Lab2

基于LR1的Yacc

1. Motivation

   编写基于LR1的Yacc程序，理解语法分析过程。

2. Content description

   该程序使用java编写，运行平台MAC OS。

   程序运行过程：首先读取产生式文本，对其进行分析，基于LR1生成action表与goto表，并生成基于LR1的语法分析程序。然后对产生的语法分析程序进行编译，读取一个文本文件，首先使用上次实验写的词法分析程序对其进行分析得到token序列，再调用上一步生成的语法分析程序对去进行分析，返回结果为规约过程的式。

   实际实验的时候先写了一个解决冲突的LR1的分析程序，然后觉得很有趣又写了Yacc程序。因此写Yacc程序的时候尝试了一个比较简易的文法。

   文法如下：

   ```
   S->iSeS;
   S->iS;
   S->a
   S->b
   ```

3. Ideas/Methods

   1. 读取定义的文法
   2. 根据LR1构造action和goto表
   3. 将action与goto表以及LR1分析过程写入文件，输出文件在lr1文件夹中
   4. 编译上一步产生的语法分析程序

4. Assumptions

   1. 假设输入的文法为LR1文法
   2. 假设输入的文法只包含规定的终止符和非终止符

5. Data Structures

   1. change table 包括action表和goto表，封装所有相关操作

```java
import java.util.*;

public class ChangeTable {
    Map<Integer, String[]> changeTable = new TreeMap<>();//the action table
    Map<Integer, int[]> gotoTable = new TreeMap<>();// the goto table

    List<Character> ts = new ArrayList<>(); // the Vt
    List<Character> uts = new ArrayList<>(); // the Vn

    public ChangeTable(List<Character> ts, List<Character> uts) {
        this.ts = ts;
        this.uts = uts;
    }

    public void setShiftString(int i1, int i2, char c) {
        int pos = this.ts.indexOf(c);

        String[] str = changeTable.get(i1);
        if (str == null) {
            str = new String[ts.size()];
        }
        str[pos] = "s" + i2;
        this.changeTable.put(i1, str);
    }

    public void setGotoTable(int i1, int pos, int to) {
        int[] ints = this.gotoTable.get(i1);
        if (ints == null) {
            ints = new int[uts.size()];
            for (int i = 0; i < ints.length; i++) {
                ints[i] = -1;
            }
        }
        ints[pos] = to;
        this.gotoTable.put(i1, ints);
    }

    public void setReduceTable(int p, Map<Character, Integer> map) {
        String[] str = this.changeTable.get(p);
        if (str == null) {
            str = new String[ts.size()];
        }
        for (Character c : map.keySet()) {
```

2. PFTuple 产生式的元组，包括产生式序号，点的位置，follow的字符，产生式

```java
import javafx.util.Pair;

public class PFTuple {
    int pNumber;// the number of production
    int pointPosition; // the position of point in the production
    char follow; // the follow char
    String production; // the production

    public PFTuple(int pNumber,int pointPosition,char follow,String production){
        this.pNumber = pNumber;
        this.pointPosition = pointPosition;
        this.follow = follow;
        this.production = production;
    }

    public PFTuple movePoint(char c){
        PFTuple tuple=null;
        if (pointPosition<=production.length()-1&&production.charAt(pointPosition)==c){
            tuple= new PFTuple(pNumber, pointPosition: pointPosition+1,follow,production);
        }
        return tuple;
    }

    public boolean isSameTo(PFTuple t){
        return pNumber == t.pNumber && pointPosition == t.pointPosition && follow==(t.follow);
    }

    public boolean end(){
        if (this.pointPosition == this.production.length()){
            return true;
        }
        return false;
    }

    public Pair<Character,Character> nextIfUnterminal(){
        if (this.pointPosition>=this.production.length()){
            return null;
        }
        char thisChar = this.production.charAt(this.pointPosition);
        if (thisChar<='Z'&&thisChar>='A'){
            if (this.pointPosition+1<this.production.length()) {
                return new Pair<>(thisChar, this.production.charAt(pointPosition + 1));
            }
```

