BJTU编译原理lab4实验报告

编译原理Lab04: 算符优先分析

22281089 陈可致

一. 实验要求

1. 实验项目

以专题 1 词法分析程序的输出为语法分析的输入, 实现算符优先分析算法, 完成以下描述算术表达式的算符优先文法的算符优先分析过程

```
G[E]:E→E+T | E-T | T
T→T*F | T/F | F
F→(E) | i
```

2. 设计说明

- 终结符号 i 为用户定义的简单变量, 即标识符的定义

3. 设计要求

- 构造该算符优先文法的优先关系矩阵或优先函数
- 输入串应是词法分析的输出二元式序列, 即某算术表达式"专题 1"的输出结果 输出为输入串是否为该文法定义的算术表达式的判断结果
- 算符优先分析过程应能发现输入串出错
- 设计至少四个测试用例(尽可能完备, 正确和出错), 并给出测试结果
- 选做: 考虑根据算符优先文法构造算符优先关系矩阵, 包括 FIRSTVT 和 LASTVT 集合, 并添加到你的算符优先分析程序中

二. 开发环境

- Ubuntu 24.04.1 LTS
- g++ (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0

三. 运行方式

1. 需要 g++ 编译器, 没有可以用以下命令安装

```
sudo apt update
sudo apt install g++
```

2. 对于每个实验, 都编写了 sh脚本 和 测试数据文件 用于测试项目, 只需要在项目文件夹目录下运行.sh文件即可进行测试

```
cd your_file_forder
bash go_mp.sh
```

```
cd your_file_forder
bash go_opp.sh
```

四. 项目概述

本实验实现了一个算符优先分析程序, 能够解析基于算符优先文法的算术表达式 程序的输入是词法分析程序的输出二元式序列, 输出是输入串是否为文法定义的算术表达式的判断结果 该程序通过构造算符优先关系矩阵, 实现了算符优先分析过程, 并能够发现输入串中的语法错误

五. 程序设计概述

- LL solver

```
class lycoris {
   public:
    lycoris() {}
    lycoris(const grammar &g) : G(g), ok(0) {}
    meion check(const string &s) -> bool {}
    meion show() -> void {}

   private:
    grammar G;
    hash_map<hash_map<char>> mp;
    token_solver t_sol;
    grammar_solver g_sol;
    int ok;
    meion build() -> void {}
    meion get_precedence(token_type L, token_type R) -> char {}
};
```

六. 程序设计

1. 项目结构
   项目分为几个模块
   - Lib: 头文件
     1. MeloN_H.hpp: 用到的标准库头文件. 使用的stl容器, 宏定义
     2. MeloN_debug.hpp: 调试头文件, 用于格式化输出不定参数的变量信息, 标准运行环境下 **不会** 生效
     3. 3_opp_solver.hpp: 定义了OPP::lycoris类, 用于LL(1)分析, 提供了一个方法用于测试
   - testcase 测试数据 | std
     1. 4组测试数据 (in0 - in3)
     2. 4组对应的标准输出 (std0 - std3)
   - testcase_mp 测试数据2 | std
     1. 标准输出 std
   - 测试程序
     1. test_mp.cpp: 用于测试OPP::lycoris类
     2. go_mp.sh: 用于测试项目的脚本
     3. test_opp.cpp: 也是用于测试OPP::lycoris类
     4. go_opp.sh: 用于测试项目的脚本
2. OPP::lycoris类
   1. 构造函数:
      - lycoris(): 默认构造函数, 创建一个空对象, 不做任何初始化

- lycoris(const grammar &g): 接收一个文法 g, 初始化类成员并执行以下操作:
  - 调用 token_solver 构建初始状态
  - 设置文法 g 给 grammar_solver
  - 计算 FIRSTVT 和 LASTVT 集合
  - 调用 build() 方法构建算符优先关系表

