 

**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **编译技术** |
| **学生姓名：** |  |
| **学生学号：** |  |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2024-2025第二学期** |

**软件学院**

**2025年05月**

# From NFAs to DFAs

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **地 点：** | | | B7 楼 | | 333 房； | | **实验台号：** | |  | |
| **实验日期与时间：** | | | 2025/5/16 | | | | **评 分：** | |  | |
| **预习检查纪录：** | | |  | | | | **实验教师：** | | 应伟勤 | |
| **电子文档存放位置：** | | |  | | | | | | | |
| **电子文档文件名：** | | |  | | | | | | | |
| **批改意见：** | | |  | | | | | | | |
| 出勤情况（20%） | 课堂态度（20%） | | 实验报告质量（60%） | | | | | |
| 任务完成情况 | | 源代码注释 | | 报告内容 | |
| □ 出勤  □ 缺勤  □ 早退  □ 请假 | □ 认真  □ 不认真 | | □ 完成  □ 未完成 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | |

# 报告内容

1、实验目标

（1）使用Subset Construction算法实现NFA转换为DFA

（2）使用state minimization算法实现DFA最小化

2、实验环境

IDEA，Vscode

3、关键数据结构和核心算法

3.1 数据结构

3. 2 核心算法

(1) epsilonClosures

    private HashMap<Integer, State> epsilonClosures(State s, LabeledDirectedGraph<State> tb){

        if (!tb.vertexSet().contains(s)) { //if vertex s not in the transition table

                return null;

        }

        HashMap<Integer,State> nfaStates = new HashMap<>();

        ArrayDeque<State> queue = new ArrayDeque<>();

        queue.add(s);

        while(!queue.isEmpty()) {

            State state = queue.poll();

            nfaStates.putIfAbsent(state.getId(), state);

            //add each new state reached by an edge from ‘state’ on 'ε' to ‘queue’

            for (LabelEdge e : tb.edgeSet()) {

                //Add your implementation

                /\*

                补充代码

                 \*/

                if(tb.getEdgeSource(e).equals(state)&&e.getLabel()=='ε'&&!nfaStates.containsKey(tb.getEdgeTarget(e).getId()))

                    queue.push(tb.getEdgeTarget(e));

            }

        }

        return nfaStates;

    }

(2) moves

    private HashMap<Integer,State> moves(State s, Character ch, LabeledDirectedGraph<State> tb){

        HashMap<Integer,State> nfaStates = new HashMap<>();

        //put each state reached by an edge from ‘s’ on 'ch' to ‘nfaStates’

        for(LabelEdge e : tb.edgeSet()){

            //Add your implementation

            /\*

            补充代码

             \*/

            if(tb.getEdgeSource(e).equals(s)&&e.getLabel()==ch)

                nfaStates.put(tb.getEdgeTarget(e).getId(),tb.getEdgeTarget(e));

        }

        return nfaStates;

    }

(3) ifNotExistsInDFA

    void ifNotExistsInDFA(ArrayDeque<State> queue, HashMap<Integer, State> TargetNfaStateSet, State tnfaStartState,

                          State tnfaAcceptState, State SourceDfaState, RDFA dfa, Character ch) {

        State targetDfaState = new State();

        queue.add(targetDfaState);

        //judge the dfaState whether is start state, middle state or accepting state

        if (TargetNfaStateSet.containsValue(tnfaAcceptState)) {

            targetDfaState.setType(State.ACCEPT);

        } else {

            targetDfaState.setType(State.MIDDLE);

        }

        //add the new DFA state ‘targetDfaState’ and its NFA state set to mapping

        dfa.setStateMappingBetweenDFAAndNFA(targetDfaState, TargetNfaStateSet);

        //add the new DFA state and new edge to transit table

        /\*

        补充代码

         \*/

        dfa.getTransitTable().addVertex(targetDfaState);

        dfa.getTransitTable().addEdge(new LabelEdge(SourceDfaState, targetDfaState, ch));

    }

(4) distinguishEquivalentState

    private HashMap<Integer,HashMap<Integer, State>> distinguishEquivalentState(RDFA dfa){

        LabeledDirectedGraph<State> tb = dfa.getTransitTable();

        ArrayList<Character> symbols = dfa.getAlphabet();

        HashMap<Integer, State> acceptGroup = new HashMap<>();

        HashMap<Integer, State> nonAcceptGroup = new HashMap<>();

        for(State s : tb.vertexSet()){

            if(s.getType() == State.ACCEPT || s.getType() == State.ACCEPTANDSTART)

                acceptGroup.put(s.getId(),s);

            else

                nonAcceptGroup.put(s.getId(),s);

        }

        HashMap<Integer,HashMap<Integer, State>> groupSet = new HashMap<>(); // group set

        groupSet.put(0,acceptGroup);

        if(!nonAcceptGroup.isEmpty()){

            groupSet.put(1,nonAcceptGroup);

        }

        HashMap<Integer,HashMap<Integer, State>> newGroupSet = new HashMap<>();

        int lastNum = 0; //used for recording #groups

        for(Character ch : symbols){//each symbol

            newGroupSet.clear();

            while(newGroupSet.size() != groupSet.size()){ //until no equivalent state can be distinguished

                if(newGroupSet.size() != 0 ) newGroupSet.clear();

                newGroupSet.putAll(groupSet);

                groupSet.clear(); //clear for holding new groups

                lastNum = 0;

                for (HashMap<Integer, State> group : newGroupSet.values()) { // each group

                    //distinguish non-equivalent states

                    HashMap<State, Integer> split = new HashMap<>();

                    for(State s : group.values()){

                        split.put(s,-1); //initialize that all states reach no group

                    }

                    //Check and record which group in groupSet each state 's' can reach on "ch".

