UCS 1602 - Compiler Design

Assignment-3

Elimination of Immediate Left Recursion using C

Swetha Saseendran
CSE-C
185001183

Aim:

To write a program in C to find whether the given grammar is Left Recursive or not. If it is found to be left recursive, convert the grammar in such a way that the left recursion is removed.

Elimination of Left Recursion:

If we have the left-recursive pair of productions-

A → Aα | β (Left Recursive Grammar)

Then, we can eliminate left recursion by replacing the pair of productions with:

A → βA'

A' → αA' | ∈

(Right Recursive Grammar)

Code:

```
#include<stdio.h>
#include<string.h>

/* FUNCTIONS */
int getProds(char a[50][100]);
void postProds(char nt[],char prod[100][100],int n);
int prodStatus(char prod[],char prodRHS[100][100],char nt[],int k);
char getNT(char a[]);
int getProductionRHS(char a[],char prodRHS[100][100]);
int isLR(char nt,char prodRHS[100][100],int n);
int getBetaList(char nt,char prodRHS[100][100],int n,char beta[100][100]);
int getAlphaList(char nt,char prodRHS[100][100],int n,char alpha[100][100]);
void modifyBeta(char nt,char beta[100][100],char beta_mod[100][100],int n);
void modifyAlpha(char nt,char beta[100][100],char alpha_mod[100][100],int n);

//-------------------------------------------------------------------------------------

void main()
{
    char a[50][100];
    int n,status = 0;

    n = getProds(a);

    for(int i = 0;i<n;i++)
    {
        char nt[5];
        char prodRHS[100][100];
        int k;
        nt[0] = getNT(a[i]);
        nt[1] = '\0';
        k = getProductionRHS(a[i],prodRHS);

        if(status == 0)
            status = prodStatus(a[i],prodRHS,nt,k);
        else
            prodStatus(a[i],prodRHS,nt,k);
    }

    if(f == 1)
        printf("\n\t\t GRAMMAR IS LR\n");
    else
        printf("\n\t\t GRAMMAR IS GOOD TO GO\n");
```

```
    for(int i = 0; i < n; i++) //for each prod
    {
        char nt[3];
        char prodRHS[100][100],alpha[100][100],beta[100][100],alpha_mod[100][100],beta_mod[
100][100];
        int k,alpha_c,beta_c;
        int flag = 0;

        nt[0] = getNT(a[i]);
        nt[1] = '\0';
        k = getProductionRHS(a[i],prodRHS);

        if(isLR(nt[0],prodRHS,k))
        {
            alpha_c = getAlphaList(nt[0],prodRHS,k,alpha);
            beta_c = getBetaList(nt[0],prodRHS,k,beta);
            modifyBeta(nt[0],beta,beta_mod,beta_c);
            modifyAlpha(nt[0],alpha,alpha_mod,alpha_c);


            char newnt[5];
            n_nt[0] = nt[0];
            n_nt[1] = '\'';
            n_nt[2] = '\0';
            postProds(nt,beta_mod,beta_c);
            postProds(n_nt,alpha_mod,alpha_c+1);
        }
        else
        {
            postProds(nt,prodRHS,k);
        }

    }
}

//-----------------------------------------------------------------------------------------

/* DEFINED FUNCTIONS */


int getProds(char a[50][100])
{
    int n;
    printf("Enter the number of productions in the grammar: ");
    scanf("%d",&n);
    for(int i = 0; i<n;i++)
    {
        printf("Enter Production-%d:  ",i+1);
        scanf("%s",a[i]);
```

```
    }
    return n;
}

void postProds(char nt[],char prod[100][100],int n)
{
   printf("%s->%s",nt,prod[0]);
   for(int i = 1; i < n; i++)
   {
     printf("|%s",prod[i]);
   }
   printf("\n");
}

int prodStatus(char prod[],char prodRHS[100][100],char nt[],int k)
{
    int flag  = 0;
    if(isLR(nt[0],prodRHS,k))
    {
        printf("%s is LR\n",prod);
        flag = 1;
    }
    else
        printf("%s is not LR\n",prod);
    return flag;
}

char getNT(char a[])
{
  return a[0];
}

int getProductionRHS(char a[],char prodRHS[100][100])
{
  int i = 0;
  while(a[i]!='>') i++; //RHS
  int k = 0,t = 0,j = 0;
  char temp[100];
  while(a[i]!='\0'){
    i++;
    if(a[i]=='|'||a[i]=='\0')
    {
      temp[j] = '\0';
      strcpy(prodRHS[k++],temp);
      j =0;
      temp[j] = '\0';
    }
```

```
    else
    {
      temp[j++] = a[i];
    }
  }
  return k;
}

int isLR(char nt,char prodRHS[100][100],int n)
{
  for(int i = 0;i<n;i++)
  {
    if(prodRHS[i][0]==nt)
        return 1;
  }
  return 0;
}

int getBetaList(char nt,char prodRHS[100][100],int n,char beta[100][100])
{
   int k = 0;
   for(int i = 0;i<n;i++)
   {

     if(prodRHS[i][0]!=nt)
     {
       strcpy(beta[k],prodRHS[i]);
       //printf("BETA: %s\n",beta[k]);
       k++;
     }
   }
   return k;
}

int getAlphaList(char nt,char prodRHS[100][100],int n,char alpha[100][100])
{
  int k = 0;
  char temp[100];
   for(int i = 0;i<n;i++)
   {
     if(prodRHS[i][0]==nt)
     {
       int pos = 0,j = 1;
       temp[pos]='\0';
       while( j < strlen(prodRHS[i]) )
       {
           temp[pos] = prodRHS[i][j];
           pos++;
```

```
            j++;
        }
        temp[pos]='\0';
        strcpy(alpha[k],temp);
        //printf("ALPHA: %s\n",alpha[k]);
        k++;
      }
    }
    return k;
}

void modifyBeta(char nt,char beta[100][100],char beta_mod[100][100],int n)
{
  if(n!=0)
  {
    for(int i = 0;i < n ; i++)
    {
      char temp[100]="",temp1[5];
      strcat(temp,beta[i]);
      temp1[0]=nt;
      temp1[1]='\'';
      temp1[2]='\0';
      strcat(temp,temp1);
      strcpy(beta_mod[i],temp);
    }
  }
  else
  {
    char temp1[5];
    temp1[0]=nt;
    temp1[1]='\'';
    temp1[2]='\0';
    strcpy(beta_mod[0],temp1);
  }
}

void modifyAlpha(char nt,char alpha[100][100],char alpha_mod[100][100],int n)
{
  int i;
  for( i = 0;i < n ; i++)
  {
    char temp[100]="";
    strcat(temp,alpha[i]);
    int l = strlen(temp);
    temp[l]=nt;
    temp[l+1]='\'';
    temp[l+2] = '\0';
    strcpy(alpha_mod[i],temp);
```

```
  }
  strcpy(alpha_mod[i],"ep");
}
```

