华南师范大学实验报告

学院： 计算机学院

课题：编译原理课程项目

小组成员：

林光裕：20142100236

邱世余：20142100001

吴奇明：20142100168

Yacc 文法分析产生器文档

一. 简介

使用 Java 语言实验简单的 Yacc。这个程序读取一个 BNF 语法定义文件，然后输出一个 Java 语言的 LL1 分析器(如果输入的 BNF 定义文法不是 LL1 文法则报错)，这个 LL1 分析器能够判断一个输入的单词流是否符合 BNF 文法。

二. 实验环境

1. 语言环境: JDK8.0

2. 系统平台: Ubuntu16.04 (x64), CPU(i5 4590), 内存(8g)

3. 开发工具: NetBean 8.1

三. 实验过程分析

1. BNF 文法输入单词拆分:

通过 Java 的文件读取操作，读入每一行的 BNF 文法规则，通过识别::=符号对文法规则进行拆为两部分，非终结符和推导公式。接着保存非终结符和推导公式的映射，然后分析推导公式进行拆分，通过|符号进行拆分为多个推导情况，把每个推导情况放入数组中保存，接着对每个推导情况进行单个符号进行提取，转化为数组再放入。其结构如下:

举例:

<S> ::= <a> | <E><b> 转化后的格式： Map(S => [[a] , [E,b]])

依次循环读取建立结构。

2. 建立 First 集

对于 X 的 First(X)

(1). X 是一个终结符，那么 First(X) = X

(2). X 是一个非终结符, 递归找出 First(X)

代码分析:

```
// 获取一个非终结符的 First 集
 public HashSet<String> getFirstSet(String key) {
        Node node = this.bnf.get(key);
        ArrayList<String[]> childs = node.getChilds();
       //判断是否有分支情况
        if (childs.size() > 1) {
            this.mutilChoose.add(key);
        }
        HashSet<String> firstSet = new HashSet<>();
       //遍历递归找出 First 集
        for (int i = 0; i < childs.size(); i++) {
            for (int j = 0; j < childs.get(i).length; j++) {
                if (this.isTerminator(childs.get(i)[j])) { // 如果为终结符就直接保存并跳出
                    firstSet.add(childs.get(i)[j]);
                    break;
                } else { // 为非终结符则递归查找
                    HashSet<String> tempFirstSet = this.getFirstSet(childs.get(i)[j]);
                    Iterator itr = tempFirstSet.iterator();
                    while (itr.hasNext()) {
                        firstSet.add(itr.next().toString());
                    }
                    break;
                }
            }
        }
        return firstSet;
    }
// 获取所有非终结符的 First 集，并保存到 Map 结构中
```

```
public HashMap<String, HashSet<String>> setFirstSet() {
    //对所有非终结符进行 First 的查找建立
    Iterator iter = this.bnf.keySet().iterator();
    while (iter.hasNext()) {
        String key = iter.next().toString();
        if (!this.first.containsKey(key)) {
            HashSet<String> value = this.getFirstSet(key);
            this.first.put(key, value);
        }
    }
    return this.first;
}
```

3. 建立 Follow 集

(1). $放入 Follow(S)，S 代表最开始的符号， $表示最右端的结束标记

(2). 如果存在 A => aBb, 那么 First(b)中除空字符之外所有符号都在 Follow(B)中

(3). 如果存在 A => aB 或者 A => aBb 且 First(b)包含空字符, 那么 Follow(A)中的所有符号都在 Follow(B)中

代码分析:

```
// 获取一个非终结符的 Follow 集
public HashSet<String> getFollowSet(String key) {
    HashSet<String> followSet = new HashSet<>();
    Iterator itr = this.bnf.keySet().iterator();
    while (itr.hasNext()) {
        String _key = itr.next().toString();
        Node node = this.bnf.get(_key);
        ArrayList<String[]> childs = node.getChilds();
        for (int i = 0; i < childs.size(); i++) {
            for (int j = 0; j < childs.get(i).length; j++) {
```

```
                //设置能推出改 childs.get(i)[j]的字符
                this.setParentSet(childs.get(i)[j], _key);
                if (childs.get(i)[j].equals(key)) {
                    if (j < childs.get(i).length - 1) {
                        if (this.isTerminator(childs.get(i)[j + 1])) {
                            followSet.add(childs.get(i)[j + 1]);
                        } else {
                            HashSet<String> tempFollowSet = this.first.get(childs.get(i)[j + 1]);
                            Iterator _itr = tempFollowSet.iterator();
                            while (_itr.hasNext()) {
                                followSet.add(_itr.next().toString());
                            }
                        }
                    } else if (!_key.equals(key)) {
                        if (_key.equals(this.start)) {
                            followSet.add("$");
                        } else {
                            HashSet<String> tempFollowSet = this.getFollowSet(_key);
                            Iterator _itr = tempFollowSet.iterator();
                            while (_itr.hasNext()) {
                                followSet.add(_itr.next().toString());
                            }
                        }
                    }
                }
            }
        }
        return followSet;
    }
```

```
//获取所有非终结符的 Follow 集，并保存到 Map 结构中
public HashMap<String, HashSet<String>> setFollowSet() {
        Iterator iter = this.bnf.keySet().iterator();
        HashSet<String> firstWorldFollowSet = new HashSet<>();
        firstWorldFollowSet.add("$");
        this.follow.put(this.start, firstWorldFollowSet);
        while (iter.hasNext()) {
            String key = iter.next().toString();
            HashSet<String> tempFollowSet = this.getFollowSet(key);
            this.follow.put(key, tempFollowSet);
        }
        return this.follow;
    }
```

