**Practical no. 7**

**finite state automata**

**A. Define grammar using nltk. Analyze a sentence using the same.**

In [3]:

**import** nltk

**import** nltk

**from** nltk **import** tokenize

grammar1 **=** nltk**.**CFG**.**fromstring("""

S -> VP

VP -> VP NP

NP -> Det NP

Det -> 'that'

NP -> singular Noun

NP -> 'flight'

VP -> 'Book'

""")

sentence **=** "Book that flight"

**for** index **in** range(len(sentence)):

all\_tokens **=** tokenize**.**word\_tokenize(sentence)

print(all\_tokens)

parser **=** nltk**.**ChartParser(grammar1)

**for** tree **in** parser**.**parse(all\_tokens):

print(tree)

tree**.**draw()

['Book', 'that', 'flight']

(S (VP (VP Book) (NP (Det that) (NP flight))))

**B. Accept the input string with Regular expression of Finite Automaton: 101+.**

In [ ]:

**def** FA(s):

*#if the length is less than 3 then it can't be accepted, Therefore end the process.*

**if** len(s)**<**3:

**return** "Rejected"

*#first three characters are fixed. Therefore, checking them using index*

**if** s[0]**==**'1':

**if** s[1]**==**'0':

**if** s[2]**==**'1':

*# After index 2 only "1" can appear. Therefore break the process if any othercharacter is detected*

**for** i **in** range(3, len(s)):

**if** s[i]**!=**'1':

**return** "Rejected"

**return** "Accepted"

**return** "Rejected"

**return** "Rejected"

**return** "Rejected"

inputs**=**['1','10101','101','10111','01010','100','','10111101','1011111']

**for** i **in** inputs:

print(FA(i))

**C. Accept the input string with Regular expression of FA: (a+b)\*bba.**

**def** FA(s):

size**=**0

*#scan complete string and make sure that it contains only 'a' & 'b'*

**for** i **in** s:

**if** i**==**'a' **or** i**==**'b':

size**+=**1

**else**:

**return** "Rejected"

*#After checking that it contains only 'a' & 'b'*

*#check it's length it should be 3 atleast*

**if** size**>=**3:

*#check the last 3 elements*

**if** s[size**-**3]**==**'b':

**if** s[size**-**2]**==**'b':

**if** s[size**-**1]**==**'a':

**return** "Accepted"

**return** "Rejected"

**return** "Rejected"

**return** "Rejected"

**return** "Rejected"

inputs**=**['bba', 'ababbba', 'abba','abb', 'baba','bbb','']

**for** i **in** inputs:

print(FA(i))

## D. Implementation of Deductive Chart Parsing using context free grammar and a given sentence.

**import** nltk

**from** nltk **import** tokenize

grammar1 **=** nltk**.**CFG**.**fromstring("""

S -> NP VP

PP -> P NP

NP -> Det N | Det N PP | 'I'

VP -> V NP | VP PP

Det -> 'a' | 'my'

N -> 'bird' | 'balcony'

V -> 'saw'

P -> 'in'

""")

sentence **=** "I saw a bird in my balcony"

**for** index **in** range(len(sentence)):

all\_tokens **=** tokenize**.**word\_tokenize(sentence)

print(all\_tokens)

*# all\_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']*

parser **=** nltk**.**ChartParser(grammar1)

**for** tree **in** parser**.**parse(all\_tokens):

print(tree)

tree**.**draw()

['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']

(S

(NP I)

(VP

(VP (V saw) (NP (Det a) (N bird)))

(PP (P in) (NP (Det my) (N balcony)))))

(S

(NP I)

(VP

(V saw)

(NP (Det a) (N bird) (PP (P in) (NP (Det my) (N balcony))))))

In [ ]:

In [ ]: