编译器设计专题实验四：
语法分析 SLR(1)分析表生成

计算机 2101 田濡豪 2203113234

1 环境配置

本人选择使用 Visual Studio Code 配合阿里云 PAI-DSW 完成实验，在阿里云中创建的 DSW 实例如下：

ID 作为本人凭据。

由于校园网带宽有限，随着代码库增大远程开发的延迟和响应性显著降低，本次实验主要在本地完成，**但是在运行测试程序时会一同展示本人的阿里云控制台界面作为凭据。**

2 实验内容（必做）

2.1 实验要求（用户需求）

目标：在实验三的基础上，对 ParsingTable 进行 SLR(1)冲突分析，生成 SLR(1)分析表。

输入：实验三的 LR(0)规范族、文法符号的 FOLLOW 集

输出：SLR(1)分析表（ACTION/GOTO 二维表）

SLR（1）分析引入了 FOLLOW 集 来辅助解决部分冲突。在遇到移进 - 归约或归约 - 归约冲突时，通过检查归约项目左部非终结符的 FOLLOW 集与当前输入符号的关系来确定操作。

2.2 实验设计

2.2.1 需求分析

显然该部分是语法分析器的一部分，其目的仍然是生成一个 ParsingTable。只是在生成过程中添加了 SLR(1)分析来尝试排除冲突。因此应当保持原有的程序不变，在语法分析接口中派生一个新的类 SLRParser。

SLR 冲突解决大致可以分为四个部分：

1. 计算 CFG 文法的 FOLLOW 集。

2. 在 ParsingTable 中找到所有的冲突。

3. 对每个冲突，尝试参考 FOLLOW 集来解决冲突。

4. 如果冲突解决全部成功，更新 ParsingTable，返回成功；否则返回失败。

从中可以抽象出几个单一职能模块：

1. CFG 文法分析器：计算一个给定文法的 FIRST 集和 FOLLOW 集。

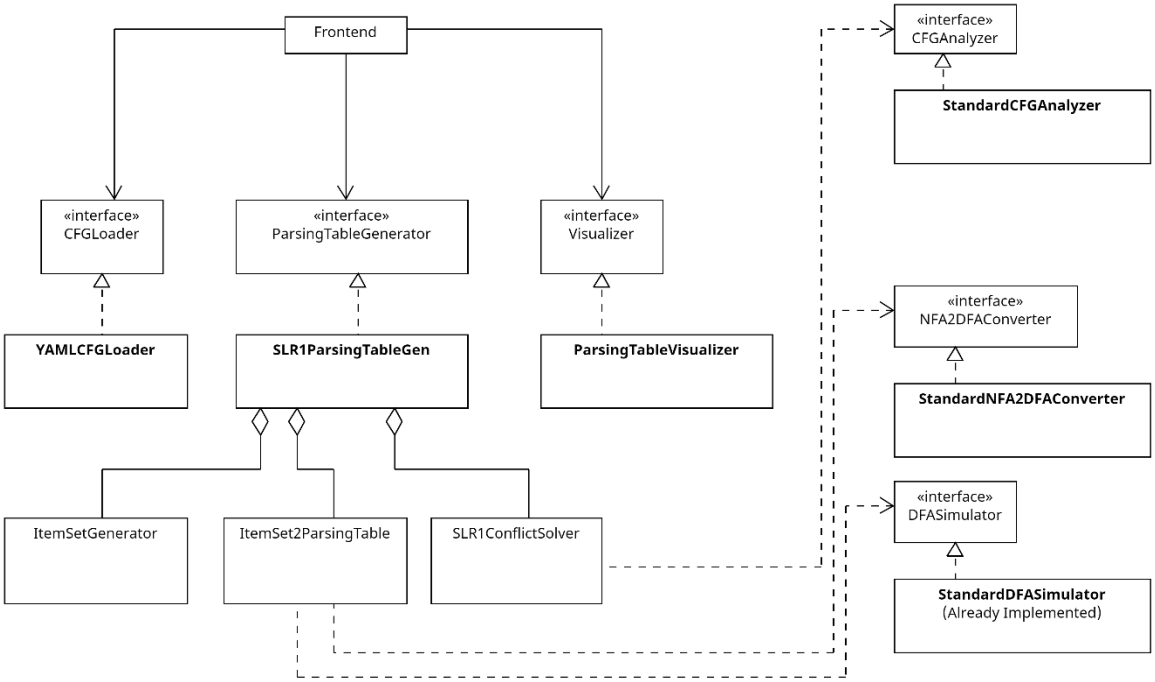
2. ParsingTable 冲突检查工具：检查并记录 ParsingTable 中的冲突。

3. SLR 冲突解决器：尝试解决 ParsingTable 中的冲突。

4. SLR 分析器：将上述三个模块结合起来，完成 SLR(1)分析表的生成。

2.2.2 整体架构

和实验三几乎相同，充分利用依赖倒置实现了一个新的 SLR（1）ParsingTable 生成器，依赖图如下：



2.2.3 关键数据模型设计

2.2.3.1 CFG 文法分析器

该模块的输入是 CFG 文法，输出是 FIRST 集和 FOLLOW 集。对于 FIRST 集，可以使用一个从`cfg_model::symbol`指向`unordered_set<symbol>`的`unordered_map`表示。此外，由于设计时不显式表示空符号，因此需要添加一个`unordered_set<symbol>`来表示 FIRST 集中拥有空符号的非终结符。