4. 判断是否为 LL1 文法

任意的 A => a | b 产生式

(1). Fisrt(a) 和 First(b)是不相交

(2). 当空字符存在 First(a)中，则 First(b)和 Follow(A)不相交

代码分析：

```
public Boolean isLL1() {
        for (int i = 0; i < this.mutilChoose.size(); i++) {
            String key = this.mutilChoose.get(i);
            for (int j = 0; j < arr.size(); j++) {
                for (int z = j + 1; z < arr.size(); z++) {
                    HashSet<String> result = new HashSet<>();
                    result.addAll(arr.get(j));
                    result.retainAll(arr.get(z));
                    if (result.size() > 0) {
                        return false;
                    }
```

```
            }
        }
        for (int j = 0; j < arr.size(); j++) {
            if (arr.get(j).contains("ε")) {
                HashSet<String> follow = this.follow.get(key);
                for (int z = 0; z < arr.size(); z++) {
                    if (j != z) {
                        HashSet<String> result = new HashSet<>();
                        result.addAll(follow);
                        result.retainAll(arr.get(z));
                        if (result.size() > 0) {
                            return false;
                        }
                    }
                }
            }
        }
    }
    return true;
}
```

5. 建立文法预测分析表

对每个文法产生式 A => a

(1)对于 First(a)， 将 A => a 加入到 M[A, a]中

(2)如果空字符在 Frists(a)中, 那么对于 Follow(A)中的每个终结符 b，将 A=>a 加入到 M[A, b]

对于空字符在 First(a)中，那么吧 A=>a 加入到 M[A, $]中。

代码分析:

```
public HashMap<String, HashMap<String, ArrayList<String[]>>> setLL1Table() {
    Iterator itr = this.notTerminator.iterator();
    while (itr.hasNext()) {
        String key = itr.next().toString();
```

```
Node node = this.bnf.get(key);
ArrayList<String[]> childs = node.getChilds();
for (int i = 0; i < childs.size(); i++) {
    if (this.isTerminator(childs.get(i)[0])) {
        if (childs.get(i)[0].equals("ε")) {
            HashSet<String> parentSet = this.parent.get(key);
            Iterator _itr = parentSet.iterator();
            while (_itr.hasNext()) {
                String parentKey = _itr.next().toString();
                HashSet<String> tempFollowSet = this.follow.get(parentKey);
                Iterator __itr = tempFollowSet.iterator();
                while (__itr.hasNext()) {
                    String _key = __itr.next().toString();
                    if (!_key.equals("ε")) {
                        this.insertLL1Table(key, _key, childs.get(i));
                    }
                }
            }
        } else if (!childs.get(i)[0].equals("ε")) {
            this.insertLL1Table(key, childs.get(i)[0], childs.get(i));
        }
    } else {
        HashSet<String> tempFirstSet = this.first.get(childs.get(i)[0]);
        if (tempFirstSet.contains("ε")) {
            HashSet<String> parentSet = this.parent.get(key);
            Iterator _itr = parentSet.iterator();
            while (_itr.hasNext()) {
                HashSet<String> tempFollowSet = this.follow.get(_itr.next().toString());
                Iterator __itr = tempFollowSet.iterator();
                while (__itr.hasNext()) {
                    String _key = __itr.next().toString();
```

```
                    if (_key.equals("ε")) {
                        this.insertLL1Table(key, _key, childs.get(i));
                    }
                }
            }
        } else {
            Iterator _itr = tempFirstSet.iterator();
            while (_itr.hasNext()) {
                String tempKey = _itr.next().toString();
                this.insertLL1Table(key, tempKey, childs.get(i));
            }
        }
    }
}
return this.ll1Table;
}
```