OUTPUT:

```
Admin@DESKTOP-1883PSF MINGW64 ~/OneDrive/Desktop
$ gcc eliV2.c -o a

Admin@DESKTOP-1883PSF MINGW64 ~/OneDrive/Desktop
$ ./a
Enter the number of productions in the grammar: 5
Enter Production-1:  E->E+T
Enter Production-2:  E->T
Enter Production-3:  T->T*F
Enter Production-4:  T->F
Enter Production-5:  F->a
E->E+T is LR
E->T is not LR
T->T*F is LR
T->F is not LR
F->a is not LR

                GRAMMAR IS LR
E->E'
E'->+TE'|ep
E->T
T->T'
T'->*FT'|ep
T->F
F->a
```

```
Admin@DESKTOP-1883PSF MINGW64 ~/OneDrive/Desktop
$ ./a
Enter the number of productions in the grammar: 3
Enter Production-1:  E->E+T|T
Enter Production-2:  T->T*F|F
Enter Production-3:  F->i
E->E+T|T is LR
T->T*F|F is LR
F->i is not LR

                GRAMMAR IS LR
E->TE'
E'->+TE'|ep
T->FT'
T'->*FT'|ep
F->i
```

```
Admin@DESKTOP-1883PSF MINGW64 ~/OneDrive/Desktop
$ ./a
Enter the number of productions in the grammar: 3
Enter Production-1:  A->a
Enter Production-2:  B->aaB
Enter Production-3:  A->B
A->a is not LR
B->aaB is not LR
A->B is not LR

                GRAMMAR IS GOOD TO GO
A->a
B->aaB
A->B
```

LEARNING OUTCOME:

- Learnt that a production of grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS.
- Understood the need for this type of conversion, as top-down parsers cannot handle left recursive grammars.
- Strengthened my knowledge and skills in string operations and to parse each production in input to check if its left recursive or not.
- Learnt to modularise long code into functions and follow the best practices.
- Was able to perform a check of whether or not a grammar is left recursive using C.

RESULT:

Successfully implemented the code to check whether the given grammar is Left Recursive or not and convert the grammar in such a way that the left recursion is removed.