2. 公有方法

- meion check(const string &s): 解析输入字符串, 判断其是否符合文法定义 使用算符优先关系矩阵驱动的语法分析过程, 通过栈操作和优先关系判断进行匹配和归约
- meion show_mp(): 打印算符优先关系矩阵, 展示每个终结符号在不同输入符号下的优先关系

3. 私有方法

- meion build(): 构建算符优先关系表 mp, 填充符号之间的优先级关系

3. OPP::test(): 测试OPP::lycoris类的函数

七. 测试

- **测试用例**
  - in0:

```
b + c * (d - e / f) + g - h * (i + j / k
```

  - in1:

```
b * (c + d) / e - f / g
```

  - in2:

```
b + c * (d - e / f) + g - h * i + j / k)
```

  - in3:

```
b + c /
```

- **std**
  - std0:

```
Zako♡~ exactly invalid
```

  - std1:

```
valid
```

  - std2:

```
Zako♡~ exactly invalid
```

  - std3:

```
Zako♡~ exactly invalid
```

  - std(st)

```
M ( ) + - * /

M - - > > > > >
( < < = < < < <
) - - > > > > >
+ < < > > > < <
- < < > > > < <
* < < > > > > >
/ < < > > > > >

```

- **测试结果**
  - 结果正确 (对于输入invalid表达式的testcase存在一些std::cerr输出的分析信息) ~~要看程序输出的话可以把sh脚本中删除输出文件的语句注释~~

```
bash go_mp.sh
accept

bash go_opp.sh
test: 0
Zako♡~ exactly invalid reduction pattern.
accept
test: 1
accept
test: 2
Zako♡~ exactly invalid reduction pattern.
accept
test: 3
Zako♡~ exactly invalid reduction pattern.
accept
```

八. 心得体会

大模拟写写写 zzz

debug过程显著提高了抗压能力

算符优先分析实验不仅帮助我巩固了对编译原理的理解, 也让我体会到编译器设计的严谨和逻辑之美 我期待在接下来的学习中挑战更加复杂的分析方法, 深入理解编译原理的更多应用场景!

附录

1. MeloN_H.hpp

```cpp
#pragma once
#include <algorithm>
#include <array>
#include <bitset>
#include <cassert>
#include <cctype>
#include <chrono>
#include <cmath>
#include <cstring>
#include <ctime>
#include <fstream>
#include <functional>
#include <iomanip>
#include <iostream>
#include <limits>
#include <map>
#include <queue>
#include <random>
#include <ranges>
#include <set>
#include <stack>
#include <string>
#include <tuple>
#include <unordered_map>
#include <unordered_set>

using   std::array, std::bitset, std::deque, std::greater, std::less, std::map,
        std::multiset, std::pair, std::priority_queue, std::set, std::stack,
        std::string, std::vector, std::tuple, std::function;
```

```
30
31  using NAME = void;       using uint = unsigned;   using ll = long long;      using ull = unsigned long lor
32  using ld = long double;  using i128 = __int128_t; using u128 = __uint128_t;  using f128 = __float128;
33
34  #define meion     auto
35  #define iroha     return
36
```

2. MeIoN_debug.hpp

```
1   #pragma once
2
3   template <class T, size_t size = std::tuple_size<T>::value>
4   std::string to_debug(T, std::string s = "")
5       requires(not std::ranges::range<T>);
6   std::string to_debug(meion x)
7       requires requires(std::ostream& os) { os << x; }
8   {
9       iroha static_cast<std::ostringstream>(std::ostringstream() << x).str();
10  }
11  std::string to_debug(std::ranges::range meion x, std::string s = "")
12      requires(not std::is_same_v<decltype(x), std::string>)
13  {
14      for (meion xi : x) {
15          s += ", " + to_debug(xi);
16      }
17      iroha "[" + s.substr(s.empty() ? 0 : 2) + "]";
18  }
19  template <class T, size_t size>
20  std::string to_debug(T x, std::string s)
21      requires(not std::ranges::range<T>)
22  {
23      [&]<size_t... I>(std::index_sequence<I...>) {
24          ((s += ", " + to_debug(std::get<I>(x))), ...);
25      }(std::make_index_sequence<size>());
26      iroha "(" + s.substr(s.empty() ? 0 : 2) + ")";
27  }
28  #ifdef MeIoN
29  #define debug(...)  std::cout << "Ciallo～(∠・ω< )⌒★ " << "(" #__VA_ARGS__ ") = " << to_debug(std::tupl
30  #else
31  #define debug(...) void(0721)
32  #endif
```

