Exp 10 - Construction of LR(0) items

Aim: A program to implement construction of LR(0) items

Algorithm:

1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.
5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.
6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.
9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.
10. Go to step 5.
11. End of state building.
12. Display the output.
13. End.

Exp 10 - Construction of LR(0) items
RA1911003010143
Abhishek Kumar

Program:

```
import os
from collections import Counter
import pyfiglet
import termtables as tt


def append_dot(a):
    jj = a.replace("->", "->.")
    return jj


def compress_name(name: str):
    res = Counter(name)
    comp = ''
    for r in res:
        comp += r + str(res[r])

    return comp


def save_file(final_string, grammar, name):
    directory = os.path.dirname("parsable_strings/" + str(grammar) + "/")
    if not os.path.exists(directory):
        os.makedirs(directory)

    with open("parsable_strings/{0}/{1}.txt".format(grammar, name), 'w') as f:
        f.write(final_string)


def closure(a):
    temp = [a]
    for it in temp:
        jj = it[it.index(".") + 1]
        if jj != len(it) - 1:
            for k in prod:
                if k[0][0] == jj and (append_dot(k)) not in temp:
                    temp.append(append_dot(k))
        else:
            for k in prod:
                if k[0][0] == jj and it not in temp:
                    temp.append(it)

    return temp


def swap(new, pos):
    new = list(new)
    temp = new[pos]
    if pos != len(new):
        new[pos] = new[pos + 1]
        new[pos + 1] = temp
        new1 = "".join(new)
        return new1
    else:
        return "".join(new)


def goto1(x1):
    hh = []
    pos = x1.index(".")
    if pos != len(x1) - 1:
        jj = list(x1)
        kk = swap(jj, pos)
        if kk.index(".") != len(kk) - 1:
            jjj = closure(kk)
            return jjj
        else:
            hh.append(kk)
            return hh
    else:
        return x1


def get_terminals(gram):
    terms = set()
    for p in gram:
        x1 = p.split('->')
        for t in x1[1].strip():
            if not t.isupper() and t != '.' and t != '':
                terms.add(t)

    terms.add('$')

    return terms


def get_non_terminals(gram):
    terms = set()
    for p in gram:
        x1 = p.split('->')
        for t in x1[1].strip():
            if t.isupper():
                terms.add(t)

    return terms


def get_list(graph, state):
    final = []
    for g in graph:
        if int(g.split()[0]) == state:
            final.append(g)

    return final


if __name__ == '__main__':
```