对于 FOLLOW 集，除了使用`unordered_map`表示外，还需要记录 CFG 文法的结束符号。

```
namespace cfg_model {
    struct CFG
    {
        cfg_model::symbol start_symbol;
        std::unordered_set<cfg_model::symbol> terminals;
        std::unordered_set<cfg_model::symbol> non_terminals;
        std::unordered_map<cfg_model::symbol, std::unordered_set<std::vect
or<cfg_model::symbol>>> production_rules;
        std::unordered_set<cfg_model::symbol> epsilon_production_symbols;
    };
    // FIRST set & FOLLOW set
    struct FirstSet
    {
```

```
        std::unordered_map<cfg_model::symbol, std::unordered_set<cfg_model
::symbol>> first_set;
        std::unordered_set<cfg_model::symbol> symbols_with_epsilon;
    };
    struct FollowSet
    {
        std::unordered_map<cfg_model::symbol, std::unordered_set<cfg_model
::symbol>> follow_set;
        cfg_model::symbol end_symbol;
    };
} // namespace cfg_model
```

2.2.3.2 ParsingTable 冲突检查工具

考虑到该工具对词法分析很常用，将其作为成员函数集成在`ParsingTable`类中。该工具的输入是 ParsingTable，输出是冲突列表。冲突列表可以使用一个`multimap<symbol, string>`表示，表示在相应的表格中存在冲突，使用 multimap 是因为一个 symbol 可能有多个 string 同时发生冲突。

2.2.3.3 SLR 冲突解决器

冲突解决需要以下几个方面的信息：

1. ParsingTable
2. 对应的 CFG 文法，用于计算 FIRST 集和 FOLLOW 集
3. 每个 ParsingTable 项对应的 LR(0)项目集合

对于前两者，可以直接传入；对于第三者，需要设计一个从`ParsingTable`到 LR(0)项目集合的映射。可以使用一个`unordered_map<string, unordered_map<symbol, unordered_set<shared_ptr<Item>>>>`表示。该映射的 key 为 ParsingTable 项的字符串表示，value 为该项对应的 LR(0)项目集合。由于 LR(0)项目集合是一个集合，因此可以使用`unordered_set<shared_ptr<Item>>`表示。特别的，这个映射应该在 ParsingTable 构造时一同产生。

```
    // map each parsing table cell to a set of items
    struct ItemSetParsingTableMapping
    {
        // Mapping from parsing table cell to ItemSet
        std::unordered_map<std::string, std::unordered_map<cfg_model::symb
ol, std::unordered_set<std::shared_ptr<Item>>>> parsing_table_cell_to_item
_set;
        // Mapping from ItemSet to parsing table cell
```

```
        std::unordered_map<std::shared_ptr<Item>, std::multimap<std::strin
g, cfg_model::symbol>> item_set_to_parsing_table_cell;
    };
```

2.3 实现细节

2.3.1 CFG 文法分析器

CFG 文法分析器接受一个 CFG 文法，提供计算 FIRST 集和 FOLLOW 集的功能。其中计算 FOLLOW 集要求优先计算 FIRST 集，否则报错。

2.3.1.1 计算 FIRST 集

计算 FIRST 集的算法就是经典的标准算法：

1. 对于每个 CFG 文法的非终结符号，初始化其 FIRST 集为空。

2. 对于每个 CFG 文法的终结符号，初始化其 FIRST 集为其本身。

3. 对于每个 CFG 文法的产生式，遍历其右侧符号序列：

   1. 如果该符号是终结符号，则将其加入 FIRST 集中，并退出。

   2. 如果该符号是非终结符号，则将其 FIRST 集中的所有非空符号加入到当前 FIRST 集中，并继续遍历下一个符号。

   3. 如果该符号是空产生式，则将空符号加入到当前 FIRST 集中，同时查看是否有下一个符号，如果有，将其 FIRST 集并入当前 FIRST 集中，退出。

4. 不断重复上述过程，直到所有非终结符号的 FIRST 集不再变化为止。

但是，注意到我们定义的 CFG 文法中，空产生式和非空产生式是分别存储在两个不同的 `unordered_set`中的，同时在 first 集的定义中，也没有显式指出空符号，而是单独使用一个`unordered_set<symbol>`来表示。因此在实现时不能简单遍历产生式：

1. 对于每个 CFG 文法的非终结符号，初始化其 FIRST 集为空。

2. 查找 CFG 的可空符号集，将对应的非终结符号加入 FIRST 集数据结构的含空符号 FIRST 集中。

3. 对于每个 CFG 文法的产生式，遍历其右侧符号序列：

1. 如果该符号是终结符号，则将其加入 FIRST 集中，并退出。

2. 如果该符号是非终结符号，则将其 FIRST 集中的所有非空符号加入到当前 FIRST 集中，并继续遍历下一个符号。

3. 如果该符号是空产生式，则将当前产生式的左部符号加入含空符号 FIRST 集中。接下来，查看是否有下一个符号，如果有，将其 FIRST 集并入当前 FIRST 集中，*并且还需要查看第二个符号是否在 CFG 的可空符号集中，如果在，还需要再次尝试将左部符号加入含空符号 FIRST 集中*（因为使用集合，所以不会有重复加入问题），退出。

4. 不断重复上述过程，直到所有非终结符号的 FIRST 集不再变化为止。

```
        // Initialize the first set: for each non-
terminal, the first set is empty, for each terminal, the first set is the
terminal itself
        for (const auto &non_terminal : cfg.non_terminals)
        {
            first_set.first_set[non_terminal] = std::unordered_set<cfg_mod
el::symbol>();
            // if the non-
terminal has epsilon production, add it to the epsilon set
            if (cfg.epsilon_production_symbols.find(non_terminal) != cfg.e
psilon_production_symbols.end())
            {
                first_set.symbols_with_epsilon.insert(non_terminal);
            }
        }
        for (const auto &terminal : cfg.terminals)
        {
            first_set.first_set[terminal] = std::unordered_set<cfg_model::
symbol>({terminal});
        }
        // start iterating over the production rules until no changes are
made
        bool changed = true;
        while(changed)
        {
            changed = false;
            // iterate over the non-espilon production rules
            for (const auto &rule : cfg.production_rules)
            {
                const cfg_model::symbol &lhs = rule.first;
                // iterate over all the candidate productions
                for (const auto &rhs : rule.second)
                {
                    const cfg_model::symbol &first_symbol = rhs[0];
                    // add every symbol in the first set of the first symb
ol to the first set of the lhs
                    if (first_set.first_set.at(first_symbol).size() == 0)
                    {
```

```
                        // the first set is still empty
                        continue;
                    }
                    int current_lhs_first_set_size = first_set.first_set[l
hs].size();
                    int current_epsilon_set_size = first_set.symbols_with_
epsilon.size();
                    // add the first set of the first symbol to the first
set of the lhs
                    first_set.first_set[lhs].insert(first_set.first_set[fi
rst_symbol].begin(), first_set.first_set[first_symbol].end());
                    // also, check if the first symbol set has epsilon
                    if (first_set.symbols_with_epsilon.find(first_symbol)
!= first_set.symbols_with_epsilon.end())
                    {
                        // 1. now an epsilon symbol must in the first set
of the first symbol, so add it to the first set of the lhs
                        first_set.symbols_with_epsilon.insert(lhs);
                        // 2. check if there is a second symbol in the rhs
                        if (rhs.size() > 1)
                        {
                            // 3. if there is a second symbol, add the fir
st set of the second symbol to the first set of the lhs
                            const cfg_model::symbol &second_symbol = rhs[1
];
                            // add the first set of the second symbol to t
he first set of the lhs
                            first_set.first_set[lhs].insert(first_set.firs
t_set[second_symbol].begin(), first_set.first_set[second_symbol].end());
                            // 4. !!! important: check if the second symbo
l has epsilon
                            if (first_set.symbols_with_epsilon.find(second
_symbol) != first_set.symbols_with_epsilon.end())
                            {
                                // 5. if the second symbol has epsilon, ad
d it to the first set of the lhs
                                first_set.symbols_with_epsilon.insert(lhs)
;
                            }
                        }
                    }
                    // check if the first set of the lhs has changed
                    if (first_set.first_set[lhs].size() != current_lhs_fir
st_set_size || first_set.symbols_with_epsilon.size() != current_epsilon_se
t_size)
                    {
                        changed = true;
                    }
                }

            }
        }
```

2.3.1.2 计算 FOLLOW 集

尽管不显示表示空符号在计算 FIRST 集时带来了额外复杂度，但是在计算 FOLLOW 集时，单独表示空符号使计算更加简单。

经典的计算 FOLLOW 集的算法如下：

1. 对于每个 CFG 文法的非终结符号，初始化其 FOLLOW 集为空。

2. 对于 CFG 文法的开始符号，初始化其 FOLLOW 集为结束符号。

3. 对于每个 CFG 文法的产生式，遍历其右侧符号序列：

   1. 如果符号是终结符，跳过并遍历下一个符号。

   2. 如果下一个符号是终结符，则将其加入当前非终结符的 FOLLOW 集中，并继续遍历下一个符号。

   3. 如果下一个符号是非终结符，则将其 FIRST 集中的所有非空符号加入当前非终结符的 FOLLOW 集中，并继续遍历下一个符号。

   4. 如果下一个符号是空产生式，或者下一个非终结符的 FIRST 集包含空符号，则将当前非终结符的 FOLLOW 集并入下一个非终结符的 FOLLOW 集中，并继续遍历下一个符号。

4. 不断重复上述过程，直到所有非终结符号的 FOLLOW 集不再变化为止。

由于单独存储空产生式，实际实现更加直观：

1. 对于每个 CFG 文法的非终结符号，初始化其 FOLLOW 集为空。

2. 对于 CFG 文法的开始符号，初始化其 FOLLOW 集为结束符号。

3. 对于每个 CFG 文法的产生式，遍历其右侧符号序列：

   1. 如果符号是终结符，跳过并遍历下一个符号。

   2. 如果下一个符号是终结符，则将其加入当前非终结符的 FOLLOW 集中，并继续遍历下一个符号。

   3. 如果下一个符号是非终结符，直接将其 FIRST 集加入当前非终结符的 FOLLOW 集中（因为不显式表示空符号），并继续遍历下一个符号。

4. 如果下一个符号是空产生式，或者下一个非终结符能在 FIRST 集数据结构的包含空符号 FIRST 集中找到，则将当前非终结符的 FOLLOW 集并入下一个非终结符的 FOLLOW 集中，并继续遍历下一个符号。