3. I 代表每个状态 包括状态的序号及一个PFTuple的列表

```java
public class I {
    private List<PFTuple> tuples = new ArrayList<>();
    private int number;

    public void setNumber(int number) { this.number = number; }

    public int getNumber() { return number; }

    public void addTuple(PFTuple tuple) { tuples.add(tuple); }

    public boolean notEmpty() {
        if (tuples == null || tuples.size() == 0) {
            return false;
        }
        return true;
    }

    public I movePoint(char c) {
        I result = new I();
        for (PFTuple t : tuples) {
            PFTuple tuple = t.movePoint(c);
            if (tuple != null) {
                result.addTuple(tuple);
            }
        }
        return result;
    }

    public int getSize() { return this.tuples.size(); }

    public boolean hasTuple(PFTuple t) {
        for (int i = 0; i < tuples.size(); i++) {
            if (tuples.get(i).isSameTo(t))
                return true;
        }
        return false;
    }
```

6. Core algorithms
    1. yacc部分：
        1. 首先读取文法，此处的文法定义将if、else用i和e表示。默认第一个产生式的第一个字符为start。
        2. 添加#->S产生式
        3. 从0号产生式开始构造状态0。将0号产生式加入状态0，follow设为$，然后进行状态内部扩展，并修改扩展产生式的follow项。
        4. 对所有的终止符进行遍历，转移的新状态加入到维护的状态队列中，转移则用来修改维护的action表。
        5. 对所有的非终止符进行遍历，到达的新状态加入到维护的状态队列中，转移则记录到维护的goto表。
        6. 每次结束所有遍历后查看是否有点在产生式最后，如果有则进行规约，规约记录到维护的action表。
        7. 从状态队列中继续取新的状态。对新状态继续4。当队列为空则构造结束。
        8. 可将表写入文件中。

```java
public static void main(String[] args) {
    List<Character> uts = new ArrayList<>();
    List<Character> ts = new ArrayList<>();
    List<String> productions = ProductionsReader.readFromFile();
    Character startChar = productions.get(0).charAt(0);

    for (String s : productions) {
```

```
        for (Character c : s.toCharArray()) {
            if (c >= 'A' && c <= 'Z') {
                if (!uts.contains(c))
                    uts.add(c);
            } else if (c != '#' && c != '-' && c != '>' &&
(!ts.contains(c))) {
                ts.add(c);
            }
        }
    }
    ts.add('$');

    ChangeTable changeTable = new ChangeTable(ts, uts);


    I I0 = new I();
    I0.setNumber(0);

    for (int i = 0; i < productions.size(); i++) {
        String pro = productions.get(i);
        if (pro.charAt(0) == startChar) {
            I0.addTuple(new PFTuple(i, deviationPosition + 0, '$', pro));
        }
    }

    I0 = extendInside(I0, productions);

    Queue<I> iQueue = new LinkedBlockingQueue<>();
    iQueue.offer(I0);
    iList.add(I0);

    int nextNumber = 1;

    while (true) {
        if (iQueue.isEmpty()) {
            break;
        }
        I currentI = iQueue.poll();

        int thisNumber = currentI.getNumber();

        for (Character c : ts) {
            I nextI = currentI.movePoint(c);
            nextI = extendInside(nextI, productions);
            if (nextI.notEmpty()) {
                I same = containsI(nextI);
                if (same == null) {
                    nextI.setNumber(nextNumber);
                    iQueue.offer(nextI);
                    iList.add(nextI);
                    changeTable.setShiftString(thisNumber, nextNumber, c);
                    nextNumber++;
```

```
                } else {
                    changeTable.setShiftString(thisNumber,
same.getNumber(), c);
                }
            }
        }

        for (int i = 0; i < uts.size(); i++) {
            char c = uts.get(i);
            I nextI = currentI.movePoint(c);
            nextI = extendInside(nextI, productions);
            if (nextI.notEmpty()) {
                I same = containsI(nextI);
                if (same == null) {
                    nextI.setNumber(nextNumber);
                    iQueue.offer(nextI);
                    iList.add(nextI);
                    changeTable.setGotoTable(thisNumber, i, nextNumber);
                    nextNumber++;
                } else {
                    changeTable.setGotoTable(thisNumber, i,
same.getNumber());
                }
            }
        }

        Map<Character, Integer> reductions = currentI.getReductions();
        if (reductions != null && reductions.size() != 0) {
            changeTable.setReduceTable(thisNumber, reductions);
        }
    }

    changeTable.printTable();

    String productionsToWrite = getProductionsToWrite(productions);
    String actionTable = changeTable.getTableString();
    String gotoTable = changeTable.getGOTOString();
    String gotoSequence = changeTable.getGotoSequence();
    String tokenMap = changeTable.getUtsString();


 FileWriter.writeAllFiles(productionsToWrite,actionTable,gotoTable,tokenMap
,gotoSequence);

}
```