Exp 10 - Construction of LR(0) items
RA1911003010143
Abhishek Kumar

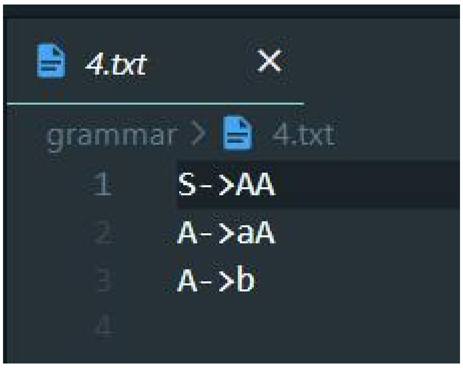
```
result = pyfiglet.figlet_format("LR (0) Parsing", font="epic")
print(result)

prod = []
set_of_items = []
c = []

num = int(input("Enter grammar number: "))
print("\n")

with open("grammar/" + str(num) + ".txt", 'r') as fp:
    for i in fp.readlines():
        prod.append(i.strip())

prod.insert(0, "X->.S")
print("--------------------------------------------------------------")
print("Augmented Grammar")
print(prod)

prod_num = {}
for i in range(1, len(prod)):
    prod_num[str(prod[i])] = i

j = closure("X->.S")
set_of_items.append(j)

state_numbers = {}
dfa_prod = {}
items = 0
while True:
    if len(set_of_items) == 0:
        break

    jk = set_of_items.pop(0)
    kl = jk
    c.append(jk)
    state_numbers[str(jk)] = items
    items += 1

    if len(jk) > 1:
        for item in jk:
            jl = goto1(item)
            if jl not in set_of_items and jl != kl:
                set_of_items.append(jl)
                dfa_prod[str(state_numbers[str(jk)]) + " " + str(item)] = jl
            else:
                dfa_prod[str(state_numbers[str(jk)]) + " " + str(item)] = jl

for item in c:
    for j in range(len(item)):
        if goto1(item[j]) not in c:
            if item[j].index(".") != len(item[j]) - 1:
                c.append(goto1(item[j]))

print("--------------------------------------------------------------")
print("Total States: ", len(c))
```

```
    for i in range(len(c)):
        print(i, ":", c[i])
    print("--------------------------------------------------------------")

    dfa = {}
    for i in range(len(c)):
        if i in dfa:
            pass
        else:
            lst = get_list(dfa_prod, i)
            samp = {}
            for j in lst:
                s = j.split()[1].split('->')[1]
                search = s[s.index('.') + 1]
                samp[search] = state_numbers[str(dfa_prod[j])]

            if samp != {}:
                dfa[i] = samp

    # print(dfa)

    # Generate parsing table
    table = []

    term = sorted(list(get_terminals(prod)))
    header = [''] * (len(term) + 1)
    header[(len(term) + 1) // 2] = 'Action'

    non_term = sorted(list(get_non_terminals(prod)))

    header2 = [''] * len(non_term)
    header2[(len(non_term)) // 2] = 'Goto'

    table.append([''] + term + non_term)

    table_dic = {}

    for i in range(len(c)):
        data = [''] * (len(term) + len(non_term))
        samp = {}

        # Action
        try:
            for j in dfa[i]:
                if not j.isupper() and j != '' and j != '.':
                    ind = term.index(j)
                    data[ind] = 'S' + str(dfa[i][j])
                    samp[term[ind]] = 'S' + str(dfa[i][j])

        except Exception:
            if i != 1:
                s = list(c[i][0])
                s.remove('.')
                s = "".join(s)
                lst = [i] + ['r' + str(prod_num[s])] * len(term)
                lst += [''] * len(non_term)
```

