Отчет по лабораторной работе №1

по курсу “Конструирование компиляторов”

на тему “Распознавание цепочек регулярного языка”

Вариант №3

Выполнила:

Уточкина Наталия, ИУ7-22М

05.09.2020

Цель работы: приобретение практических навыков реализации важнейших элементов лексических анализаторов на примере распознавания цепочек регулярного языка.

Задачи работы:

1) Ознакомиться с основными понятиями и определениями, лежащими в основе построения лексических анализаторов.
2) Прояснить связь между регулярным множеством, регулярным выражением, праволинейным языком, конечно- автоматным языком и недетерминированным конечно-автоматным языком.
3) Разработать, тестировать и отладить программу распознавания цепочек регулярного или праволинейного языка в соответствии с предложенным вариантом грамматики.

Задание к варианту №3

Напишите программу, которая в качестве входа принимает произвольное регулярное выражение, и выполняет следующие преобразования:

1) По регулярному выражению строит НКА.
2) По НКА строит эквивалентный ему ДКА.
3) По ДКА строит эквивалентный ему КА, имеющий наименьшее возможное количество состояний.
4) Моделирует минимальный КА для входной цепочки из терминалов исходной грамматики.

Текст программы:

Интерфейс состояния и переходов:

```
export interface ITransition {

    symbol: string;

    next_state: string;

}

export interface IFAState {

    states: string[];

    transitions: Map<string, ITransition[]>;

    begin_state: string;

    end_states: string[];

}
```

Построение по регулярному выражению НКА.

Функция создания состояния

```
private createState():string {

        let new_state = `q${this.states.length}`;

        if (this.states.indexOf(new_state) !== -1) {

            new_state = `q${this.states.length + this.states.length}`;

        }

        this.states.push(new_state);



        return new_state;

}
```

Функция создания перехода по символу

```
private fromSymbol(symbol?:string) {

        const begin_state = this.createState();

        const next_state = this.createState();

        this.createTransition(begin_state, next_state, symbol);



        this.stack.push({

            states: new Array(...this.states),

            transitions: new Map(this.transitions),

            begin_state: begin_state,

            end_states: [next_state]

        });

}
```

Функция создания замыкания

```
private closure(state:IFAState) {

        const start_state = this.createState();

        const end_state = this.createState();



        this.createTransition(start_state, state.begin_state);

        this.createTransition(state.end_states[0], end_state);

        this.createTransition(start_state, end_state);

        this.createTransition(state.end_states[0], state.begin_state);

        this.stack.push({

            states: new Array(...this.states),

            transitions: new Map(this.transitions),

            begin_state: start_state,

            end_states: [end_state]

        });

}
```

Функция ИЛИ

```
private union(begin:IFAState, end:IFAState) {

        const begin_left = begin.begin_state;

        const begin_right = begin.end_states[0];

        const end_left = end.begin_state;

        const end_right = end.end_states[0];



        const new_states = [`${begin_left}${end_left}`,
`${begin_right}${end_right}`];



        const begin_left_transitions = begin.transitions.get(begin_left);
```

```
        const end_left_transitions = end.transitions.get(end_left);



        this.transitions.delete(begin_left);

        this.transitions.delete(end_left);



        this.transitions.set(new_states[0], [...begin_left_transitions!,
...end_left_transitions!]);



        const new_transitions = new Map();

        for (let [key, value] of this.transitions) {

            let new_value:ITransition[] = [];

            value?.map((transition:ITransition) => {

                if (transition.next_state === begin_right || transition.next_state
=== end_right) {

                    new_value.push({

                        symbol: transition.symbol,

                        next_state: new_states[1]

                    });

                } else {

                    new_value.push(transition);

                }

            });

            new_transitions.set(key, new_value);

        }

        this.transitions = new_transitions;

        this.states.push(...new_states);
```

```
        this.states = this.states.filter((state) => [begin_left, begin_right,
end_left, end_right].indexOf(state) === -1);



        this.stack.push({

            states: new Array(...this.states),

            transitions: new Map(this.transitions),

            begin_state: new_states[0],

            end_states: [new_states[1]]

        });

    }
```

Функция И

```
private concat(begin:IFAState, end:IFAState) {

        const start_of_concat = begin.end_states[0];

        const end_of_concat = end.begin_state;



        const transition = this.transitions.get(end_of_concat)!;

        this.transitions.delete(end_of_concat);

        this.transitions.set(start_of_concat, transition);

        this.states = this.states.filter((state) => state !== end_of_concat);

        this.stack.push({

            states: new Array(...this.states),

            transitions: new Map(this.transitions),

            begin_state: begin.begin_state,

            end_states: end.end_states

        });

 }
```