2. LR1分析器部分
    1. 读入LexParser产生的token序列

2. 从0状态开始根据读入的字符以及yacc产生的分析表，查表如果发现为shift则进行状态转移，每次新状态加入状态栈中；当遇到规约时进行规约，根据规约式规约元素长度弹出相应状态数。
3. 当读取到$时则结束。

```
public List<String> Analyse() {
    List<String> reductions = new ArrayList<>();

    int currentState = 0;
    Stack<Integer> stateStack = new Stack<>();
    Stack<String> symbolStack = new Stack<>();
    symbolStack.push("$");
    stateStack.push(0);
    Token currentToken;

    System.out.println("State_Stack                Symbol_Stack
  Current_Token              Action");
    String states = "0";
    String symbols="$";
    while (true) {
        currentToken = tokens.get(0);
        if (currentToken == null || currentToken.getTokenType() == null) {
            break;
        }
        Action action = Table.getAction(currentState,
currentToken.getTokenType());

        if (action==null){
            System.out.println("Wrong input");
            return reductions;
        }

        if (action.getActionType() == ActionType.SHIFT) {
            symbolStack.push(currentToken.getLabel());
            currentState = action.getNumber();
            stateStack.push(currentState);
            states = currentState+" "+states;
            symbols = currentToken.getLabel()+" "+symbols;
            tokens.remove(0);
            System.out.println(states+"                "+symbols+"
       "+currentToken.getLabel()+"                "+action.toString());
        } else if (action.getActionType() == ActionType.REDUCE) {
            String reduce = Table.getProduction(action.getNumber());
            String [] args = reduce.split("->");
            String [] front = args[0].trim().split("");
            String [] back = args[1].trim().split("");

            for (int j = back.length-1; j >=0; j--) {
                if (back[j].equals(symbolStack.peek())){
                    symbolStack.pop();
                    stateStack.pop();
                    symbols= symbols.substring(symbols.indexOf(" ")+1);
```

```
                    states = states.substring(states.indexOf(" ")+1);
                }else {
                    System.out.println(back[j]);
                    System.out.println(symbolStack.peek());
                    System.out.println("Unknown syntax!");
                    return reductions;
                }
            }

            for (int j = 0; j < front.length; j++) {
                symbolStack.add(front[j]);
                symbols = front[j]+" "+symbols;
            }

            reductions.add(reduce);
            currentState = stateStack.peek();

            System.out.println(states+"                "+symbols+"
       "+currentToken.getLabel()+"                "+action.toString());

            if (Table.belongToGOTOs(symbolStack.peek())){
                currentState =
Table.getGOTO(currentState,symbolStack.peek());
                stateStack.push(currentState);
                states = currentState+" "+states;
                System.out.println(states+"                "+symbols+"
           "+currentToken.getLabel()+"                "+action.toString());
            }

        }else if (action.getActionType()==ActionType.ACCEPT){
            reductions.add(Table.getProduction(0));
            symbolStack.pop();
            stateStack.pop();
            symbols= symbols.substring(symbols.indexOf(" ")+1);
            states = states.substring(states.indexOf(" ")+1);
            System.out.println(states+"                "+symbols+"
       "+currentToken.getLabel()+"                "+action.toString());
            if ("$".equals(symbolStack.peek())){
                symbolStack.pop();
                System.out.println("Done");
            }else {
                System.out.println("Wrong input");
            }
            break;
        }else {
            System.out.println("NULL");
        }
    }

    return reductions;
}
```

