Compiler Design

Exp–2 Regex To NFA

Name:- K. DUSHYANT REDDY Reg No.:- RA1911033010029

Branch:- CSE-SE

CODE:

*class* Sort:

    Character=1

    Join=2

    Positive=3

    Kleene=4

*class* DispDataTree:

*def* \_\_init\_\_(*self*, *\_type*,*value*=None):

*self*.\_type = *\_type*

*self*.value = *value*

*self*.left = None

*self*.right = None

*def* buildDataTree(*exp*):

    load=[]

    for x in *exp*:

        if x.isalpha():

            load.append(DispDataTree(Sort.Character, x))

        else:

            if x == "+":

                s=DispDataTree(Sort.Positive)

                s.right=load.pop()

                s.left=load.pop()

            elif x == ".":

                s=DispDataTree(Sort.Join)

                s.right=load.pop()

                s.left=load.pop()

            elif x == "\*":

                s=DispDataTree(Sort.Kleene)

                s.left=load.pop()

            load.append(s)

    return load[0]

*def* seq(*val*):

    if *val*.\_type == Sort.Character:

        print(*val*.value)

    elif *val*.\_type == Sort.Join:

        seq(*val*.left)

        print(".")

        seq(*val*.right)

    elif *val*.\_type == Sort.Positive:

        seq(*val*.left)

        print("+")

        seq(*val*.right)

    elif *val*.\_type == Sort.Kleene:

        seq(*val*.left)

        print("\*")

*def* first(*p1*,*p2*):

    sym=["+",".","\*"]

    return sym.index(*p1*)>sym.index(*p2*)

*def* ad(*exp*):

    t=[]

    for i in range(len(*exp*)):

        if i != 0\

            and (*exp*[i-1].isalpha() or *exp*[i-1] == ")" or *exp*[i-1] == "\*")\

            and (*exp*[i].isalpha() or *exp*[i] == "("):

            t.append(".")

        t.append(*exp*[i])

*exp* = t

    load = []

    res = ""

    for x in *exp*:

        if x.isalpha():

            res=res+x

            continue

        if x == ")":

            while len(load) != 0 and load[-1] != "(":

                res=res+load.pop()

            load.pop()

        elif x == "(":

            load.append(x)

        elif x == "\*":

            res=res+x

        elif len(load) == 0 or load[-1] == "(" or first(x, load[-1]):

            load.append(x)

        else:

            while len(load) != 0 and load[-1] != "(" and not first(x, load[-1]):

                res=res+load.pop()

            load.append(x)

    while len(load) != 0:

        res=res+load.pop()

    return res

*class* FA:

*def* \_\_init\_\_(*self*):

*self*.changestate = {}

*def* cal(*val*):

    if *val*.\_type == Sort.Character:

        return calCharacter(*val*)

    elif *val*.\_type == Sort.Join:

        return calJoin(*val*)

    elif *val*.\_type == Sort.Positive:

        return calPositive(*val*)

    elif *val*.\_type == Sort.Kleene:

        return calKleene(*val*)

*def* calCharacter(*val*):

    initialstate=FA()

    finalstate=FA()

    initialstate.changestate[*val*.value] = [finalstate]

    return initialstate,finalstate

*def* calJoin(*val*):

    lnfa=cal(*val*.left)

    rnfa=cal(*val*.right)

    lnfa[1].changestate['-'] = [rnfa[0]]

    return lnfa[0],rnfa[1]

*def* calPositive(*val*):

    initialstate=FA()

    finalstate=FA()

    tnfa=cal(*val*.left)

    bnfa=cal(*val*.right)

    initialstate.changestate['-']=[tnfa[0],bnfa[0]]

    tnfa[1].changestate['-']=[finalstate]

    bnfa[1].changestate['-']=[finalstate]

    return initialstate,finalstate

*def* calKleene(*val*):

    initialstate=FA()

    finalstate=FA()

    snfa=cal(*val*.left)

    initialstate.changestate['-']=[snfa[0],finalstate]

    snfa[1].changestate['-']=[snfa[0],finalstate]

    return initialstate,finalstate

*def* showTrns(*state*,*complete*,*table*):

    if *state* in *complete*:

        return

*complete*.append(*state*)

    for symbol in list(*state*.changestate):

        prntopt="q"+str(*table*[*state*])+"\t\t\t"+symbol+"\t\t\t\t"

        for er in *state*.changestate[symbol]:

            if er not in *table*:

*table*[er]=1+sorted(*table*.values())[-1]

            prntopt=prntopt+"q"+str(*table*[er])+" "

        print(prntopt)

        for er in *state*.changestate[symbol]:

            showTrns(er,*complete*,*table*)

*def* showTTable(*NFA*):

    print("Present State\t\tTransition\t\t\tNext State")

    showTrns(*NFA*[0], [], {*NFA*[0]:0})

r=input("Enter regex: ")

pr=ad(r)

val=buildDataTree(pr)

fa = cal(val)

showTTable(fa)

OUTPUT:

