MyYACC

> 简易LR(1)分析程序的生成程序
> 161250174 许元武

Motivation/Aim

为了巩固编译原理中语法分析这部分重中之重，特别是构造LR(1)分析表和根据分析表来parse的部分，进行本次实验，构造一个简单的YACC程序。

Content description

- YACC输入文件(expression.y)
- 语法分析器生成程序MyYACC源代码(/src)
- 项目结构文件(CMakeLists.txt)
- 测试输入文件(input.txt)
- 输出文件(production_sequence.txt)
- 实验报告(本文件)

Ideas/Methods

语法分析有多种方法，能分析的语法各不相同。有自上而下的LL(1)文法，自下而上的SRL(1)、LR(1)文法等。
本程序生成了一个可以分析LR(1)文法的parser

程序运行过程：

- 读入.y文件的产生式，生成LR(0)items
- 根据LR(0)items 使用"epsilon闭包法"(状态内部扩展)和"子集构造法"(状态间扩展)生成LR(1)items并构造LR(1)DFA
- 根据LR(1)DFA 产生 分析表，将分析表写入将要生成的parser中
- 生成parser(parser内部模拟LR分析模型[读头、状态栈等])

Assumptions

- 分别以'#','$','%'代表 CFG`的开始符，输入流结束符和 epsilon
- .y文件中第一行为单个语法开始符
- .y文件中大写字母代表非终结符，终结符用非大写字母的单个字符表示
- .y文件中每个符号要用空格隔开
- 输入流文件中结尾要有'$'结束符，并且不能包含空格
- 输入文法不能有二义性

Related FA descriptions

程序中没有生成layeredFA，而是直接根据LR(0)item 生成LR(1)DFA
LR(1)DFA的数据结构参见 Description of important Data Structures

Description of important Data Structures

程序中有以下重要数据结构
LR(0)item:

```
class LR_0_item {
public:
    int productionId,dotPos;
    char afterDot,VN;
    LR_0_item(int p = -1,int pos = -1,char c = 0,char v = 0):productionId(p),dotPos(pos),afterDot(c),VN(v){}
    bool operator<(const LR_0_item& other) const{
```

```
            return productionId == other.productionId? dotPos<other.dotPos:productionId<other.productionId;
        }
        bool operator == (const LR_0_item& other) const {
            return productionId == other.productionId && dotPos == other.dotPos;
        }
    };
```

LR(1)item:

```
    class LR_1_item {
    public:
        LR_0_item item;
        char predictSymbol;
        LR_1_item(LR_0_item i,char s):item(i),predictSymbol(s){}
        bool operator<(const LR_1_item& other) const{
            return item == other.item ? (predictSymbol<other.predictSymbol): (item<other.item);
        }
    };
```

LR(1)DFA:

```
    struct edge{
        char val;
        int dest;
        edge(char v,int d):val(v),dest(d){}
    };

    class LR_1_DFA {
    public:
        vector<set<LR_1_item>> nodes;
        map<set<LR_1_item>,int> node_id;
        vector<vector<edge>> edges;
        LR_1_DFA(set<LR_0_item>& items);

    private:
        int id = 0;
        /**
         *
         * @param items :LR(0)items, like [S -> {dot}ABc]
         * @param core :set of LR(1)items
         * @return epsilon_closure of the core
         */
        set<LR_1_item> epsilonClosure(set<LR_0_item>& items,set<LR_1_item> core);
        /**
         *
         * @param items :LR(0)items, like [S -> {dot}ABc]
         * @param node :node_id in the dfa(start node)
         * @param value :the value of the edge, a VN or VT
         * @return
         */
        set<LR_1_item> subSetConstruction(set<LR_0_item>& items,int node,char value);
        void first(const string& s,set<char>& res);
    };
```

Description of core Algorithms

- 构造LR(1)DFA时用到的"epsilon闭包法"和"子集构造"法
- 生成LR(1)DFA 节点时用到的BFS
- parser中的LR分析模型(状态栈+读头+分析表)

Use cases on running

本次实验采用了以下文法:

```
E
E -> E + T | E - T | T
T -> T * F | T / F | F
```

```
E -> ( E )
F -> i
```

采用了以下输入流:

```
((i+i)*i+i/(i-i))*i$
```

生成了以下产生式序列:

```
F -> i
T -> F
E -> T
F -> i
T -> F
E -> E+T
F -> (E)
T -> F
F -> i
T -> T*F
E -> T
F -> i
T -> F
F -> i
T -> F
E -> T
F -> i
T -> F
E -> E-T
F -> (E)
T -> T/F
E -> E+T
F -> (E)
T -> F
F -> i
T -> T*F
E -> T
parsing finished.
```

Problems occurred and related solutions

- 如何优雅地将分析表写入分析程序中
  - 解决：使用c++ 11的统一初始化列表

    ```
    map<int,map<char,string>> PT = {{0, {{ '(' ,"s1"}, { 'E' ,"2"}, { 'F' ,"3"}...
    ```

  这样可以直接通过PT[state_id][ch_val]来查表

Your feelings and comments

FA模型真的非常强大