**Compiler Lab Report**

**Experiment: 05**

**Submitted By**

Md. Mushfekur Rahman

Roll – 01

15th Batch

Dept. of Computer Science & Engineering

University of Dhaka

**Name of the Experiment**

Construction of Minimum State DFA from given Regular Expression and Correctness Evaluation with Test Strings

**Source Code**

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

#define MAXN 50

#define isopt(a) ((a=='+') || (a=='\*') || (a=='.'))

#define isunary(a) (a=='\*')

#define isleaf(a) ((a->left == NULL) && (a->right==NULL))

#define isalpha(a) ((a>=65 && a<=90) || (a>=97 && a<=122) || (a=='#'))

#define present strstr

typedef struct pos {

int \*data;

int n;

} pos;

pos \*followpos, \*dfa[MAXN];

typedef struct node {

char data;

int num;

pos fpos, lpos;

struct node \*left, \*right;

} node;

node \*Tree;

typedef struct mystack {

int top;

node \*data[MAXN];

} mystack;

char \*terminals, \*sym;

int leaves, states, sym\_n;

void init(mystack \*a);

void push(mystack \*a, node \*data);

node\* pop(mystack \*a);

char\* re2post(char \*re);

node\* createTree(char \*str);

int isNullable(node \*a);

int firstpos(node \*n,int \*\*set);

int lastpos(node \*n,int \*\*set);

void follpos(node \*root);

int Union(int \*\*res,int \*arr1,int \*arr2,int n1,int n2);

int inStates(pos a);

void postfix(node \*a);

void printState(pos a);

void re2dfa(node \*root);

int main()

{

// freopen("in.txt","r",stdin);

// freopen("out.txt","w",stdout);

char in[MAXN], \*post;

int i, j;

printf("Enter a RE: ");

scanf("%s",in);

post = re2post( in );

printf("\nPostfix: "); puts( post );

Tree = createTree( post );

printf("\n\n%s\t%s\t%s\t%s\n","No","Node","FirstPos","LastPost");

postfix( Tree );

follpos( Tree );

printf("\n\n\n\nNo.\tFollowPos\n");

for(i=1;i<=leaves;i++)

{

if(followpos[i].n)

{

printf("\n%d\t{",i);

for(j=0;j<followpos[i].n;j++)

printf("%d,",followpos[i].data[j]);

printf("\b}");

}

}

printf("\n\n\n\nRE to DFA Table\n");

printf("\nState\t");

for(i=0;i<sym\_n;i++)

printf("%2c\t",sym[i]);

printf("\b\n");

re2dfa( Tree );

getchar();

return 0;

}

void init(mystack \*a) {

a->top = -1;

}

void push(mystack \*a, node \*data) {

a->top++;

a->data[a->top] = data;

}

node\* pop(mystack \*a) {

if(a->top < 0) return NULL;

return a->data[a->top--];

}

char\* re2post(char \*re)

{

int nalt, natom;

static char buf[8000];

char \*dst;

struct {

int nalt, natom;

} paren[100], \*p;

p = paren;

dst = buf;

nalt = natom = 0;

if(strlen(re) >= sizeof(buf)/2) return NULL;

for(; \*re; re++) {

switch( \*re ) {

case '(':

if(natom > 1) {

--natom;

\*dst++ = '.';

}

if(p >= paren+100) return NULL;

p->nalt = nalt;

p->natom = natom;

p++;

nalt = natom = 0;

break;

case '+':

if( !natom ) return NULL;

while(--natom > 0) \*dst++ = '.';

nalt++;

break;

case ')':

if(p == paren) return NULL;

if( !natom ) return NULL;

while(--natom > 0) \*dst++ = '.';

for(; nalt > 0; nalt--) \*dst++ = '+';

--p;

nalt = p->nalt;

natom = p->natom;

natom++;

break;

case '\*':

case '?':

if( !natom ) return NULL;

\*dst++ = \*re;

break;

default:

if(natom > 1) {

--natom;

\*dst++ = '.';

}

\*dst++ = \*re;

natom++;

break;

}

}

if(p != paren) return NULL;

while(--natom > 0) \*dst++ = '.';

for(; nalt>0; nalt--) \*dst++ = '+';

\*dst++ = '#';

\*dst++ = '.';

\*dst = 0;

return buf;

}

void postfix(node \*a)

{

int i;

static int leaf = 1;

if(a == NULL) return;

postfix( a->left );

postfix( a->right );

printf("\n%-2d\t%-4c\t{",(isleaf(a) ? leaf++ :0),a->data);

for(i=0; i<a->fpos.n; i++) printf("%d,",a->fpos.data[i]);

printf("\b}\t\t{");

for(i=0; i<a->lpos.n; i++) printf("%d,",a->lpos.data[i]);

printf("\b}");

}

int Union(int \*\*uni,int \*arr1,int \*arr2,int n1,int n2)

{

int i = 0, k = 0, j = 0;

int \*res;

res = (int \*)malloc(sizeof(int)\*(n1+n2));

while(i<n1 && j<n2) {

if(arr1[i] < arr2[j]) res[k++] = arr1[i++];

else if(arr1[i] > arr2[j]) res[k++] = arr2[j++];

else {

res[k++] = arr1[i++];

j++;

}

}

while(i < n1) res[k++] = arr1[i++];

while(j < n2) res[k++] = arr2[j++];

\*uni = res;

return k;

}

node\* createTree(char \*str)

{

int i = 0, j, num = 1;

node \*nd;

char a, temp[2];

mystack s;

init( &s );

temp[1] = 0;

leaves = 0;

while( str[i] ) {

a = str[i];

nd = (node \*)malloc(sizeof(node));

nd->left = nd->right = NULL;

nd->data = a;

nd->num = 0;

if( isalpha(a) ) {

nd->num = num++;

nd->fpos.n = firstpos(nd,&(nd->fpos.data));

nd->lpos.n = lastpos(nd,&(nd->lpos.data));

push(&s, nd);

i++; leaves++;

continue;

}

if( isopt(a) ) {

if( isunary(a) ) {

nd->left = pop( &s );

} else {

nd->right = pop( &s );

nd->left = pop( &s );

}

nd->fpos.n = firstpos(nd, &(nd->fpos.data));

nd->lpos.n = lastpos(nd, &(nd->lpos.data));

push(&s, nd);

i++;

}

}

sym = (char \*)calloc(sizeof(char),leaves+2);

terminals = (char \*)calloc(sizeof(char),leaves+2);

i = sym\_n = 0;

j = 1;

terminals[0] = ' ';

while( str[i] ) {

temp[0] = str[i];

if( isalpha(str[i]) ) {

terminals[j++] = str[i];

if( !present(sym, temp) ) sym[sym\_n++] = temp[0];

}

i++;

}

sym\_n--;

followpos = (pos \*)calloc(sizeof(pos),(leaves+1));

return pop( &s );

}

int isNullable(node \*a)

{

if(a == NULL) return 0;

if(a->data == '\*') return 1;

if((a->left==NULL) && (a->right==NULL)) return 0;

if(a->data == '+') return((isNullable(a->left)) || (isNullable(a->right)));

if(a->data == '.') return((isNullable(a->left)) && (isNullable(a->right)));

return 0;

}

int firstpos(node \*n,int \*\*set)

{

int k, n1, n2;

int \*arr1, \*arr2, \*res;

k = 0;

if( n->num ) {

arr1 = (int \*)malloc(sizeof(int)\*2);

arr1[0] = n->num;

\*set = arr1;

return 1;

}

if(n->data == '+') {

n1 = firstpos(n->left, &arr1);

n2 = firstpos(n->right, &arr2);

k = Union(&res, arr1, arr2, n1, n2);

free( arr1 );

free( arr2 );

\*set = res;

return k;

}

if(n->data == '.') {

n1 = firstpos(n->left,&arr1);

if( isNullable(n->left) ) {

n2 = firstpos(n->right,&arr2);

k = Union(&res,arr1,arr2,n1,n2);