7. use cases in test

文法：

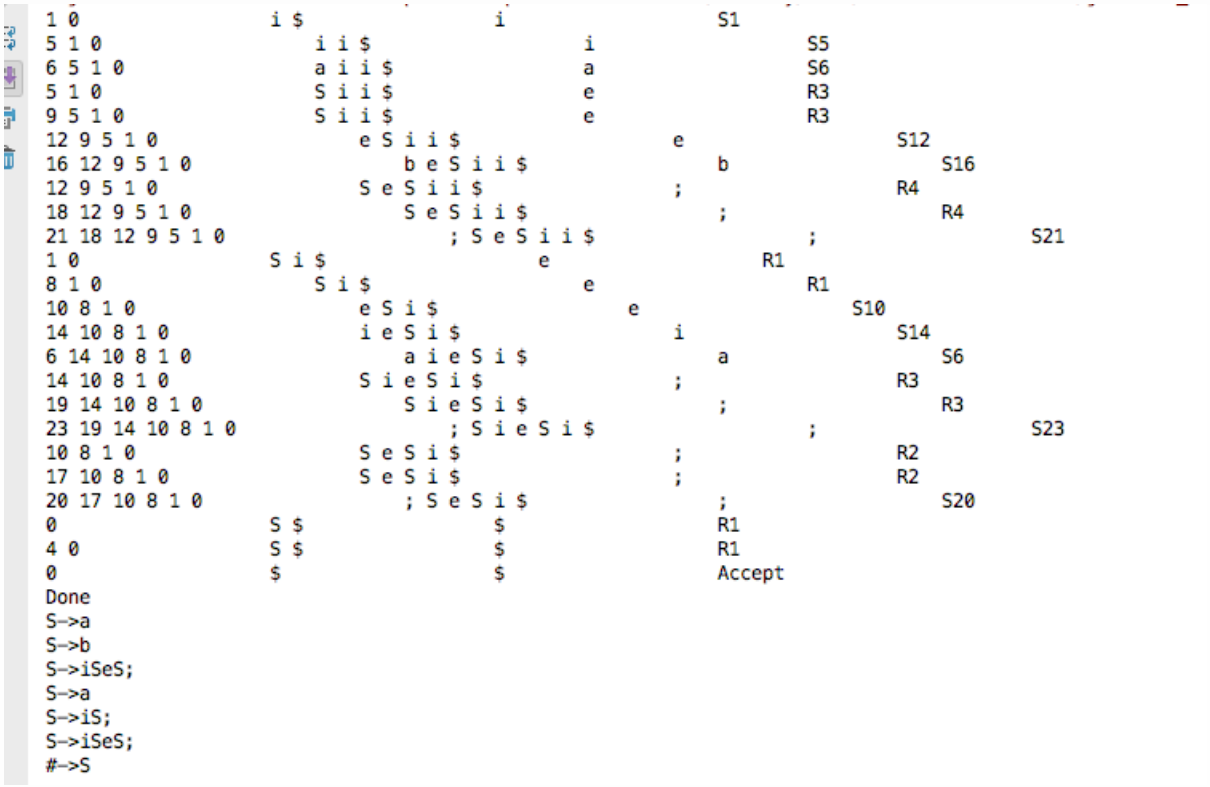
```
S->iSeS;
S->iS;
S->a
S->b
```

分析：

```
if
  if a
  else b;
else
  if a;
;
```

输出结果：

```
yacc.ChangeTable
{"s1",null,null,"s2","s3",null}4
{"s5",null,null,"s6","s7",null}8
{null,null,null,null,null,"r3"}
{null,null,null,null,null,"r4"}
{null,null,null,null,null,"a"}
{"s5",null,null,"s6","s7",null}9
{null,"r3","r3",null,null,null}
{null,"r4","r4",null,null,null}
{null,"s10","s11",null,null,null}
{null,"s12","s13",null,null,null}
{"s14",null,null,"s15","s16",null}17
{null,null,null,null,null,"r2"}
{"s14",null,null,"s15","s16",null}18
{null,"r2","r2",null,null,null}
{"s5",null,null,"s6","s7",null}19
{null,null,"r3",null,null,null}
{null,null,"r4",null,null,null}
{null,null,"s20",null,null,null}
{null,null,"s21",null,null,null}
{null,"s22","s23",null,null,null}
{null,null,null,null,null,"r1"}
{null,"r1","r1",null,null,null}
{"s14",null,null,"s15","s16",null}24
{null,null,"r2",null,null,null}
{null,null,"s25",null,null,null}
{null,null,"r1",null,null,null}
```



```
1 0                 i $                 i                   S1
5 1 0                 i i $                 i                   S5
6 5 1 0                 a i i $                 a                   S6
5 1 0                 S i i $                 e                   R3
9 5 1 0                 S i i $                 e                   R3
12 9 5 1 0                 e S i i $                 e                   S12
16 12 9 5 1 0                 b e S i i $                 b                   S16