4. 不断重复上述过程，直到所有非终结符号的 FOLLOW 集不再变化为止。

```
        // clear the follow set
        follow_set.follow_set.clear();
        // find the end symbol
        cfg_model::symbol end_symbol;
        bool end_symbol_found = false;
        for (const auto &terminal : cfg.terminals)
        {
            if (terminal.special_property == "END")
            {
                end_symbol = terminal;
                end_symbol_found = true;
                break;
            }
        }
        if (!end_symbol_found)
        {
            throw std::runtime_error("End symbol not found in the CFG. Ple
ase add an end symbol to the CFG.");
        }
        // initialize the follow set for each non-terminal
        for (const auto &non_terminal : cfg.non_terminals)
        {
            follow_set.follow_set[non_terminal] = std::unordered_set<cfg_m
odel::symbol>();
        }
        // add the end symbol to the follow set of the start symbol
        follow_set.follow_set[cfg.start_symbol].insert(end_symbol);
        follow_set.end_symbol = end_symbol;
        // start iterating over the production rules until no changes are
made
        bool changed = true;
        while (changed)
        {
            changed = false;
            // iterate over the production rules
            for (const auto &rule : cfg.production_rules)
            {
                const cfg_model::symbol &lhs = rule.first;
                // iterate over all the candidate productions
                for (const auto &rhs : rule.second)
                {
                    std::string rhs_str = "";
                    for (const auto &symbol : rhs)
                    {
```

```
                        rhs_str += std::string(symbol) + " ";
                    }
                    spdlog::debug("Processing production rule: {} -> {}",
std::string(lhs), rhs_str);
                    // iterate over the symbols in the rhs
                    for (size_t i = 0; i < rhs.size(); i++)
                    {
                        const cfg_model::symbol &current_symbol = rhs[i];
                        // if is terminal, continue
                        if (current_symbol.is_terminal)
                        {
                            continue;
                        }
                        bool epsilon_suffix = false;
                        bool is_last_symbol = false;
                        // if the current symbol is the last symbol, then
it has epsilon suffix
                        if (i == rhs.size() - 1)
                        {
                            is_last_symbol = true;
                            epsilon_suffix = true;
                        }
                        // else if the next symbol is in the symbol with e
psilon set, then it has epsilon suffix
                        else if (first_set.symbols_with_epsilon.find(rhs[i
 + 1]) != first_set.symbols_with_epsilon.end())
                        {
                            spdlog::debug("Symbol {} has epsilon suffix",
std::string(current_symbol));
                            epsilon_suffix = true;
                        }
                        // else it does not have epsilon suffix
                        else
                        {
                            epsilon_suffix = false;
                        }
                        // if the current symbol has an epsilon suffix, th
en add the follow set of the lhs to the follow set of the current symbol
                        if (epsilon_suffix)
                        {
                            int current_follow_set_size = follow_set.follo
w_set[current_symbol].size();
                            // add the follow set of the lhs to the follow
 set of the current symbol
                            follow_set.follow_set[current_symbol].insert(f
ollow_set.follow_set[lhs].begin(), follow_set.follow_set[lhs].end());
                            // if not the last symbol, add the first set o
f the next symbol to the follow set of the current symbol
                            if (!is_last_symbol)
                            {
                                const cfg_model::symbol &next_symbol = rhs
[i + 1];
```

```
                                // add the first set of the next symbol to
 the follow set of the current symbol
                                follow_set.follow_set[current_symbol].inse
rt(first_set.first_set[next_symbol].begin(), first_set.first_set[next_symb
ol].end());
                            }
                            // check if the follow set of the current symb
ol has changed
                            if (follow_set.follow_set[current_symbol].size
() != current_follow_set_size)
                            {
                                changed = true;
                            }
                        }
                        // if the current symbol does not have an epsilon
 suffix, then add the first set of the next symbol to the follow set of the
 current symbol
                        else
                        {
                            // the symbol must not be the last symbol
                            const cfg_model::symbol &next_symbol = rhs[i +
 1];
                            // add the first set of the next symbol to the
 follow set of the current symbol
                            int current_follow_set_size = follow_set.follo
w_set[current_symbol].size();
                            // here our first set does not contain epsilon
, so we can add it directly
                            spdlog::debug("Adding first set of {} to follo
w set of {}", std::string(next_symbol), std::string(current_symbol));
                            follow_set.follow_set[current_symbol].insert(f
irst_set.first_set[next_symbol].begin(), first_set.first_set[next_symbol].
end());
                            // check if the follow set of the current symb
ol has changed
                            if (follow_set.follow_set[current_symbol].size
() != current_follow_set_size)
                            {
                                changed = true;
                            }
                        }
                    }
                }
            }
        }
```

2.3.2 ParsingTable 冲突检查工具

实现很简单，只需要遍历 Action 表中的每一个格子，然后将其中 Action 个数多于 1 的格子的坐标，即状态和输入符号，加入冲突列表中即可。

```
    std::multimap<std::string, cfg_model::symbol> LRParsingTable::find_conflicts() const
    {
        std::multimap<std::string, cfg_model::symbol> conflicts;
        int conflict_count = 0;
        for (const auto &state_pair : action_table)
        {
            const std::string &state = state_pair.first;
            for (const auto &symbol_pair : state_pair.second)
            {
                const cfg_model::symbol &symbol = symbol_pair.first;
                if (symbol_pair.second.size() > 1)
                {
                    spdlog::warn("Conflict found in action table for state {} and symbol {}: {} actions", state, std::string(symbol), symbol_pair.second.size());
                    conflicts.insert({state, symbol});
                    conflict_count++;
                }
            }
        }
        spdlog::debug("Found {} conflicts in action table", conflict_count);
        return conflicts;
    }
```

2.3.3 SLR 冲突解决器

尽管 SLR 冲突解决方法较为直观，然而在实现细节上需要一定考量，下面给出 SLR 冲突解决器的实现思路：

1. 初始化：复制一个新的 ParsingTable，作为 SLR 分析器的输出表
2. 调用冲突检查工具检查冲突，如果冲突坐标个数为 0 则直接退出，避免后续计算。
3. 调用 CFG 文法分析器计算 FIRST 集和 FOLLOW 集。
4. 开始遍历每一组冲突坐标：
    1. 对于每一个冲突坐标，获取该坐标对应的 LR(0)项目集合。
    2. 对 LR(0)项目集合进行第一次遍历，其职能是确定是否能解决冲突。
        1. 如果是移进项目，记录其下一个符号。

