语法分析器(LR(1))实践报告

2018211307 班 罗天佑 2018211349

一．系统功能设计

本语法分析器采取自底向上的分析方式，对用户指定的文法进行分析，可以自动获取非终结符的 FIRST 集、构建 LR(1)有效项目族、构建 LR(1)预测分析表，构造 LR(1)预测分析程序。并且提供了一定的错误检测机制。

二．模块划分

1. 读取文法模块,此模块提供文件读入与手工输入两种方式，丰富了用户使用选择。

```
void readGrammar(unordered_map<char, nonTerminator> &NonTerminator, unordered_map<
char, vector<string>> &Grammar, string defaultPath);
```

3. 计算自动生成 FIRST 集模块

```
void getFirst(unordered_map<char, nonTerminator> &NonTerminator, unordered_map<cha
r, vector<string>> &Grammar);
```

4. 构建 LR(1)有效项目族

```
void constructValidItemFamily(unordered_map<char, nonTerminator> NonTerminator,
unordered_map<char, vector<string>> Grammar, vector<validItemSet>
&validItemFamily);

validItemSet constructValidItemSet(validItemSet &VIS, char name);

void getClosure(unordered_map<char, nonTerminator> NonTerminator,
unordered_map<char, vector<string>> Grammar, validItemSet &VIS);
```

5. 根据已知的 FIRST 集，自动构建 LR(1)预测分析表

```
void makeTable(unordered_map<char, nonTerminator> NonTerminator, unordered_map<cha
r, vector<string>> Grammar, vector<vector<string>> &analysisTable);
```

6. 根据 LR(1)预测分析表判断用户输入的符号串是否符合该文法

```
bool analyzer(vector<vector<string>> analysisTable, string input);
```

三．功能函数

1. 在文法中找到终结符

```
void findTerminator(unordered_map<char, vector<string>> &Grammar);
```

2. 根据读入的符号与分析栈顶的符号在分析表中查询相应的候选式

```
string findCandidate(char row, char line, vector<vector<string>> analysisTable);
```

3. 将候选式填入响应的分析表中,如果冲突返回-1

```
int fillTable(char row, char line, string Candidate, vector<vector<string>> &analy
sisTable);
```

4. 判断当前符号的类别，1 为非终结符、7 为终结符、5 为空串

```
int isTerminator(char target);
```

5. 找到该字符在数组中的位置

```
int findIndex(char target, vector<char> arr);
```

6. 为新创建的非终结符创建一个不是非终结符也不是终结符的符号作为名字

```
char creatNewname(char name);
```

7. 判断该字符是否在该数组中出现

```
bool visedChar(char target, vector<char> arr);
```

8. 在控制台中输出绿色的横线以区分每个部分

```
void splitLine();
```

9. 查找一个字符串的 FIRST 集

```
vector<char> findFirst(string str, unordered_map<char, nonTerminator>
NonTerminator);
```

10. 寻找在 VIS、VIB 中的位置

```
int findVIBIndex(string str, vector<validItemBody> VIB);

int findVISIndex(validItemSet targetVIS, vector<validItemSet> VIF);
```

11. 创建一个新的 VIB

```
validItemBody CreateVIB(string str, char name, int isMath, int metaPos);
```

四．实验环境

1. Windows 10 家庭中文版 版本 1909（OS 内部版本 18363.1139）
2. IDE：VS code
3. 编译器：gcc version 5.1.0 (tdm64-1)

五．数据结构

1. 为每个非终结符提供一个结构体保存该非终结符是否存有空串并记录其 first

```
typedef struct NodeIII
{
    int isEmpty;
    vector<char> first;
```

```
} nonTerminator;
```

2. 记录文法中出现的非终结符

```
vector<char> nonterminator;
```

3. 记录文法中出现的终结符

```
vector<char> terminator; //此处'&'代表 num（整数），'^'代表 letter（字母）
```

