PDA TO CFG CONVERSION

Theory of Computation Assignment

-Kapeel Suryavanshi (BT16CSE084)

Algorithm:

1. First, I print all the permutations start states of the CFG i.e. S -> [qZp] where p being states of PDA.
2. For transitions in PDA of form $\delta(q,a,X) = (r, Y_1Y_2.....Y_k)$, I will get a CFG of form $[qXr_k]$ -> a $[rY_1r_1]$ $[r_1Y_2r_2]$……$[r_{k-1}Y_kr_k]$ for each r_i (i=1,2,…k) being a state of PDA. In order to get this, I generate all permutations of the states in PDA.
3. For transitions in PDA of form $\delta(q,a,X) = (r, \varepsilon)$, I will get a CFG of form [qXr] -> a.
4. Lastly, I add [qZp] -> ε for p and q being states of PDA.

Implementation:

- **<u>Variables and Data Type used</u>*:***

  Data Type Used: vector<string>

  Variables used are in this context : For each transition I of PDA,

  δ(from_state[i] ,input[i] ,top_stack[i]) = (to_state[i] , push_stack[i])

  (Only variable permutes[][] is a 2D vector of strings, and it stores all the permutations)

- **<u>Functions and their description:</u>**

  - **int main():**

    Driver Function. It takes the input of the CFG from a file and calls PDA_to_CFG with appropriate parameters.

  - **void PDA_to_CFG(***vector<string> &from_state, vector<string> &input, vector<string> &top_stack, vector<string> &to_state, vector<string> &push_stack, vector<string> states***):**

    This function does the main job of printing the CFG for the given PDA. It starts by printing all the Start States of the CFG. Then, based on the form of the transition in PDA, it prints the corresponding form in CFG. For transition in PDA of form as in step 2 of algorithm, it calls print_permute(), to print its equivalent production rule.

  - **void generate_permutations(***vector<string> states, vector<string> &temp,vector<vector<string> > &permutes,int k***):**

    This function generates all the permutations of the states of PDA and stores it in permutes[][]. permutes[][] is used by print_permute();

  - **void print_permutes(***string from_state, string input, string top_stack, string to_state, string push_stack, vector<string> states***):**

    This function prints the production rules of a transition of form mentioned in step 2 of algorithm. It calls generate_permutations() that generates all the permutations of the states that are to be used in writing production rule, and stores them in permutes[][]. It then uses permutes[][] to print the production rule of the transition in PDA.

- **Input:**
  - ❖ Input is taken from a file.
  - ❖ First line of the input consists of the states of PDA.
  - ❖ Rest of the lines, are the transitions in PDA
  - ❖ Example :

    q p

    δ(q,1,Z) = (q,XZ)

    δ(q,1,X) = (q,XX)

    δ(q,0,X) = (p,X)

    δ(p,1,X) = (p,ε)

    δ(p,0,Z) = (q,Z)
- **Output:**
  - ❖ First, the PDA given as input is printed.
  - ❖ Then, its corresponding CFG is printed.
  - ❖ Output of my program when above input was given is :

```
kapeel@kapeel-VirtualBox:~/Desktop/TOC$ g++ PDA_to_CFG.cpp
kapeel@kapeel-VirtualBox:~/Desktop/TOC$ ./a.out
PDA given as input :
States = q p
δ(q,1,Z) = (q,XZ)
δ(q,1,X) = (q,XX)
δ(q,0,X) = (p,X)
δ(p,1,X) = (p,ε)
δ(p,0,Z) = (q,Z)
--------------------------------------------------------------------------
Corresponding CFG :
Start State = S -> [qZq] | [qZp]

δ(q,1,Z) = (q,XZ)
[qZq] -> 1 [qXq] [qZq]
[qZp] -> 1 [qXq] [qZp]
[qZq] -> 1 [qXp] [pZq]
[qZp] -> 1 [qXp] [pZp]

δ(q,1,X) = (q,XX)
[qXq] -> 1 [qXq] [qXq]
[qXp] -> 1 [qXq] [qXp]
[qXq] -> 1 [qXp] [pXq]
[qXp] -> 1 [qXp] [pXp]

δ(q,0,X) = (p,X)
[qXq] -> 0 [pXq]
[qXp] -> 0 [pXp]

δ(p,1,X) = (p,ε)
[pXp] -> 1

δ(p,0,Z) = (q,Z)
[pZq] -> 0 [qZq]
[pZp] -> 0 [qZp]

[qZq] -> ε
[qZp] -> ε
[pZq] -> ε
[pZp] -> ε
kapeel@kapeel-VirtualBox:~/Desktop/TOC$
```