Exp 10 - Construction of LR(0) items
RA1911003010143
Abhishek Kumar

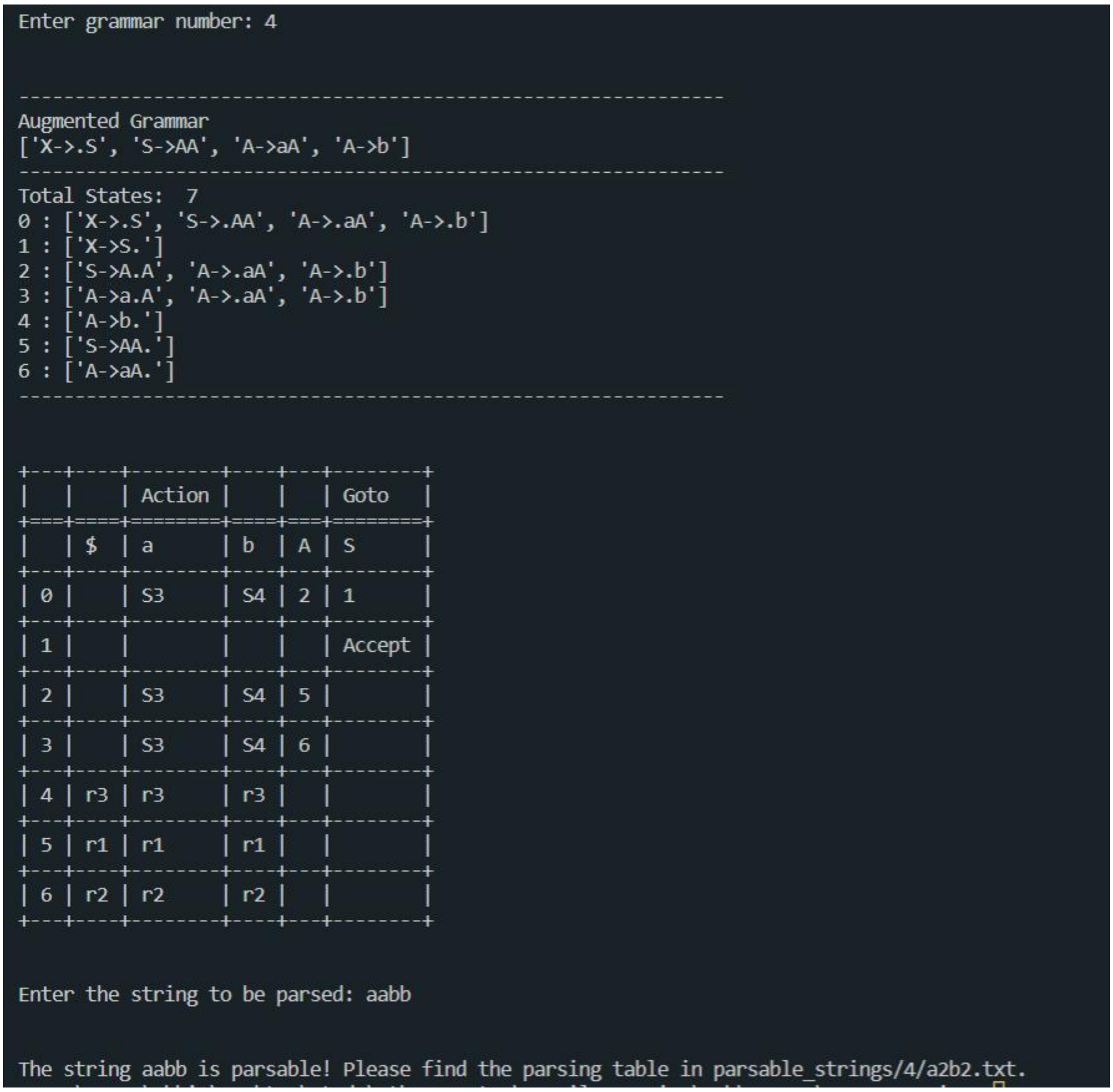
```
                table.append(lst)
                for j in term:
                    samp[j] = 'r' + str(prod_num[s])
            else:
                lst = [i] + [''] * (len(term) + len(non_term))
                lst[-1] = 'Accept'
                table.append(lst)

        # Goto
        try:
            for j in dfa[i]:
                if j.isupper():
                    ind = non_term.index(j)
                    data[len(term) + ind] = dfa[i][j]

                    samp[j] = str(dfa[i][j])

            table.append([i] + data)
        except Exception:
            pass

        if samp == {}:
            table_dic[i] = {'$': 'Accept'}
        else:
            table_dic[i] = samp

    final_table = tt.to_string(data=table, header=header + header2, style=tt.styles.ascii_thin_double,
padding=(0, 1))

    print("\n")
    print(final_table)
    print("\n")

    # Parse String
    string = input("Enter the string to be parsed: ")
    string += '$'
    print("\n")

    stack = [0]
    pointer = 0

    # print(table_dic)

    header = ['Process', 'Look Ahead', 'Symbol', 'Stack']
    data = []

    i = 0
    accepted = False
    while True:
        try:
            try:
                prods = dfa[stack[-1]]
                prod_i = prods[string[i]]  # state num
            except Exception:
                prod_i = None
            try:
                tab = table_dic[stack[-1]]
                tab_i = tab[string[i]]  # S or r
            except Exception:
                tab = table_dic[stack[-2]]
                tab_i = tab[stack[-1]]  # S or r

            if tab_i == 'Accept':
                data.append(['Action({0}, {1}) = {2}'.format(stack[-1], string[i], tab_i), i, string[i], str(stack)])
                accepted = True
                break
            else:
                if tab_i[0] == 'S' and not str(stack[-1]).isupper():
                    lst = ['Action({0}, {1}) = {2}'.format(stack[-1], string[i], tab_i), i, string[i]]
                    stack.append(string[i])
                    stack.append(prod_i)
                    lst.append(str(stack))
                    data.append(lst)
                    i += 1
                elif tab_i[0] == 'r':
                    lst = ['Action({0}, {1}) = {2}'.format(stack[-1], string[i], tab_i), i, string[i]]
                    x = None
                    for i1 in prod_num:
                        if prod_num[i1] == int(tab_i[1]):
                            x = i1
                            break

                    length = 2 * (len(x.split('->')[1]))
                    for _ in range(length):
                        stack.pop()

                    stack.append(x[0])
                    lst.append(str(stack))
                    data.append(lst)
                else:
                    lst = ['goto({0}, {1}) = {2}'.format(stack[-2], stack[-1], tab_i), i, string[i]]
                    stack.append(int(tab_i))
                    lst.append(str(stack))
                    data.append(lst)
        except Exception:
            accepted = False
            break

    try:
        parsing_table = tt.to_string(data=data, header=header, style=tt.styles.ascii_thin_double,
padding=(0, 1))

        if accepted:
            string = string[:-1]

            compressed_name = compress_name(string)
            save_file(parsing_table, num, compressed_name)

            print("The string {0} is parsable! Please find the parsing table in "
                  "parsable_strings/{1}/{2}.txt.".format(string, num, compressed_name))
        else:
            print("The string {0} is not parsable!".format(string))
    except Exception:
        print("Invalid string entered!")
```