4. 使用 unordered_map 保存非终结符的信息，使对非终结符的查询的时间效率达到 O(1)级别。

```
unordered_map<char, nonTerminator> NonTerminator
```

5. 使用 unordered_map 保存每个产生式的候选式，使对产生式的查询的时间效率达到 O(1)级别。

```
unordered_map<char, vector<string>> Grammar
```

6. 使用结构体保存有效项目集，记录该有效项目集的状态转换以及该集中所有的有效项目

```
typedef struct NodeII
{
    unordered_map<char, int> readyTogo;
    vector<validItemBody> VIB;
} validItemSet;
```

7. 使用 vector 容器保存 LL(1)预测分析表

```
vector<vector<string>> analysisTable
```

8. 使用结构体保存有效项目，记录该项目的式子、向前搜索符、项目类型、以及原符号位置

```
typedef struct NodeI
{
    int kind;    //1：待约项目、2：移进项目、3.规约项目、4：接受项目
    int metaPos; //记录原符号的位置
    string Candidate;
    vector<char> forwardSearch;
} validItemBody;
```

六．创新点

1. 本语法分析器强调用户的交互感受，因此程序提供手动输入与文件输入两种形式，其中文件输入形式中还提供了默认文件地址，使用户可以快速对文法进行分析，并快速进入预测分析程序
2. 本程序提供一定的错误处理。对于不属于 LR(1)文法的文法，在填写预测分析表时如果发现冲突，程序会显示冲突

七．流程图

八．测试用例以及运行结果

文法：

1. S->A
2. A->BA|@
3. B->aB|b

运行结果：

```
--------------------------------------------------------------------------
Please enter your symbol string:abab


ANALYSISING...
--------------------------------------------------------------------------
No:1
stateStack:0
symbolStack:-
input:a b a b $
output:Shift 2
--------------------------------------------------------------------------
No:2
stateStack:02
symbolStack:-a
input:b a b $
```

```
output:Shift 1
--------------------------------------------------------------------------
No:3
stateStack:021
symbolStack:-ab
input:a b $
output:Reduce B->b
--------------------------------------------------------------------------
No:4
stateStack:026
symbolStack:-aB
input:a b $
output:Reduce B->aB
--------------------------------------------------------------------------
No:5
stateStack:03
symbolStack:-B
input:a b $
output:Shift 2
--------------------------------------------------------------------------
No:6
stateStack:032
symbolStack:-Ba
input:b $
output:Shift 1
--------------------------------------------------------------------------
No:7
stateStack:0321
symbolStack:-Bab
input:$
output:Reduce B->b
--------------------------------------------------------------------------
No:8
stateStack:0326
symbolStack:-BaB
input:$
output:Reduce B->aB
--------------------------------------------------------------------------
No:9
stateStack:033
symbolStack:-BB
input:$
output:Reduce A->@
--------------------------------------------------------------------------
No:10
stateStack:0335
symbolStack:-BBA
input:$
output:Reduce A->BA
--------------------------------------------------------------------------
No:11
stateStack:035
symbolStack:-BA
input:$
```

```
output:Reduce A->BA
--------------------------------------------------------------------------
No:12
stateStack:04
symbolStack:-A
input:$
output:ACCEPT
--------------------------------------------------------------------------
The symbol string you entered is right.
Congratulations.
```