                    //Record in split

                    //Check and record which group in groupSet each state 's' can reach on "ch".

                    //Record in split

                    for (State s : group.values()) {

                        //get the edge with "ch" as label and  's' as source vertex

                        //Add your implementation

                        boolean found = false;

                        for(LabelEdge e : tb.edgeSet()){

                            if(tb.getEdgeSource(e).equals(s) && e.getLabel().equals(ch)){

                                State target = tb.getEdgeTarget(e);

                                // Find which group the target state belongs to

                                for(Map.Entry<Integer, HashMap<Integer, State>> entry : newGroupSet.entrySet()){

                                    if(entry.getValue().containsKey(target.getId())){

                                        split.put(s, entry.getKey());

                                        found = true;

                                        break;

                                    }

                                }

                                break;

                            }

                        }

                        if(!found){

                            split.put(s, -1); // no transition on this symbol

                        }

                    }

                    //regroup equivalent states, and add the group into the group set

                    // set group id for new groups

                    HashMap<Integer,HashMap<Integer, State>> tempGroupSet = new HashMap<>();

                    for(Integer g : split.values()){//create groups

                        tempGroupSet.put(g, new HashMap<>());

                    }

                    for(State s : split.keySet()){

                        tempGroupSet.get(split.get(s)).put(s.getId(), s);

                    }

                    for(int i = 0; i < tempGroupSet.size(); i++){

                        groupSet.put(i+lastNum,new HashMap<Integer,State>());

                    }

                    int j =0;

                    for(HashMap<Integer,State> h : tempGroupSet.values()){

                        groupSet.put(j+lastNum, h);

                        j++;

                    }

                    lastNum = groupSet.size();

                }

            }

        }

        return groupSet;

    }

3. 3 错误处理

3.4 运行结果

（1）

Regular Grammar

Alphabet:[c, a, b]

Regexes:

[regex1 := c(a|b)\*]

{regex1 := c(a|b)\*=The regex tree:

(-:1)

    firstChild:(c:0)

    (\*:3)

(c:0)

(\*:3)

    firstChild:(|:2)

(|:2)

    firstChild:(a:0)

    (b:0)

(a:0)

(b:0)

}

Show the NFA:

Alphabet:[c, a, b]

Total edges:12

Start State:2

the transitTable is:

(3:1->4:1@ε)

(7:1->5:2@ε)

(4:1->6:1@ε)

(8:1->9:1@a)

(10:1->11:1@b)

(6:1->8:1@ε)

(2:0->3:1@c)

(6:1->10:1@ε)

(4:1->5:2@ε)

(9:1->7:1@ε)

(11:1->7:1@ε)

(7:1->6:1@ε)

Show the DFA:

Alphabet:[c, a, b]

Total edges:7

Start State:14

the transitTable is:

(14:0->25:2@c)

(27:2->27:2@b)

(26:2->27:2@b)

(25:2->26:2@a)

(25:2->27:2@b)

(26:2->26:2@a)

(27:2->26:2@a)

Show the miniDFA:

Alphabet:[c, a, b]

Total edges:3

Start State:30

the transitTable is:

(41:2->41:2@b)

(41:2->41:2@a)

(30:0->41:2@c)

（2）

Regular Grammar

Alphabet:[a]

Regexes:

[regex1 := a|ε]

{regex1 := a|ε=The regex tree:

(|:2)

    firstChild:(a:0)

    (ε:0)

(a:0)

(ε:0)

}

Show the NFA:

Alphabet:[a]

Total edges:6

Start State:0

the transitTable is:

(0:0->4:1@ε)

(3:1->1:2@ε)

(2:1->3:1@a)

(0:0->2:1@ε)

(5:1->1:2@ε)

(4:1->5:1@ε)

Show the DFA:

Alphabet:[a]

Total edges:1

Start State:6

the transitTable is:

(6:20->13:2@a)

Show the miniDFA:

Alphabet:[a]

Total edges:1

Start State:14

the transitTable is:

(14:20->21:2@a)

4、总结

本次实验通过实现正则表达式转DFA，深化了我对词法分析理论的理解并提升了算法实践能力：Subset Construction算法的编码让我直观掌握NFA到DFA的确定化过程，而状态最小化过程则体现了编译优化思想，结合Java集合框架实现算法的过程，也让我认识到理论需与编程语言特性结合，为后续学习语法分析等奠定了基础。