2. 如果是规约项目，记录其 FOLLOW 集。

3. 检查上述记录下的所有符号和 FOLLOW 集任意两者之间是否存在交集，如果是则退出整个程序并报告冲突无法解决。

4. 对 LR(0)项目集合进行第二次遍历，尝试解决冲突：

1. 如果当前冲突坐标的输入符号集合在移进项目中，则将该坐标的 Action 表项设置为移进项目。

2. 如果当前冲突坐标的输入符号集合在规约项目中，则将该坐标的 Action 表项设置为对应的规约项目。

5. 如果每一组冲突都被成功解决了，则将 SLR 分析器的输出表赋值给原 ParsingTable，返回成功；否则返回失败。

```
        spdlog::debug("Resolving conflicts in the parsing table...");
        // copy the parsing table to the resolved parsing table
        resolved_parsing_table.all_symbols = parsing_table.all_symbols;
        resolved_parsing_table.all_states = parsing_table.all_states;
        resolved_parsing_table.action_table = parsing_table.action_table;
        resolved_parsing_table.goto_table = parsing_table.goto_table;
        // get the conflicts in the parsing table
        auto conflicts = parsing_table.find_conflicts();
        // check if there are any conflicts
        if (conflicts.empty())
        {
            spdlog::debug("No conflicts found in the parsing table");
            return true; // no conflicts to resolve
        }
        else
        {
            spdlog::debug("Found {} conflicts in the parsing table", confl
icts.size());
        }
        // conflict solving preparation: find the follow sets
        CFGAnalyzer cfg_analyzer(cfg);
        cfg_analyzer.computeFirstSet();
        cfg_analyzer.computeFollowSet();
        const cfg_model::FollowSet &follow_set = cfg_analyzer.getFollowSet
();
        // iterate over the conflicts and resolve them
        for (const auto &conflict : conflicts)
        {
            const std::string &state = conflict.first;
            const cfg_model::symbol &symbol = conflict.second;
            spdlog::debug("Resolving conflict for state {} and symbol {}",
 state, std::string(symbol));
            // get the items in the conflict from mapping
```

```
            std::unordered_set<std::shared_ptr<lr_parsing_model::Item>> it
ems_in_conflict = item_set_parsing_table_mapping.parsing_table_cell_to_ite
m_set[state][symbol];
            spdlog::debug("Found {} items in the conflict", items_in_confl
ict.size());
            // get the shift symbols from the items
            std::unordered_set<cfg_model::symbol> shift_symbols = slr1_con
flict_resolver_helper::get_shift_symbols(items_in_conflict);
            int shift_symbols_count = shift_symbols.size();
            spdlog::debug("Found {} shift symbols in the conflict", shift_
symbols_count);
            // for clarity we iterate over the items two times, one for ch
ecking the conflict and one for resolving it
            // check if the conflict is resolvable by checking the interse
ction of the shift symbols and the follow sets of accepting items
            std::unordered_set<cfg_model::symbol> total_symbols;
            int expected_total_symbols_count = 0;
            for (const auto &item : items_in_conflict)
            {
                // check if the item is accepting
                if (item->is_accepting())
                {
                    // get the follow set of the item
                    std::unordered_set<cfg_model::symbol> current_item_fol
low_set = follow_set.follow_set.at(item->left_side_symbol);
                    // add the follow set to the total symbols
                    total_symbols.insert(current_item_follow_set.begin(),
current_item_follow_set.end());
                    expected_total_symbols_count += current_item_follow_se
t.size();
                }
            }
            // check if the shift symbols and the follow sets intersect
            if (expected_total_symbols_count != total_symbols.size())
            {
                // there is a conflict that cannot be resolved
                spdlog::info("Conflict at state {} and symbol {} cannot be
 resolved because the shift symbols and follow sets intersect", state, std
::string(symbol));
                return false;
            }
            // start resolving the conflict
            // get all potential actions in the conflict
            std::unordered_set<lr_parsing_model::Action> actions_in_confli
ct = resolved_parsing_table.get_actions(state, symbol);
            // clear the action table cell
            resolved_parsing_table.action_table[state].erase(symbol);
            // 1. check if the symbol is in the shift symbols
            bool resolved_by_shift = false;
            if (shift_symbols.find(symbol) != shift_symbols.end())
            {
                // find the shift action in the conflict
```

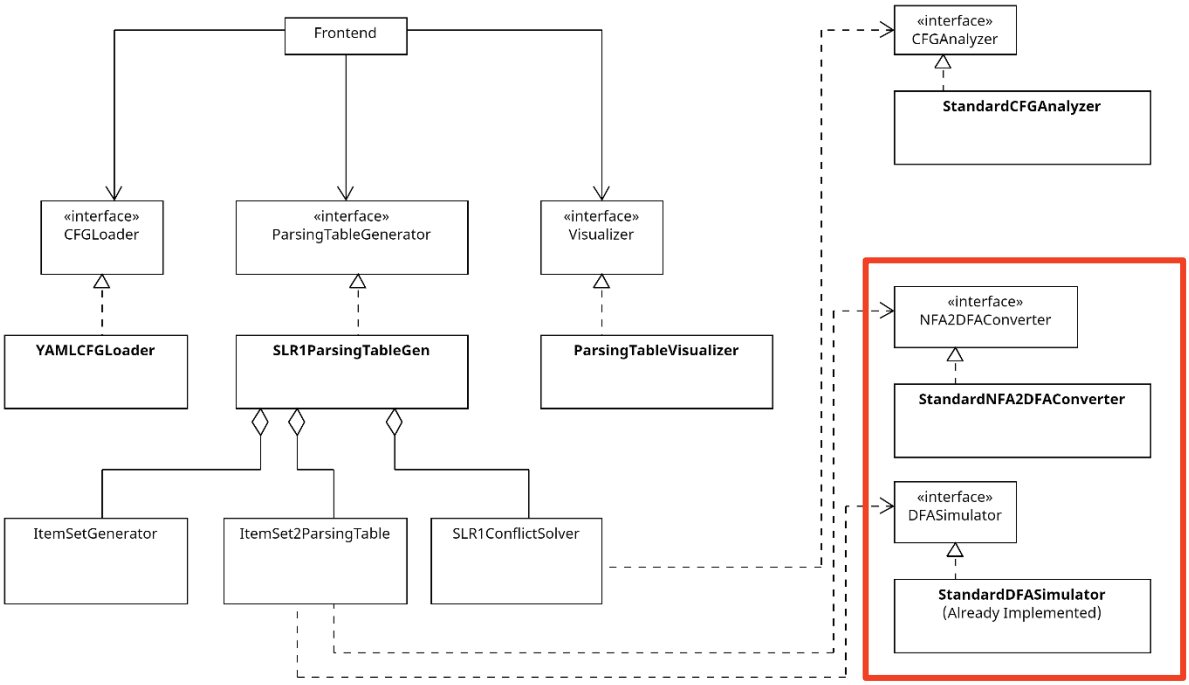
```
                bool shift_action_found = false;
                for (const auto &action : actions_in_conflict)
                {
                    if (action.action_type == "shift")
                    {
                        // add the shift action to the parsing table
                        resolved_parsing_table.add_action(state, symbol, a
ction);
                        shift_action_found = true;
                        spdlog::debug("Added shift action for state {} and
 symbol {}", state, std::string(symbol));
                        break;
                    }
                }
                if (!shift_action_found)
                {
                    // no shift action found, this is an error
                    std::string error_message = "Error: No shift action fo
und for state " + state + " and symbol " + std::string(symbol);
                    spdlog::error(error_message);
                    throw std::runtime_error(error_message);
                }
                resolved_by_shift = true;
            }
            if (resolved_by_shift)
            {
                spdlog::debug("Resolved conflict by shift for state {} and
 symbol {}", state, std::string(symbol));
                continue; // continue to the next conflict
            }
            // 2. if not resolved by shift, find the accepting item whose
follow set contains the symbol
            for (const auto &item : items_in_conflict)
            {
                // check if the item is accepting
                if (item->is_accepting())
                {
                    // get the follow set of the item
                    std::unordered_set<cfg_model::symbol> current_item_fol
low_set = follow_set.follow_set.at(item->left_side_symbol);
                    // check if the symbol is in the follow set
                    if (current_item_follow_set.find(symbol) != current_it
em_follow_set.end())
                    {
                        // create a reduce action
                        lr_parsing_model::Action reduce_action;
                        reduce_action.action_type = "reduce";
                        reduce_action.reduce_rule_lhs = item->left_side_sy
mbol;
                        reduce_action.reduce_rule_rhs = item->sequence_alr
eady_parsed;
                        // add the reduce action to the parsing table
```

```
                        resolved_parsing_table.add_action(state, symbol, r
educe_action);
                        spdlog::debug("Added reduce action for state {} an
d symbol {}", state, std::string(symbol));
                    }
                }
            }
        }
        spdlog::debug("Conflicts resolved successfully");
        return true; // all conflicts resolved
```