\*set = res;

free( arr1 );

free( arr2 );

return k;

} else {

\*set = arr1;

return n1;

}

}

if(n->data == '\*') {

n1 = firstpos(n->left, &arr1);

\*set = arr1;

return n1;

}

\*set = NULL;

return 0;

}

int lastpos(node \*n, int \*\*set)

{

int k, n1, n2;

int \*arr1, \*arr2, \*res;

k = 0;

if( n->num ) {

arr1 = (int \*)malloc(sizeof(int)\*2);

arr1[0] = n->num;

\*set = arr1;

return 1;

}

if(n->data == '+') {

n1 = lastpos(n->left, &arr1);

n2 = lastpos(n->right, &arr2);

k = Union(&res, arr1, arr2, n1, n2);

\*set = res;

free( arr1 );

free( arr2 );

return k;

}

if(n->data == '.') {

n2 = lastpos(n->right, &arr2);

if( isNullable(n->right) ) {

n1 = lastpos(n->left, &arr1);

k = Union(&res, arr1, arr2, n1, n2);

\*set = res;

free( arr1 );

free( arr2 );

return k;

} else {

\*set = arr2;

return n2;

}

}

if(n->data == '\*') {

n1 = lastpos(n->left, &arr1);

\*set = arr1;

return n1;

}

\*set = NULL;

return 0;

}

void follpos(node \*root)

{

int i, n, num;

int \*res;

node \*a, \*b;

if(root == NULL) return;

a=root->left;

b=root->right;

if(root->data == '.') {

for(i=0; i<a->lpos.n; i++) {

num = a->lpos.data[i];

n = Union(&res, followpos[num].data, b->fpos.data, followpos[num].n, b->fpos.n);

free( followpos[num].data );

followpos[num].n = n;

followpos[num].data = res;

}

}

if(root->data == '\*') {

for(i=0; i<root->lpos.n; i++) {

num = root->lpos.data[i];

res = (int \*)malloc(sizeof(int)\*(followpos[num].n + root->fpos.n));

n = Union(&res, followpos[num].data, root->fpos.data, followpos[num].n, root->fpos.n);

free(followpos[num].data);

followpos[num].n = n;

followpos[num].data = res;

}

}

follpos( a );

follpos( b );

}

int inStates(pos a)

{

int i, j, same;

for(i=0; i<states; i++) {

if(a.n != dfa[i][0].n) continue;

same = 1;

for(j=0; j<a.n; j++) {

if(a.data[j] != dfa[i][0].data[j]) {

same = 0; break;

}

}

if( same ) return 1;

}

return 0;

}

void printState(pos a) {

for(int i=0;i<a.n;i++)

printf("%d",a.data[i]);

}

void re2dfa(node \*root)

{

int i=0, j, loc, k, n;

int \*res;

char temp[2];

states = 1;

dfa[i] = (pos \*)calloc(sizeof(pos),sym\_n+1);

dfa[i][0].n = root->fpos.n;

dfa[i][0].data = root->fpos.data;

temp[1] = 0;

while(i < states) {

j = 0; k = 1;

while(j < dfa[i][0].n) {

k = dfa[i][0].data[j];

temp[0] = terminals[k];

if(terminals[k] == '#') {

j++; continue;

}

loc = present(sym,temp) - sym;

loc++;

n = Union(&res, followpos[k].data, dfa[i][loc].data, followpos[k].n, dfa[i][loc].n);

dfa[i][loc].data = res;

dfa[i][loc].n = n;

j++;

}

printf("\n");

printState( dfa[i][0] );

printf("\t");

for(j=1; j<=sym\_n; j++) {

if( !inStates(dfa[i][j]) ) {

dfa[states] = (pos \*)calloc(sizeof(pos),sym\_n+1);

dfa[states][0].n = dfa[i][j].n;

dfa[states][0].data = dfa[i][j].data;

states++;

}

printState(dfa[i][j]);

printf("\t");

}

printf("\b");

i++;

}

}