文法

1. E->E+T|E-T|T
2. T->T*F|T/F|F
3. F->(E)|&

运行结果：

```
--------------------------------------------------------------------------
Please enter your symbol string:1-2/3+(8)
ANALYSISING...
--------------------------------------------------------------------------
No:1
stateStack:0
symbolStack:-
input:1 - 2 / 3 + ( 8 ) $
output:Shift 1
--------------------------------------------------------------------------
No:2
stateStack:01
symbolStack:-1
input:- 2 / 3 + ( 8 ) $
output:Reduce F->&
--------------------------------------------------------------------------
No:3
stateStack:03
symbolStack:-F
input:- 2 / 3 + ( 8 ) $
output:Reduce T->F
--------------------------------------------------------------------------
No:4
stateStack:04
symbolStack:-T
input:- 2 / 3 + ( 8 ) $
output:Reduce E->T
--------------------------------------------------------------------------
```

```
No:5
stateStack:02
symbolStack:-E
input:- 2 / 3 + ( 8 ) $
output:Shift 26
--------------------------------------------------------------------------
No:6
stateStack:0226
symbolStack:-E-
input:2 / 3 + ( 8 ) $
output:Shift 1
--------------------------------------------------------------------------
No:7
stateStack:02261
symbolStack:-E-2
input:/ 3 + ( 8 ) $
output:Reduce F->&
--------------------------------------------------------------------------
No:8
stateStack:02263
symbolStack:-E-F
input:/ 3 + ( 8 ) $
output:Reduce T->F
--------------------------------------------------------------------------
No:9
stateStack:022629
symbolStack:-E-T
input:/ 3 + ( 8 ) $
output:Shift 22
--------------------------------------------------------------------------
No:10
stateStack:02262922
symbolStack:-E-T/
input:3 + ( 8 ) $
output:Shift 1
--------------------------------------------------------------------------
No:11
stateStack:022629221
symbolStack:-E-T/3
input:+ ( 8 ) $
output:Reduce F->&
--------------------------------------------------------------------------
No:12
stateStack:0226292225
symbolStack:-E-T/F
input:+ ( 8 ) $
output:Reduce T->T/F
--------------------------------------------------------------------------
No:13
stateStack:022629
symbolStack:-E-T
input:+ ( 8 ) $
output:Reduce E->E-T
--------------------------------------------------------------------------
```

```
No:14
stateStack:02
symbolStack:-E
input:+ ( 8 ) $
output:Shift 27
--------------------------------------------------------------------------
No:15
stateStack:0227
symbolStack:-E+
input:( 8 ) $
output:Shift 5
--------------------------------------------------------------------------
No:16
stateStack:02275
symbolStack:-E+(
input:8 ) $
output:Shift 6
--------------------------------------------------------------------------
No:17
stateStack:022756
symbolStack:-E+(8
input:) $
output:Reduce F->&
--------------------------------------------------------------------------
No:18
stateStack:022758
symbolStack:-E+(F
input:) $
output:Reduce T->F
--------------------------------------------------------------------------
No:19
stateStack:022759
symbolStack:-E+(T
input:) $
output:Reduce E->T
--------------------------------------------------------------------------
No:20
stateStack:022757
symbolStack:-E+(E
input:) $
output:Shift 21
--------------------------------------------------------------------------
No:21
stateStack:02275721
symbolStack:-E+(E)
input:$
output:Reduce F->(E)
--------------------------------------------------------------------------
No:22
stateStack:02273
symbolStack:-E+F
input:$
output:Reduce T->F
--------------------------------------------------------------------------
```

```
No:23
stateStack:022728
symbolStack:-E+T
input:$
output:Reduce E->E+T
--------------------------------------------------------------------------
No:24
stateStack:02
symbolStack:-E
input:$
output:ACCEPT
--------------------------------------------------------------------------
The symbol string you entered is right.
Congratulations.
```