3 实验内容（选做）

3.1 复用实验一的 DFA

事实上得益于依赖倒置的接口分类，从实验二就开始复用实验一的 DFA 了。



从架构图也能看出，使用的一直都是实验一的 DFA 模型。

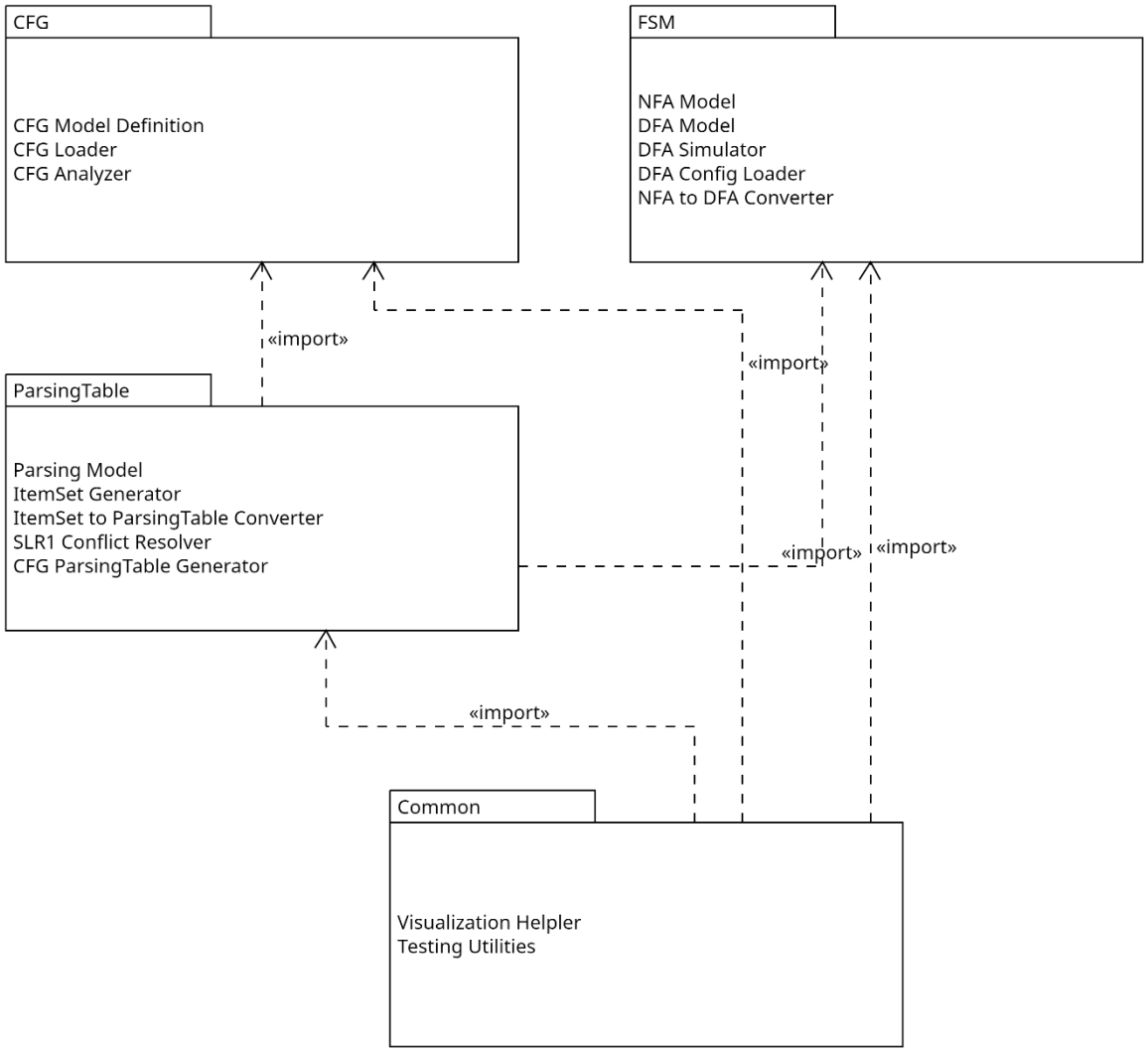
3.2 项目文件管理

实验指导 PPT 建议使用 Makefile，但是由于实验文件繁多，而且前三个实验均已经使用基于 Makefile 的 CMake 进行管理，本次实验决定还是使用 CMake。将实验文件按照职能整理如下：