Функция обработки входной строки:

```
if (regex === '' || !regex) {
            this.fromSymbol();
} else {
            for (const token of regex) {
                switch (token) {
                    case "*": {
                        this.closure(this.stack.pop()!);
                        break;
                    }
                    case "+": {
                        const end = this.stack.pop()!;
                        const begin = this.stack.pop()!;
                        this.union(begin, end);
                        break;
                    }
                    case ".": {
                        const end = this.stack.pop()!;
                        const begin = this.stack.pop()!;
                        this.concat(begin, end);
                        break;
                    }
                    default: {
                        this.fromSymbol(token);
                        break;
                    }
                }
            }
 }
```

Построение ДКА по НКА:

Функция поиска состояний, достижимых из состояний, входящих в R, только по eps переходам:

```
private e_closure(states:string[]):string[] {
        const result = [...states];

        let states_for_detour = [...states];
        states_for_detour.forEach((current_state) => {
            const transitions = this.transitions.get(current_state);
            transitions?.forEach((value) => {
                if (value.symbol === this.epsilon &&
result.indexOf(value.next_state) === -1) {
                    result.push(value.next_state);
                    states_for_detour.push(value.next_state);
                }
            });
            states_for_detour = states_for_detour.filter((state) => state !==
current_state);
        });
```

```
        return result;
    }
```

Функция поиска состояний, достижимых из состояний, входящих в R, по a переходам, где a - символ алфавита:

```
private move(states:string[], a:string):string[] {
        const result:string[] = [];

        states.forEach((current_state) => {
            const transitions = this.transitions.get(current_state);
            transitions?.forEach((value) => {
                if (value.symbol === a) {
                    result.push(value.next_state);
                }
            });
        });

        return result;
    }
```

Функция перевода НКА в ДКА:

```
public toDFA() {
        if (this.dfa) {
            return;
        }

        const nfa = this.getNFA();
        let begin_state = this.e_closure([nfa.begin_state]).sort();
        const Q = [begin_state];
        const D = new Map<string, ITransition[]>();
        let Q_dynamic = [...Q];

        while (Q_dynamic.length !== 0) {
            const current_q = Q_dynamic[0];
            for (let i = 0; i < this.alphabet.length; i++) {
                const S = this.e_closure(this.move(current_q,
this.alphabet[i])).sort();
                if (S.length) {
                    let push:boolean = true;
                    Q.forEach((value) => {
                        if (this.compare(value, S)) {
                            push = false;
                        }
                    });

                    if (push) {
                        Q.push(S);
                        Q_dynamic.push(S);
```

```
                    }
                    const R = current_q.join("");
                    const R_transitions = D.get(R);
                    R_transitions?.push({ symbol: this.alphabet[i], next_state:
S.join("") });
                    D.set(R, R_transitions
                        ? R_transitions
                        : [{ symbol: this.alphabet[i], next_state: S.join("") }]
                    );
                }
            }
            Q_dynamic = Q_dynamic.filter((q) => !this.compare(q, current_q));
        }

        const new_states = Q.map((q) => q.join(""));
        this.dfa = {
            begin_state: begin_state.join(""),
            states: new_states,
            transitions: D,
            end_states: new_states.filter((q) =>
q.includes(this.stack[this.stack.length - 1].end_states[0]))
        };
    }
```