12 9 5 1 0                 S e S i i $                 ;                   R4
18 12 9 5 1 0                 S e S i i $                 ;                   R4
21 18 12 9 5 1 0                 ; S e S i i $                 ;                   S21
1 0                 S i $                 e                   R1
8 1 0                 S i $                 e                   R1
10 8 1 0                 e S i $                 e                   S10
14 10 8 1 0                 i e S i $                 i                   S14
6 14 10 8 1 0                 a i e S i $                 a                   S6
14 10 8 1 0                 S i e S i $                 ;                   R3
19 14 10 8 1 0                 S i e S i $                 ;                   R3
23 19 14 10 8 1 0                 ; S i e S i $                 ;                   S23
10 8 1 0                 S e S i $                 ;                   R2
17 10 8 1 0                 S e S i $                 ;                   R2
20 17 10 8 1 0                 ; S e S i $                 ;                   S20
0                 S $                 $                   R1
4 0                 S $                 $                   R1
0                 $                 $                   Accept
Done
S->a
S->b
S->iSeS;
S->a
S->iS;
S->iSeS;
#->S
```

（附一个最开始写的LR1分析器的文法和结果（考虑了优先级和结合性）：

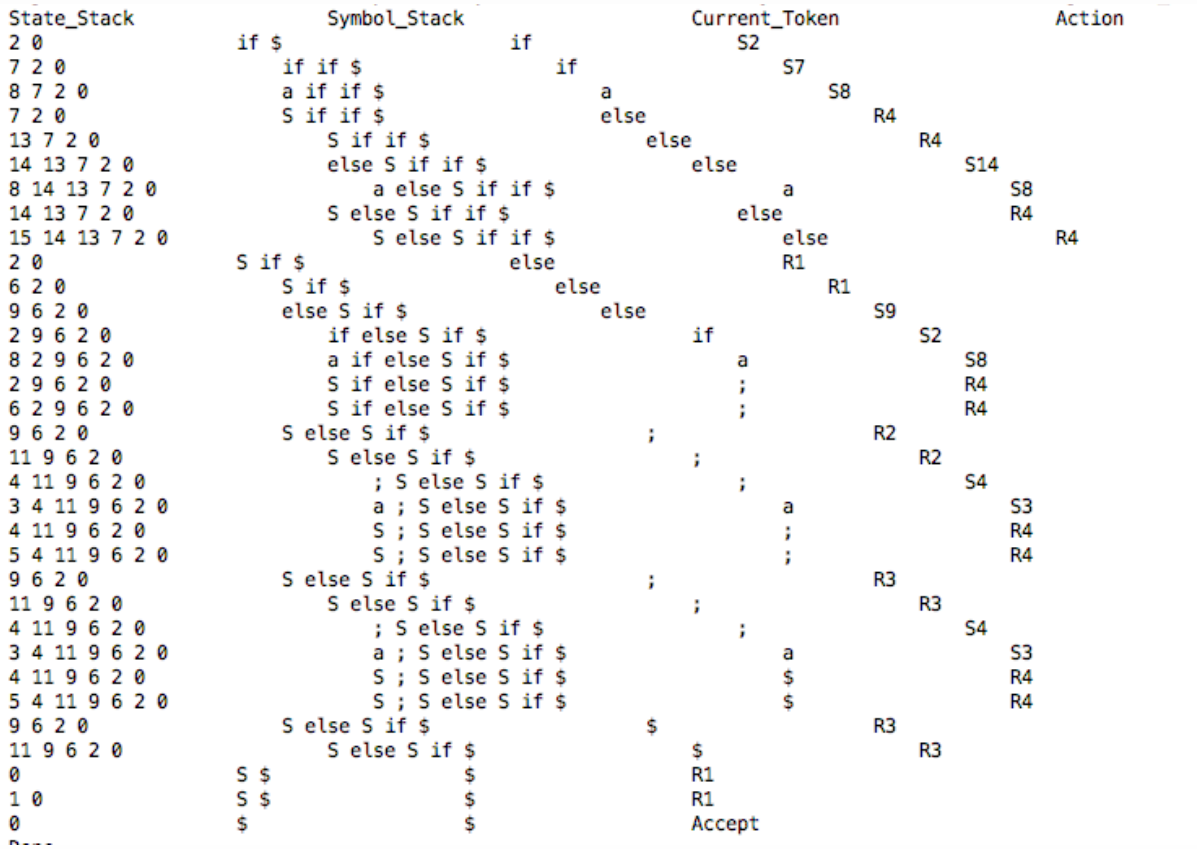
文法：

```
private static final String[] productions = {
        "S' -> S",
        "S -> if S else S",
        "S -> if S",
        "S -> S ; S",
        "S -> a"
};
```