6.预测分析

代码分析：

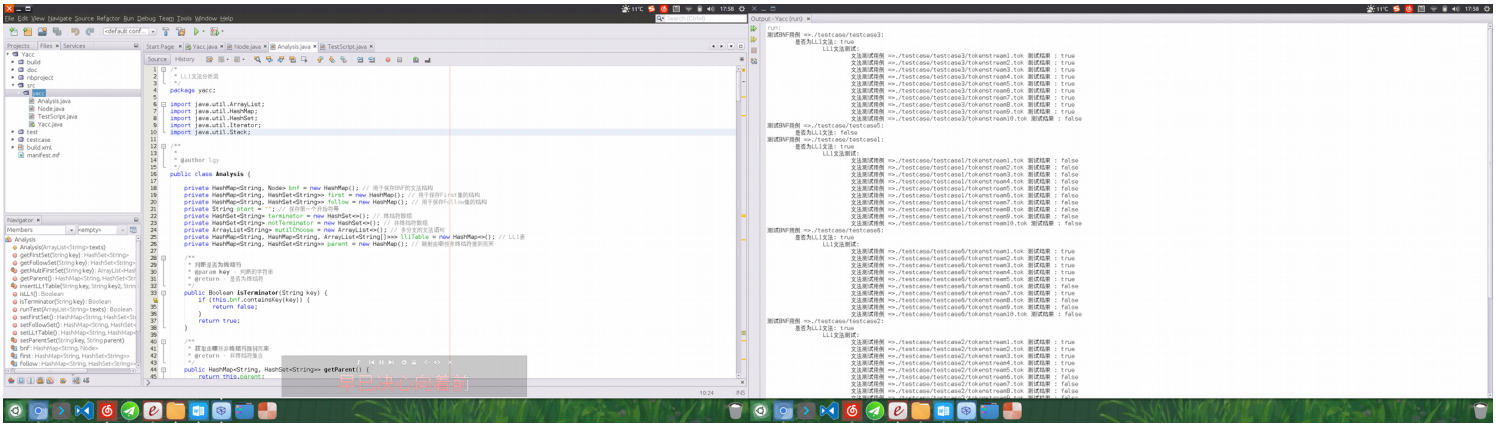
```
public Boolean runTest(ArrayList<String> texts) {
    for (int i = 0; i < texts.size(); i++) {
        String tempStr = texts.get(i).replaceAll("<|>|\"", "");
        if (tempStr.length() <= 0) {
            texts.set(i, "ε");
        } else {
            texts.set(i, tempStr);
        }
    }
    texts.add("$");
```

```
Stack<String> stack = new Stack<>();
stack.push("$");
stack.push(this.start);
String x = stack.peek();
HashMap<String, ArrayList<String[]>> tempMap = null;
ArrayList<String[]> tempList = null;
int ip = 0;
while (!x.equals("$")) {
    Boolean tableExist = true;
    if (this.ll1Table.containsKey(x)) {
        tempMap = this.ll1Table.get(x);
        if (!tempMap.containsKey(texts.get(ip))) {
            tableExist = false;
        } else {
            tempList = tempMap.get(texts.get(ip));
            if (tempList.size() > 1) {
                tableExist = false;
            }
        }
    } else {
        tableExist = false;
    }
    if (x.equals(texts.get(ip))) {
        stack.pop();
        ip++;
    } else if (this.isTerminator(x)) {
        return false;
    } else if (!tableExist) {
        return false;
    } else {
        stack.pop();
```

```
            String[] tempArray = tempList.get(0);
            for (int i = tempArray.length - 1; i >= 0; i--) {
                stack.push(tempArray[i]);
            }
        }
        x = stack.peek();
    }
    return true;
}
```

五. 运行方法及结构展示

使用 NetBean 进行运行



六. 实验总结

通过此次实验有利于深入了解 LL1 文法的规则以及求解 LL1 文法预测分析的过程，让我们充分学习求解 First 集，Follow 集这两个过程的具体分析方法，通过一步步的深入递进分析，最终得出结果，从编程思想上体现出步步递进，层层分析，最后把单一的职责方法结合为一个统一的分析过程。但此次实验中也存在着不足，就是代码的解耦能力不足，存在于一定的耦合性。希望接下来的实现过程能够以此次实验为基础，慢慢编程高内聚，低耦合的代码模式。

七. 参考资料

《编译原理》第二版 机械工业出版社

《编译原理及实践》第一版 机械工业出版社