运行结果（错误分析）

```
--------------------------------------------------------------------------
Please enter your symbol string:1-(2/3()


ANALYSISING...
--------------------------------------------------------------------------
No:1
stateStack:0
symbolStack:-
input:1 - ( 2 / 3 ( ) $
output:Shift 1
--------------------------------------------------------------------------
No:2
stateStack:01
symbolStack:-1
input:- ( 2 / 3 ( ) $
output:Reduce F->&
--------------------------------------------------------------------------
No:3
stateStack:03
symbolStack:-F
input:- ( 2 / 3 ( ) $
output:Reduce T->F
--------------------------------------------------------------------------
No:4
stateStack:04
symbolStack:-T
input:- ( 2 / 3 ( ) $
output:Reduce E->T
--------------------------------------------------------------------------
No:5
stateStack:02
symbolStack:-E
input:- ( 2 / 3 ( ) $
```

```
output:Shift 26
------------------------------------------------------------------------
No:6
stateStack:0226
symbolStack:-E-
input:( 2 / 3 ( ) $
output:Shift 5
------------------------------------------------------------------------
No:7
stateStack:02265
symbolStack:-E-(
input:2 / 3 ( ) $
output:Shift 6
------------------------------------------------------------------------
No:8
stateStack:022656
symbolStack:-E-(2
input:/ 3 ( ) $
output:Reduce F->&
------------------------------------------------------------------------
No:9
stateStack:022658
symbolStack:-E-(F
input:/ 3 ( ) $
output:Reduce T->F
------------------------------------------------------------------------
No:10
stateStack:022659
symbolStack:-E-(T
input:/ 3 ( ) $
output:Shift 16
------------------------------------------------------------------------
No:11
stateStack:02265916
symbolStack:-E-(T/
input:3 ( ) $
output:Shift 6
------------------------------------------------------------------------
No:12
stateStack:022659166
symbolStack:-E-(T/3
input:( ) $
output:Can not find Candidate
------------------------------------------------------------------------
There are some problems with the string of symbols you entered.
Please try again
```

九．遇到的问题及解决办法

1. 本程序的有效项目集族由有效项目、有效项目集、有效项目集族三级数据结构组成，在构建有效项目集族中分为构建有效项目、构建有效项目集、构建有效项目集族三级构建。这种分级构建的方式提升调试效率以及简化构建逻辑

```
validItemBody CreateVIB(string str, char name, int isMath, int metaPos);

validItemSet constructValidItemSet(validItemSet &VIS, char name);

void constructValidItemFamily(unordered_map<char, nonTerminator> NonTerminator,
unordered_map<char, vector<string>> Grammar, vector<validItemSet>
&validItemFamily);
```

2. 在构建出新的有效项目集后，为了避免冗余，要将新有效项目集与有效项目集族中其他有效项目逐一比对。一开始，我认为要先对有效项目集中的有效项目进行排序，对排好序的产生式比较才是有意义的。但是这种方法不仅会改变原本产生式的顺序，也是不必要的。因为，程序在构建新的有效项目集时，按照vector容器中元素顺序搜索，得到新有效项目集的元产生式，后续的闭包扩展有效项目集也是顺序拓展。因此理论上，如果两个有效项目集真的一样，意味着它们按照相同的构建方式构建出来，其有效项目的顺序也是一样的，不必再排序。

3. 在本程序中，我吸取LL(1)预测分析程序中存在的问题，将字符串的FIRST的求解独立为一个函数，使求向前搜索符函数简洁易懂、逻辑也懂清楚

4. 使用 unordered_map 保存非终结符的信息和每个产生式的候选式，使对非终结符的查询的时间效率达到O(1)级别。不仅提升了查询效率，而且将非终结符于产生式独立出来的数据结构使得我在计算FIRST集中逻辑清晰明了。而预测分析表作为LR(1)文法分析的关键，使用vector容器，可以高效使用它的许多优秀的特性，也给我的预测分析带来了很大的便利。

十．实践总结与心得

本次实验强化了我对与语法分析以及语法分析器的理解，也让我切身的体验了设计编译原理过程中的困难与解决困难获得的收获。语法分析器作为编译原理的核心分析部分，实际的上机设计与实现对于我更加深入理解LR(1)文法，自顶向下的预测分析过程有极大的帮助。在设计的过程中，我也懂得了提前规划与设计好数据结构对于后续程序功能实现的巨大作用。同时，这三次上机实验极大的锻炼了我调试程序的经验，使我逐渐学习熟练的运用c++的unoreder_map、vector容器及其优秀的特性。对于我的编写代码能力有极大的提升。这也锻炼了我处理大型工程时的经验。感谢老师提供这样的上机实验项目，让我们让张老师的同行的道路又迈进了一步。