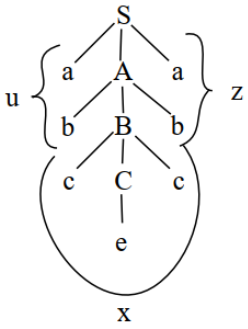
- Thus u = ab, v = cab, x = cc, y = bac, and z = ba.
- So the string abcabccbacba can be expressed as uvxyz in this way: (ab)(cab)(cc)(bac)(ba).

Now, note that the portion of the tree between the two B's can be duplicated, like this:

KLEEN'S THEOREM



This gives the string (ab)(cab)(cab)(cc)(bac)(bac)(ba), that is, uvvxyyz, or, uv2xy2z. In the same way, the portion of the parse tree between the two occurrences of B can be deleted, like this:



This gives the string (ab)(cc)(ba), that is, uxz, or, uv0xy0z. In the same way, one can obtain uvixyiz for any $i \geq 0$. Note that v and y are pumped at the same time.

Thus we have,

$S \Rightarrow^* abBba$

$B \Rightarrow^* cabBbac$;

$B \Rightarrow^* cc$.

So we can write this as

$S \Rightarrow^* uBz$

$B \Rightarrow^* vBy$

$B \Rightarrow^* x.$

Note now that

$B \Rightarrow^* vBy \Rightarrow^* vvByy \Rightarrow^* vvvByyy$

et cetera, so in general

$B \Rightarrow^* viByi$

for all i. Thus we have

$S \Rightarrow^* uBz \Rightarrow^* uxz,$

$S \Rightarrow^* uBz \Rightarrow^* uvByz \Rightarrow^* uvxyz,$

$S \Rightarrow^* uBz \Rightarrow^* uvByz \Rightarrow^* uvvByyz \Rightarrow^* uvvxyyz,$

and in a similar way we have $S \Rightarrow^* uvixyiz$ for all $i \geq 0$.

3.2.1 Example for the proof

Here is an example to illustrate the proof.

- Suppose N is 5, then $aNbNcN$ is $aaaaabbbbbcccccc$ or $a5b5c5$.
- Suppose $u = aa$, $v = aaabb$, $x = bb$, $y = bccc$, and $z = cc$.
- Thus $uvxyz$ is $(aa)(aaabb)(bb)(bccc)(cc)$.
- Then v and y together have all three symbols, and v has both a and b.
- Then $uv2xy2z$ is $(aa)(aaabb)2(bb)(bccc)2(cc)$, or, $(aa)(aaabb)(aaabb)(bb)(bccc)(bccc)(cc)$.
- This has a b before an a and is therefore not in L.

Now suppose that together v and y have only two of the three symbols.
In particular, suppose that $u = aa$, $v = aaa$, $w = bbbbb$, $y = ccc$, and $z = cc$.

- Thus $uvxyz = (aa)(aaa)(bbbbb)(ccc)(cc)$.
- Now, v and y together have only a and c, no b.
- Then $uv2xy2z$ is $(aa)(aaa)2(bbbbb)(ccc)2(cc)$.
- This string is $(aa)(aaa)(aaa)(bbbbb)(ccc)(ccc)(cc)$ which has 8 a, 5 b, and 8 c.
- The number of a and c in $uv2xy2z$ has increased over that of $uvxyz$, but the number of b has not.
- So the number of a, b, and c is not the same in $uv2xy2z$, so $uv2xy2z$ is not in L.