3. 3_OPP_solver.hpp

```
1   #include "0_token_solver.hpp"
2   #include "1_grammar_solver.hpp"
3
4   namespace OPP {
5       using grammar_solver = n_grammar_solver::lycoris;
6       using n_grammar_solver::grammar;
7       using n_grammar_solver::production;
8       using n_grammar_solver::token_solver;
9       class lycoris {
10      public:
11          lycoris() {}
12          lycoris(const grammar &g) : G(g), ok(0) {
13              t_sol.build("");
14              g_sol.set_grammar(G);
15              g_sol.compute_first_vt();
16              g_sol.compute_last_vt();
17              build();
18          }
```

```
meion check(const string &s) -> bool {
    if (not ok) build();
    if (ok == -1) {
        std::cerr << "Check: Zako♡~ build fail" << std::endl;
        iroha false;
    }
    set<vector<token_type>> patterns;
    G.view([&](const ull &key, const vector<production> &val) -> void {
        for (const production &prod : val) {
            vector<token_type> pattern;
            for (const meion &t : prod.Rs()) {
                if (g_sol.is_non_terminal(t)) {
                    pattern.emplace_back(Non_terminal);
                } else {
                    token_type type = t_sol.get_token_type(t);
                    pattern.emplace_back(type);
                }
            }
            patterns.insert(pattern);
        }
    });
    vector tokens = t_sol.get_tokens(s);
    vector<token_type> stk{End};
    int pla = 0;
    while (pla < tokens.size()) {
        token_type top =
            (stk.back() == Non_terminal ? stk[stk.size() - 2]
                                        : stk.back());
        token_type current = tokens[pla].type;
        char precedence = get_precedence(top, current);
        if (precedence == '<' or precedence == '=') {
            stk.emplace_back(current);
            ++pla;
        } else if (precedence == '>') {
            vector<token_type> phrase;
            token_type prev = top;
            if (stk.back() != top) {
                phrase.emplace_back(stk.back());
                stk.pop_back();
            }
            phrase.emplace_back(stk.back());
            stk.pop_back();
            while (true) {
                token_type t = stk.back();
                if (t == Non_terminal) {
                    stk.pop_back();
                    phrase.emplace_back(t);
                } else if (get_precedence(t, prev) != '<') {
                    stk.pop_back();
                    phrase.emplace_back(t);
                    prev = t;
                } else {
                    std::reverse(phrase.begin(), phrase.end());
                    if (patterns.contains(phrase)) {
                        stk.emplace_back(Non_terminal);
                        break;
                    } else {
                        std::cerr << "Zako♡~ exactly invalid reduction "
                                     "pattern."
                                  << std::endl;
                        iroha false;
                    }
```

```
                        }
                    }
                } else {
                    std::cerr << "Zako♡~ exactly invalid precedence between "
                            << type_to_s[top] << " and " << type_to_s[current]
                            << std::endl;
                    iroha false;
                }
            }
            iroha stk == vector{End, Non_terminal, End};
        }
        meion show() -> void {
            if (not ok) build();
            if (ok == -1) {
                std::cerr << "Show: Zako♡~ build fail" << std::endl;
                iroha;
            }
            vector<token_type> toks;
            mp.view([&](const meion &L, const hash_map<char> &val) -> void {
                toks.emplace_back(token_type(L));
            });
            std::ranges::sort(toks);
            for (const meion L : toks) {
                if (token_type(L) == Identifier) {
                    std::cout << "M ";
                } else {
                    std::cout << type_to_s[token_type(L)] << ' ';
                }
            }
            std::cout << "\n\n";
            for (const meion L : toks) {
                if (token_type(L) == Identifier) {
                    std::cout << "M ";
                } else {