1. CFG 库：包含 CFG 模型、加载器、分析器等

2. FSM 库：包含 NFA 与 DFA 模型、加载器、DFA 模拟器、NFA 到 DFA 转换器等。
3. ParsingTable 库：包含 LR 语法分析模型、ItemSet 生成器、ParsingTable 生成器、SLR1 冲突解决工具等。
4. 辅助工具库：包含测试用辅助函数（但不包含测试代码，测试代码另外存放）、可视化辅助函数等。

项目库依赖关系图如下：



整理后的文件目录如下：

```
lab4
 ├ demo
 |  ├ homework_8_4_cfg.yml
 |  ├ homework_8_5_cfg.yml
 |  └ lab123.yml
 ├ doc
```

```
│   ├ lab4_arch.uxf
│   ├ lab4_design.md
│   └ lab4_packages.uxf
├ include
│   ├ cfg
│   │   ├ cfg_analyzer.h
│   │   ├ cfg_loader.h
│   │   ├ cfg_model.h
│   │   └ yaml_cfg_loader.h
│   ├ common
│   │   ├ testing_utils.h
│   │   └ visualization_helper.h
│   ├ fsm
│   │   ├ dfa_config_frontend.h
│   │   ├ dfa_model.h
│   │   ├ dfa_simulator.h
│   │   ├ multitype_dfa_simulator.h
│   │   ├ nfa_dfa_converter.h
│   │   ├ nfa_model.h
│   │   ├ standard_nfa_dfa_converter.h
│   │   └ yaml_dfa_config_frontend.h
│   └ parsing_table
│   │   ├ itemset_generator.h
│   │   ├ itemset_to_parsing_table.h
│   │   ├ lr_parsing_model.h
│   │   ├ lr_parsing_table_generator.h
│   │   ├ simple_lr_parsing_table_generator.h
│   │   ├ slr1_conflict_resolver.h
│   │   └ slr1_parsing_table_generator.h
├ src
│   ├ cfg
│   │   ├ cfg_analyzer.cpp
│   │   └ yaml_cfg_loader.cpp
│   ├ common
│   │   ├ testing_utils.cpp
│   │   └ visualization_helper.cpp
│   ├ fsm
│   │   ├ multitype_dfa_simulator.cpp
│   │   ├ nfa_model.cpp
│   │   ├ standard_nfa_dfa_converter.cpp
│   │   └ yaml_dfa_config_frontend.cpp
│   └ parsing_table
```

```
│   │   ├ itemset_generator.cpp
│   │   ├ itemset_to_parsing_table.cpp
│   │   ├ lr_parsing_model.cpp
│   │   ├ simple_lr_parsing_table_generator.cpp
│   │   ├ slr1_conflict_resolver.cpp
│   │   └ slr1_parsing_table_generator.cpp
├ test
│   ├ cfg
│   │   ├ cfg_analyzer_tests.cpp
│   │   └ yaml_cfg_loader_tests.cpp
│   ├ common
│   │   ├ testing_utils.cpp
│   │   └ visualization_helper_tests.cpp
│   ├ data
│   │   ├ cfg
│   │   │   ├ cfg_analyzer
│   │   │   │   ├ cfg_1.yml
│   │   │   │   └ cfg_2.yml
│   │   │   └ yaml_cfg_loader
│   │   │   │   └ minimal_correct_cfg.yml
│   │   ├ common
│   │   │   └ visualization_helper
│   │   │   │   ├ complicated_cfg_1_with_conflict.yml
│   │   │   │   └ minimal_correct_cfg.yml
│   │   ├ fsm
│   │   │   └ multitype_dfa_simulator
│   │   │   │   └ dfa_config_real.yml
│   │   └ parsing_table
│   │   │   ├ itemset_generator
│   │   │   │   ├ cfg_inital_symbol_expanded_name_occupied.yml
│   │   │   │   ├ complicated_cfg_1_with_conflict.yml
│   │   │   │   └ minimal_correct_cfg.yml
│   │   │   ├ itemset_to_parsing_table
│   │   │   │   ├ minimal_correct_cfg.yml
│   │   │   │   └ minimal_correct_cfg_with_conflict.yml
│   │   │   ├ simple_lr_parsing_table_generator
│   │   │   │   ├ complicated_cfg_1_with_conflict.yml
│   │   │   │   └ minimal_correct_cfg.yml
│   │   │   └ slr1_parsing_table_generator
│   │   │   │   ├ complicated_cfg_1_with_conflict.yml
│   │   │   │   ├ homework_8_4_cfg.yml
│   │   │   │   └ minimal_correct_cfg.yml
```

```
│   ├ fsm
│   │   └ multitype_dfa_simulator_tests.cpp
│   └ parsing_table
│   │   ├ itemset_generator_tests.cpp
│   │   ├ itemset_to_parsing_table_tests.cpp
│   │   ├ simple_lr_parsing_table_generator_tests.cpp
│   │   └ slr1_parsing_table_generator_tests.cpp
├ CMakeLists.txt
└ frontend.cpp
```