Input:



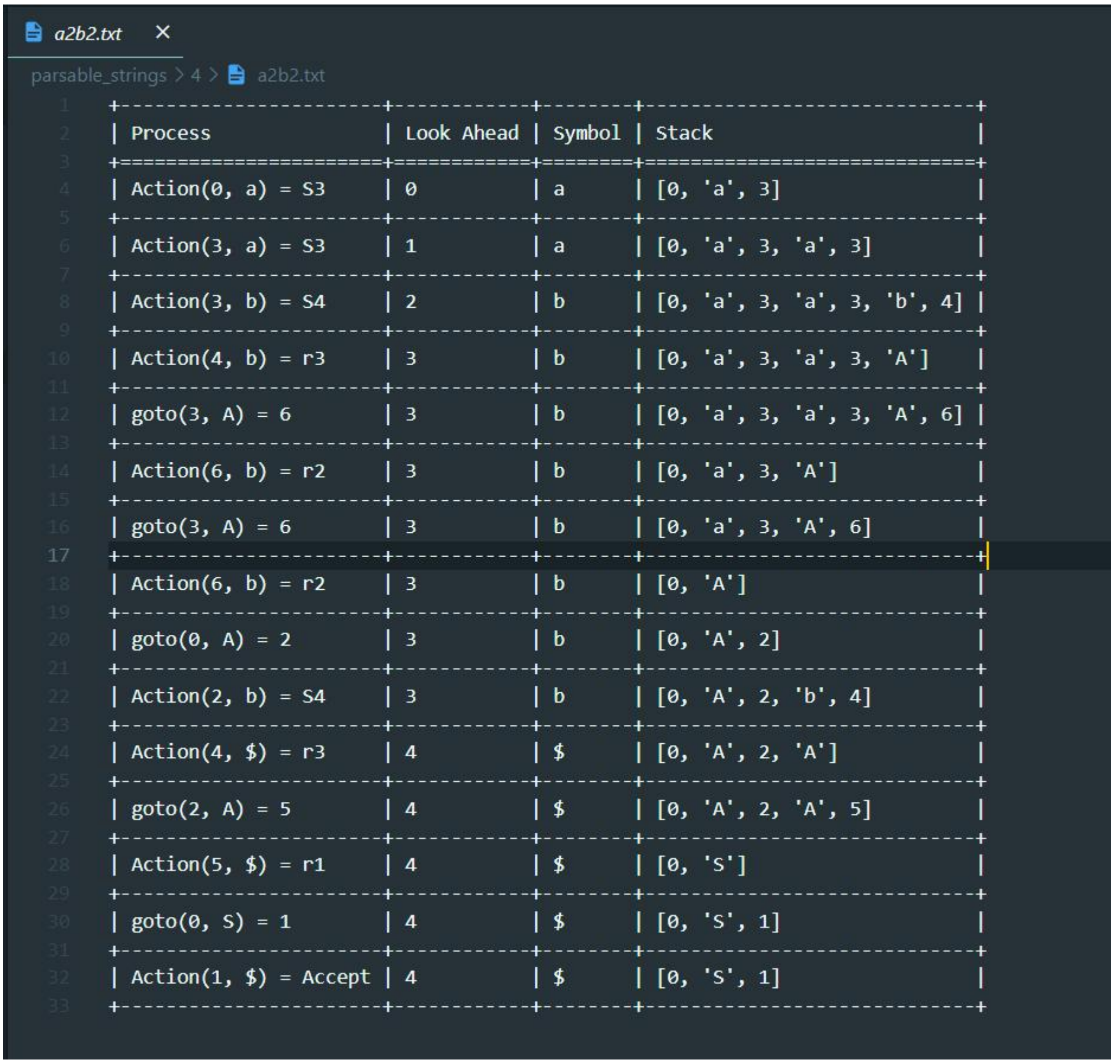
```
S->AA
A->aA
A->b
```

Output:



```
Enter grammar number: 4


--------------------------------------------------------------
Augmented Grammar
['X->.S', 'S->AA', 'A->aA', 'A->b']
--------------------------------------------------------------
Total States:  7
0 : ['X->.S', 'S->.AA', 'A->.aA', 'A->.b']
1 : ['X->S.']
2 : ['S->A.A', 'A->.aA', 'A->.b']
3 : ['A->a.A', 'A->.aA', 'A->.b']
4 : ['A->b.']
5 : ['S->AA.']
6 : ['A->aA.']
--------------------------------------------------------------



+---+----+--------+----+---+--------+
|   |    | Action |    |   | Goto   |
+===+====+========+====+===+========+
|   | $  | a      | b  | A | S      |
+---+----+--------+----+---+--------+
| 0 |    | S3     | S4 | 2 | 1      |
+---+----+--------+----+---+--------+
| 1 |    |        |    |   | Accept |
+---+----+--------+----+---+--------+
| 2 |    | S3     | S4 | 5 |        |
+---+----+--------+----+---+--------+
| 3 |    | S3     | S4 | 6 |        |
+---+----+--------+----+---+--------+
| 4 | r3 | r3     | r3 |   |        |
+---+----+--------+----+---+--------+
| 5 | r1 | r1     | r1 |   |        |
+---+----+--------+----+---+--------+
| 6 | r2 | r2     | r2 |   |        |
+---+----+--------+----+---+--------+


Enter the string to be parsed: aabb


The string aabb is parsable! Please find the parsing table in parsable_strings/4/a2b2.txt.
```

Exp 10 - Construction of LR(0) items
RA1911003010143
Abhishek Kumar



```
+-----------------------+------------+--------+-----------------------------+
| Process               | Look Ahead | Symbol | Stack                       |
+=======================+============+========+=============================+
| Action(0, a) = S3     | 0          | a      | [0, 'a', 3]                 |
+-----------------------+------------+--------+-----------------------------+
| Action(3, a) = S3     | 1          | a      | [0, 'a', 3, 'a', 3]         |
+-----------------------+------------+--------+-----------------------------+
| Action(3, b) = S4     | 2          | b      | [0, 'a', 3, 'a', 3, 'b', 4] |