Минимизация алгоритмом Хопкрофта

```
public minimize() {
        const p0 = this.dfa?.states.filter((q) => this.dfa?.end_states.indexOf(q)
!== -1);
        const p1 = this.dfa?.states.filter((q) => this.dfa?.end_states.indexOf(q)
=== -1);
        let partition = [p0, p1];
        let nextP = [p0, p1];
        let worklist = [p0, p1];

        while (worklist.length !== 0) {
            const s:string[] = worklist[0]!;
            worklist.shift();
            for (let i = 0; i < this.alphabet.length; i++) {
                // Image ← {x | δ(x,c) ∈ s}
                const image:string[] = [];
                this.dfa?.transitions.forEach((transition, key) => {
                    transition.forEach((q) => {
                        if (q.symbol === this.alphabet[i] &&
s.indexOf(q.next_state) !== -1) {
                            image.push(key);
                        }
                    })
                });

                partition.forEach((q) => {
```

```
                    if (q) {
                        const q1 = q.filter((current_q) =>
image.indexOf(current_q) !== -1);
                        const q2 = q.filter((current_q) => q1?.indexOf(current_q)
=== -1);

                        if (q1.length !== 0 && q2.length !== 0) {
                            partition = partition.filter((state) => state &&
!this.compare(state, q));
                            nextP = nextP.filter((state) => state &&
!this.compare(state, q));

                            nextP.push(q1, q2);

                            let is_q_in_worklist = false;
                            worklist.forEach((s) => {
                                if (this.compare(s!, q)) {
                                    is_q_in_worklist = true;
                                }
                            });
                            if (is_q_in_worklist) {
                                worklist.filter((s) => !this.compare(s!, q));
                                worklist.push(q1, q2);
                            } else if (q1.length <= q2.length) {
                                worklist.push(q1);
                            } else {
                                worklist.push(q2);
                            }

                            if (this.compare(s, q)) {
                                return;
                            }
                        }
                    }
                });

                partition = [...nextP];
            }
        }


 // П о с т р о е н и е  н о в ы х  п е р е х о д о в
        const new_transitions = new Map<string, ITransition[]>();
        const new_states:string[] = [];

        partition.forEach((new_state) => {
            if (new_state) {
                const new_state_str = new_state.join("");

                new_states.push(new_state_str);
```

```
                if (new_state.length === 1) {
                    const old_transition =
this.dfa?.transitions.get(new_state_str);
                    if (old_transition) {
                        new_transitions.set(new_state_str, old_transition);
                    }
                } else {
                    const started_transition:ITransition[] = [];

                    new_state.forEach((old_state) => {
                        // п о и с к  с т е й т о в  к о т о р ы е
 н а ч и н а ю т с я  н а old_state
                        this.dfa?.transitions.forEach((transition, key) => {
                            let same:boolean = false;
                            started_transition.forEach((t) => {
                                transition.forEach((q) => {
                                    if (q.symbol === t.symbol && q.next_state ===
t.next_state) {
                                        same = true;
                                    }
                                })
                            });
                            if (key === old_state && !same) {
                                started_transition.push(...transition);
                            }
                        });
                    });

                    if (started_transition.length !== 0) {
                        new_transitions.set(new_state_str, started_transition);
                    }
                }
            }
        });


        const new_transitions_with_right_ends = new Map<string, ITransition[]>();
        new_transitions.forEach((transition, key) => {
            let new_transition:ITransition[] = [];

            transition.forEach((q) => {
                if (new_states.indexOf(q.next_state) !== -1) {
                    new_transition.push(q);
                } else {
                    let new_state:string | undefined;

                    partition.forEach((part) => {
                        if (part && part.indexOf(q.next_state) !== -1) {
                            new_state = part.join("");
                            return;
                        }
```

```
                    });

                    if (new_state) {
                        new_transition.push({ symbol: q.symbol, next_state:
new_state });
                    }
                }
            });

            new_transitions_with_right_ends.set(key, new_transition);
        });

        let begin_state:string | undefined;
        const end_states:string[] = [];
        if (new_states.indexOf(this.dfa!.begin_state) !== -1) {
            begin_state = this.dfa!.begin_state;
        } else {
            partition.forEach((part) => {
                if (part && part.indexOf(this.dfa!.begin_state) !== -1) {
                    begin_state = part.join("");
                    return;
                }
            });
        }

        this.dfa!.end_states.forEach((end_state) => {
            if (new_states.indexOf(end_state) !== -1) {
                end_states.push(end_state);
            } else {
                partition.forEach((part) => {
                    const crossing = this.dfa?.end_states.filter((current_q) =>
part && part.indexOf(current_q) !== -1);
                    if (part && crossing?.length &&
end_states.indexOf(part.join("")) === -1) {
                        end_states.push(part.join(""));
                        return;
                    }
                });
            }
        });

        this.minimized = {
            begin_state: begin_state!,
            end_states: end_states,
            transitions: new_transitions_with_right_ends,
            states: new_states
        };
    }
```