分析输入：

```
if
  if a
  else a
else
  if a;
  a;
  a
```

输出结果：



State_Stack	Symbol_Stack	Current_Token	Action
2 0	if $	if S2	
7 2 0	if if $	if S7	
8 7 2 0	a if if $	a S8	
7 2 0	S if if $	else R4	
13 7 2 0	S if if $	else R4	
14 13 7 2 0	else S if if $	else S14	
8 14 13 7 2 0	a else S if if $	a S8	
14 13 7 2 0	S else S if if $	else R4	
15 14 13 7 2 0	S else S if if $	else R4	
2 0	S if $	else R1	
6 2 0	S if $	else R1	
9 6 2 0	else S if $	else S9	
2 9 6 2 0	if else S if $	if S2	
8 2 9 6 2 0	a if else S if $	a S8	
2 9 6 2 0	S if else S if $	; R4	
6 2 9 6 2 0	S if else S if $	; R4	
9 6 2 0	S else S if $	; R2	
11 9 6 2 0	S else S if $	; R2	
4 11 9 6 2 0	; S else S if $	; S4	
3 4 11 9 6 2 0	a ; S else S if $	a S3	
4 11 9 6 2 0	S ; S else S if $	; R4	
5 4 11 9 6 2 0	S ; S else S if $	; R4	
9 6 2 0	S else S if $	; R3	
11 9 6 2 0	S else S if $	; R3	
4 11 9 6 2 0	; S else S if $	; S4	
3 4 11 9 6 2 0	a ; S else S if $	a S3	
4 11 9 6 2 0	S ; S else S if $	$ R4	
5 4 11 9 6 2 0	S ; S else S if $	$ R4	
9 6 2 0	S else S if $	$ R3	
11 9 6 2 0	S else S if $	$ R3	
0	S $	$ R1	
1 0	S $	$ R1	
0	$	$ Accept	

```
S -> a
S -> a
S -> if S else S
S -> a
S -> if S
S -> a
S -> S ; S
S -> a
S -> S ; S
S -> if S else S
S' -> S
```

8. Problem occurred and related solutions

   LR1的整个实现过程还是比较顺畅的。最开始尝试的是LR1进行语法分析。遇到的问题主要是弹出状态栈的理解有问题。本来以为弹出只要弹出一个就好，结果代码怎么都跑不对... 后来仔细研究了一下发现弹出应当按照产生式右侧符号数量进行弹出，总的来说还是分析的过程理解不到位。

   Yacc部分最大的问题就是要新建许多数据结构以及需要对这些结构进行处理。稍微有点麻烦但是还好。

9. Feelings and comments

   感受就是，觉得自己写完了一个Yacc程序好棒呀。写代码的过程感觉还是蛮愉快的（除了de不出bug的时候）。另外就是实现LR1之后觉得LR1的整个过程很神奇，并且在debug过程中加深了对整个LR1过程的理解。

   //要是有时间的话希望编译原理多上一个月多做一个实验就更好了:)