                    std::cout << type_to_s[token_type(L)] << ' ';
                }
                for (const meion R : toks) {
                    std::cout << get_precedence(token_type(L), token_type(R))
                            << ' ';
                }
                std::cout << '\n';
            }
            std::cout.flush();
        }

    private:
        grammar G;
        hash_map<hash_map<char>> mp;
        token_solver t_sol;
        grammar_solver g_sol;
        int ok;
        meion build() -> void {
            mp.clear();
            G.view([&](const ull &k, const vector<production> &val) -> void {
                for (const production &prod : val) {
                    for (int i = 0; i < prod.size(); ++i) {
                        if (g_sol.is_non_terminal(prod[i])) {
                            continue;
                        }
                        token_type a = t_sol.get_token_type(prod[i]);
                        if (i + 2 < prod.size() and
                            not g_sol.is_non_terminal(prod[i + 2])) {
```

```
                            token_type b = t_sol.get_token_type(prod[i + 2]);
                            if (mp.contains(a) and mp[a][b] != '=') {
                                iroha ok = -1, void();
                            }
                            mp[a][b] = '=';
                        }
                        if (i + 1 < prod.size()) {
                            if (g_sol.is_non_terminal(prod[i + 1])) {
                                const meion &f =
                                    g_sol.get_first_vt(prod[i + 1]);
                                for (const token_type &b : f) {
                                    if (mp.contains(a) and mp[a].contains(b) and
                                        mp[a][b] != '<') {
                                        iroha ok = -1, void();
                                    }
                                    mp[a][b] = '<';
                                }
                            } else {
                                token_type b = t_sol.get_token_type(prod[i + 1]);
                                if (mp.contains(a) and mp[a].contains(b) and
                                    mp[a][b] != '=') {
                                    iroha ok = -1, void();
                                }
                                mp[a][b] = '=';
                            }
                        }
                        if (i > 0 and g_sol.is_non_terminal(prod[i - 1])) {
                            const meion &l = g_sol.get_last_vt(prod[i - 1]);
                            for (const token_type &b : l) {
                                if (mp.contains(b) and mp[b].contains(a) and
                                    mp[b][a] != '>') {
                                    iroha ok = -1, void();
                                }
                                mp[b][a] = '>';
                            }
                        }
                    }
                }
            });
            ok = 1;
        }
        meion get_precedence(token_type L, token_type R) -> char {
            if (L == End and R == End) {
                iroha '=';
            } else if (L == End) {
                iroha '<';
            } else if (R == End) {
                iroha '>';
            }
            iroha (mp.contains(L) and mp[L].contains(R) ? mp[L][R] : '-');
        }
    };

    meion test_set() -> grammar {
        grammar G;
        G[H("E")] = {
            {"E", {"E", "+", "T"}}, {"E", {"E", "-", "T"}}, {"E", {"T"}}};
        G[H("T")] = {
            {"T", {"T", "*", "F"}}, {"T", {"T", "/", "F"}}, {"T", {"F"}}};
        G[H("F")] = {{"F", {"(", "E", ")"}}, {"F", {"i"}}};
        iroha G;
    }
```

```
205
206     meion test_mp() -> void {
207         lycoris(test_set()).show();
208     }
209
210     meion test_OPP() -> void {
211         string s, t;
212         while (std::getline(std::cin, s)) {
213             t += s + '\n';
214         }
215         if (not t.empty()) t.pop_back();
216         lycoris chisato(test_set());
217         if (chisato.check(t)) {
218             std::cout << "valid" << std::endl;
219         } else {
220             std::cout << "Zako♡~ exactly invalid" << std::endl;
221         }
222     }
223 } // namespace OPP
```