Набор тестов и ожидаемые результаты для проверки правильности программы.

```
"с т р о и т  и  м и н и м и з и р у е т  a"
      "a" - true
      "b" - false
      "aabb" - false
      "" - false
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  a+b"
      "a" - true
      "b" - true
      "aabb" - false
      "" - false
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  ab"
     "ab" - true
      "b" - false
      "aabb" - false
      "" - false
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  a*"
      "a" - true
      "aaaaaaaa" - true
      "b" - false
      "" - true
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  (ab)*"
      "ab" - true
      "abababab" - true
      "aabb" - false
      "" - true
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  (a+b)*"
      "aaaabbbbb" - true
      "b" - true
      "bbbabbababbababba" - true
      "aabbxx" - false
      "" - true
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  ab*a"
      "aa" - true
      "aba" - true
      "abbbbbbbbbba" - true
```

```
      "aabbxx" - false
      "" - false
      "nnnnn" - false

"с т р о и т  и  м и н и м и з и р у е т  abb+acb  и  п р о в е р я е т  с т р о к и:"
      "abb" - true
      "acb" - true
      "a" - false
      "ab" - false
      "" - false
      "nnn" - false

"с т р о и т  и  м и н и м и з и р у е т  (a+b)*abb  и  п р о в е р я е т  с т р о к и:"
      "aaabb" - true
       "bbbbbbbbbbabb" - true
       "abb" - true
       "nnnn" - false
       "" - false

"с т р о и т  и  м и н и м и з и р у е т  a(a+b)*  и  п р о в е р я е т  с т р о к и:"
      "aaa" - true
       "a" - true
       "abb" - true
       "aaaabbbb" - true
       "" - false
      “babab” - false
   });

"с т р о и т  и  м и н и м и з и р у е т  aa*+bb*  и  п р о в е р я е т  с т р о к и:"
      "aaa" - true
       "a" - true
       "b" - true
       "baaaaa" - false
       "aaaabbbb" - false
       "" - false

"с т р о и т  и  м и н и м и з и р у е т  (a+b)*abb*  и  п р о в е р я е т
с т р о к и:"
      "ab" - true
       "abb" - true
       "aaaabbbb" - true
       "baaaaa" - false
       "abbba" - false
       "" - false

 с т р о и т  и  м и н и м и з и р у е т  (a+b)*abb(a+b)*  и  п р о в е р я е т
с т р о к и:
      "abb" - true
       "aabbb" - true
       "aaaabbbb" - true
```

```
"bbbbabbbbbb" - true
"abababbbbbba" - true
"" - false
```

Пример работы программы:

Insert regex:

(a+b)*abb submit

ab+*a.b.b.

NFA:
STATES: q0q2,q1q3,q2,q3,q5,q6,q7
TRANSITIONS:

q0q2 - a - q1q3

q0q2 - b - q1q3

q2 - eps - q0q2

q2 - eps - q3

q1q3 - eps - q3

q1q3 - eps - q0q2

q3 - a - q5

q5 - b - q6

q6 - b - q7

BEGIN STATE: q2
END STATES: q7

DFA:
STATES:
q0q2q2q3,q0q2q1q3q3q5,q0q2q1q3q3,q0q2q1q3q3q6,q0q2q1q3q3q7
TRANSITIONS:

q0q2q2q3 - a - q0q2q1q3q3q5

q0q2q2q3 - b - q0q2q1q3q3

q0q2q1q3q3q5 - a - q0q2q1q3q3q5

q0q2q1q3q3q5 - b - q0q2q1q3q3q6

q0q2q1q3q3 - a - q0q2q1q3q3q5

q0q2q1q3q3 - b - q0q2q1q3q3

q0q2q1q3q3q6 - a - q0q2q1q3q3q5

q0q2q1q3q3q6 - b - q0q2q1q3q3q7

q0q2q1q3q3q7 - a - q0q2q1q3q3q5

q0q2q1q3q3q7 - b - q0q2q1q3q3

BEGIN STATE: q0q2q2q3
END STATES: q0q2q1q3q3q7

MINIMIZED:
STATES:
q0q2q1q3q3q7,q0q2q1q3q3q6,q0q2q1q3q3q5,q0q2q2q3q0q2q1q3q3
TRANSITIONS:

q0q2q1q3q3q7 - a - q0q2q1q3q3q5

q0q2q1q3q3q7 - b -
q0q2q2q3q0q2q1q3q3

q0q2q1q3q3q6 - a - q0q2q1q3q3q5

q0q2q1q3q3q6 - b - q0q2q1q3q3q7

q0q2q1q3q3q5 - a - q0q2q1q3q3q5

q0q2q1q3q3q5 - b - q0q2q1q3q3q6

q0q2q2q3q0q2q1q3q3 - a -
q0q2q1q3q3q5

q0q2q2q3q0q2q1q3q3 - b -
q0q2q2q3q0q2q1q3q3

BEGIN STATE: q0q2q2q3q0q2q1q3q3
END STATES: q0q2q1q3q3q7

Insert string for check:

ababb submit

Result: true

Вывод:

Программа показала работоспособность на тестовых данных. В ходе работы получены знания об алгоритмах построения НКА, ДКА и минимизации ДКА методом Хопкрофта и получены практические навыки их реализации.

Список литературы:

1) Engineering a Compiler - Keith D. Cooper, Linda Torczon., - Rice University Houston, Texas, 2017.

2) http://cmcstuff.esyr.org/vmkbotva-r15/2%20%D0%BA%D1%83%D1%80%D1%81/4%20%D0%A1%D0%B5%D0%BC%D0%B5%D1%81%D1%82%D1%80/%D0%9F%D1%80%D0%B0%D0%BA/%D0%94%D0%B7/regexp.pdf

3) http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_Хопкрофта_(сложность_O(n_log_n))