College Of Engineering Trivandrum

Compiler Design Lab

CS431

Submitted By:
Neethu S
S6 CSE Roll No:42

TVE18CS043

Department of Computer Science

October 1, 2021

Contents

CS431 - Compiler Design Lab · 2021 ·

1 ϵ - closure

1.1 Aim

To write a program to find ϵ – closure of all states of any given NFA with ϵ transition.

1.2 Theory

An ϵ-NFA is represented formally by a 5-tuple, (Q, Σ, Δ, q0, F), consisting of

- a finite set of states Q
- a finite set of input symbols Σ
- a transition function Δ : Q $\times$ (Σ U ϵ)$\rightarrow$ P(Q)
- an initial (or start) state q0 ϵ Q
- a set of states F distinguished as accepting (or final) states F $\subseteq$ Q

Here, P(Q) denotes power set of Q.

The ϵ closure(P) is a set of states which are reachable from state P on ϵ-transitions.

1.3 Algorithm

1. For each transition
   (a) Read transition S1, S2, a where S1 is the start state, S2 is the destination state and a is the input symbol
   (b) If S1 doesnt́ exist create state S1
   (c) If S2 doesnt́ exist create state S2
   (d) Create transition T with S2 and a and append it to S1ś transition list
2. For each state S
   (a) For each transition T from S
      i. If a of T is "ϵ", print S2
      ii. Repeat step 2.(a) for S2

1.4 Code

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

typedef struct Transition Transition_t;
typedef struct State State_t;

struct State {
    char name;
    Transition_t *transitionListHead;
```

```
    State_t *next;
};

struct Transition {
    State_t *nextState;
    char symbol;
    Transition_t *next;
};

State_t*
createState(char symbol) {
    State_t *s = malloc(sizeof(State_t));

    s->name = symbol;
    s->next = NULL;
    s->transitionListHead = NULL;

    return s;
}

State_t*
lookupOrCreateState(State_t *HEAD, char symbol) {
    State_t *prevState = HEAD;

    while(prevState->name != symbol && prevState->next != NULL) {
        prevState = prevState->next;
    }

    if(prevState->name == symbol) {
        return prevState;
    }

    State_t *s = createState(symbol);
    prevState->next = s;

    return s;
}

Transition_t*
createTransition(State_t *dest, char symbol) {
    Transition_t *t = malloc(sizeof(Transition_t));

    t->nextState = dest;
    t->symbol = symbol;
    t->next = NULL;

    return t;
}

State_t*
addTransition(State_t *HEAD, char sourceState, char destState, char inputSymbol) {
    State_t *source = NULL, *dest = NULL;
```

```
    Transition_t *prevTransition, *t;

    if(HEAD == NULL) {
        source = createState(sourceState);
        dest = createState(destState);

        HEAD = source;
        HEAD->next = dest;
    } else {
        source = lookupOrCreateState(HEAD, sourceState);
        dest = lookupOrCreateState(HEAD, destState);
    }

    if(source == NULL || dest == NULL) {
        fprintf(stderr, "Failed to add transition\n");
        exit(0);
    }

    t = createTransition(dest, inputSymbol);

    if(source->transitionListHead == NULL) {
        source->transitionListHead = t;
    }
    else {
        prevTransition = source->transitionListHead;
        while(prevTransition->next != NULL) {
            prevTransition = prevTransition->next;
        }

        prevTransition->next = t;
    }

    return HEAD;
}

void
printEpsilonTransitionHelper(State_t *node) {
    // TODO: Remove infinite loop on cycles
    Transition_t *t = node->transitionListHead;

    while(t != NULL) {
        if(t->symbol == 'e') {
            fprintf(stdout, "%c, ", t->nextState->name);
            printEpsilonTransitionHelper(t->nextState);
        }

        t = t->next;
    }
}

void
printEpsilonTransitions(State_t *node) {
```

```
    if(node == NULL) return;

    fprintf(stdout, "%c: { %c, ", node->name, node->name);
    printEpsilonTransitionHelper(node);
    fprintf(stdout, "}\n");

    printEpsilonTransitions(node->next);
}

void
freeTransitions(Transition_t *node) {
    if(node == NULL) return;
    freeTransitions(node->next);
    free(node);
}

void
freeMemory(State_t *node) {
    if(node == NULL) return;

    if(node->next != NULL) return freeMemory(node->next);

    freeTransitions(node->transitionListHead);
    free(node);
}

int main() {
    int noOfTransitions;
    char sourceState, destState, inputSymbol;
    State_t *HEAD = NULL;

    fprintf(stdout, "Enter number of transitions: ");
    scanf("%d", &noOfTransitions);

    fprintf(
        stdout,
        "Enter transition in the format\n"
        "Start State ---- End State ---- Symbol\n"

    );

    while(noOfTransitions--) {
        scanf(" %c", &sourceState);
        scanf(" %c", &destState);
        scanf(" %c", &inputSymbol);
        HEAD = addTransition(HEAD, sourceState, destState, inputSymbol);
    }

    printEpsilonTransitions(HEAD);
    freeMemory(HEAD);
}
```

1.5 Output

```
s1842@administrator-rusa:~/s7$ cc e1.c
s1842@administrator-rusa:~/s7$ ./a.out
Enter number of transitions: 4
Enter transition in the format
Start State ---- End State ---- Symbol
A B e
A C e
B D e
C D e
A: { A, B, D, C, D, }
B: { B, D, }
C: { C, D, }
D: { D, }
s1842@administrator-rusa:~/s7$
```

1.6 Result

Implemented a program to find ϵ – closure of all states of any given NFA with ϵ transition using C in Ubuntu 20.04 and the above outputs were obtained.