+-----------------------+------------+--------+-----------------------------+
| Action(4, b) = r3     | 3          | b      | [0, 'a', 3, 'a', 3, 'A']     |
+-----------------------+------------+--------+-----------------------------+
| goto(3, A) = 6        | 3          | b      | [0, 'a', 3, 'a', 3, 'A', 6] |
+-----------------------+------------+--------+-----------------------------+
| Action(6, b) = r2     | 3          | b      | [0, 'a', 3, 'A']             |
+-----------------------+------------+--------+-----------------------------+
| goto(3, A) = 6        | 3          | b      | [0, 'a', 3, 'A', 6]          |
+-----------------------+------------+--------+-----------------------------+
| Action(6, b) = r2     | 3          | b      | [0, 'A']                     |
+-----------------------+------------+--------+-----------------------------+
| goto(0, A) = 2        | 3          | b      | [0, 'A', 2]                  |
+-----------------------+------------+--------+-----------------------------+
| Action(2, b) = S4     | 3          | b      | [0, 'A', 2, 'b', 4]          |
+-----------------------+------------+--------+-----------------------------+
| Action(4, $) = r3     | 4          | $      | [0, 'A', 2, 'A']             |
+-----------------------+------------+--------+-----------------------------+
| goto(2, A) = 5        | 4          | $      | [0, 'A', 2, 'A', 5]          |
+-----------------------+------------+--------+-----------------------------+
| Action(5, $) = r1     | 4          | $      | [0, 'S']                     |
+-----------------------+------------+--------+-----------------------------+
| goto(0, S) = 1        | 4          | $      | [0, 'S', 1]                  |
+-----------------------+------------+--------+-----------------------------+
| Action(1, $) = Accept | 4          | $      | [0, 'S', 1]                  |
+-----------------------+------------+--------+-----------------------------+
```

Result: The implementation of LR(0) Parser was compiled, executed and verified successfully