按照文件和包依赖关系编写 CMakeList，完整的 CMakeList 如下：

```
cmake_minimum_required(VERSION 3.14)
project(Lab4_SLR1_Parsing_Table_Generator)
# yaml-cpp
find_package(yaml-cpp REQUIRED)
# Add find_package for spdlog
find_package(spdlog REQUIRED)
# boost graph library for visualization
find_package(Boost REQUIRED COMPONENTS graph)
# tabulate library for visualization
find_package(tabulate REQUIRED)
# --- Global Include Directory ---
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include)
# headers are in include/ subdirectories
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include/cfg)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include/fsm)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include/parsing_table)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include/common)
# --- Component Libraries ---
# CFG Utilities Library
add_library(cfg_utils STATIC
    src/cfg/yaml_cfg_loader.cpp
    src/cfg/cfg_analyzer.cpp
)
target_link_libraries(cfg_utils PUBLIC
    yaml-cpp # yaml_cfg_loader.cpp needs this
)
# Headers for this library are in include/cfg/
# FSM Utilities Library
add_library(fsm_utils STATIC
    src/fsm/nfa_model.cpp
    src/fsm/multitype_dfa_simulator.cpp
    src/fsm/standard_nfa_dfa_converter.cpp
    src/fsm/yaml_dfa_config_frontend.cpp
)
# Headers for this library are in include/fsm/
# Parsing Table Generation Utilities Library
add_library(parsing_table_utils STATIC
    src/parsing_table/itemset_generator.cpp
```

```
    src/parsing_table/itemset_to_parsing_table.cpp
    src/parsing_table/simple_lr_parsing_table_generator.cpp
    src/parsing_table/lr_parsing_model.cpp
    src/parsing_table/slr1_conflict_resolver.cpp
    src/parsing_table/slr1_parsing_table_generator.cpp
)
# Headers for this library are in include/parsing_table/
# this library requires fsm_utils and cfg_utils
target_link_libraries(parsing_table_utils PUBLIC
    fsm_utils
    cfg_utils
)
# Visualization Utilities Library
add_library(viz_utils STATIC
    src/common/visualization_helper.cpp
)
# this library requires all previous libraries
target_link_libraries(viz_utils PUBLIC
    cfg_utils
    fsm_utils
    parsing_table_utils
)
target_link_libraries(viz_utils PUBLIC
    Boost::graph
    tabulate::tabulate
)
# Headers for this library are in include/common/ (or include/viz/)
# Test Utilities Library
add_library(test_utils STATIC
    src/common/testing_utils.cpp
)
# this library requires all previous libraries
target_link_libraries(test_utils PUBLIC
    cfg_utils
    fsm_utils
    parsing_table_utils
    viz_utils
)
# --- GoogleTest ---
include(FetchContent)
FetchContent_Declare(
  googletest
  URL https://github.com/google/googletest/archive/refs/tags/v1.14.0.zip
  DOWNLOAD_EXTRACT_TIMESTAMP TRUE
)
FetchContent_MakeAvailable(googletest)
enable_testing() # Enable CTest support
# --- Test Executable ---
set(TEST_DATA_DIR ${CMAKE_CURRENT_SOURCE_DIR}/test/data)
set(TEST_DATA_DEST ${CMAKE_CURRENT_BINARY_DIR}/test/data)
add_executable(test_all
    test/cfg/cfg_analyzer_tests.cpp
```

```
    test/cfg/yaml_cfg_loader_tests.cpp
    test/fsm/multitype_dfa_simulator_tests.cpp
    test/parsing_table/itemset_generator_tests.cpp
    test/parsing_table/itemset_to_parsing_table_tests.cpp
    test/parsing_table/simple_lr_parsing_table_generator_tests.cpp
    test/parsing_table/slr1_parsing_table_generator_tests.cpp
    test/common/visualization_helper_tests.cpp
)
target_link_libraries(test_all PRIVATE
    cfg_utils
    fsm_utils
    parsing_table_utils
    viz_utils
    test_utils
    gtest_main
    gtest
    spdlog::spdlog # Assuming tests might use logging
    # yaml-cpp is linked via cfg_utils
)
add_custom_command(
    TARGET test_all
    POST_BUILD
    COMMAND ${CMAKE_COMMAND} -E copy_directory
    ${TEST_DATA_DIR}
    ${TEST_DATA_DEST}
    COMMENT "Copying test data for test_all"
)
include(GoogleTest)
gtest_discover_tests(test_all)
# --- Main Frontend Executable ---
add_executable(frontend
    frontend.cpp
    # frontend.cpp links against component libraries
)
target_link_libraries(frontend PRIVATE
    cfg_utils
    fsm_utils
    parsing_table_utils
    viz_utils
    spdlog::spdlog
    # yaml-
cpp, Boost::graph, tabulate::tabulate are linked via component libs
)
```

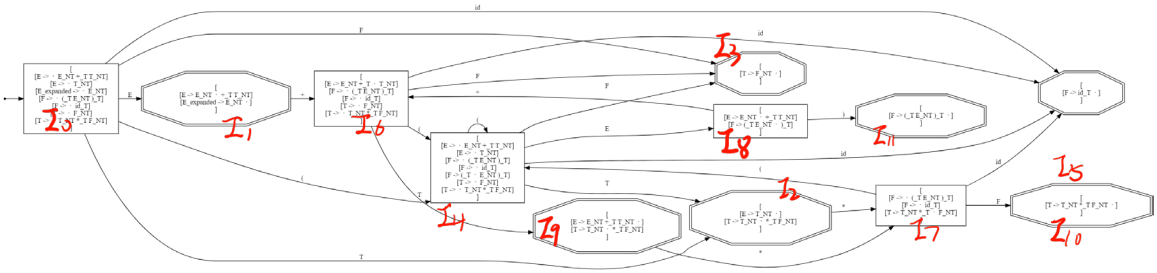
复现提示：Boost 和 Tabulate 包是需要手动 make install 到系统环境中的，当然这两个库被设计为支持 header-only，但那就没法直接用上面这份 CMake，因为我是安装好后针对系统环境包写的。

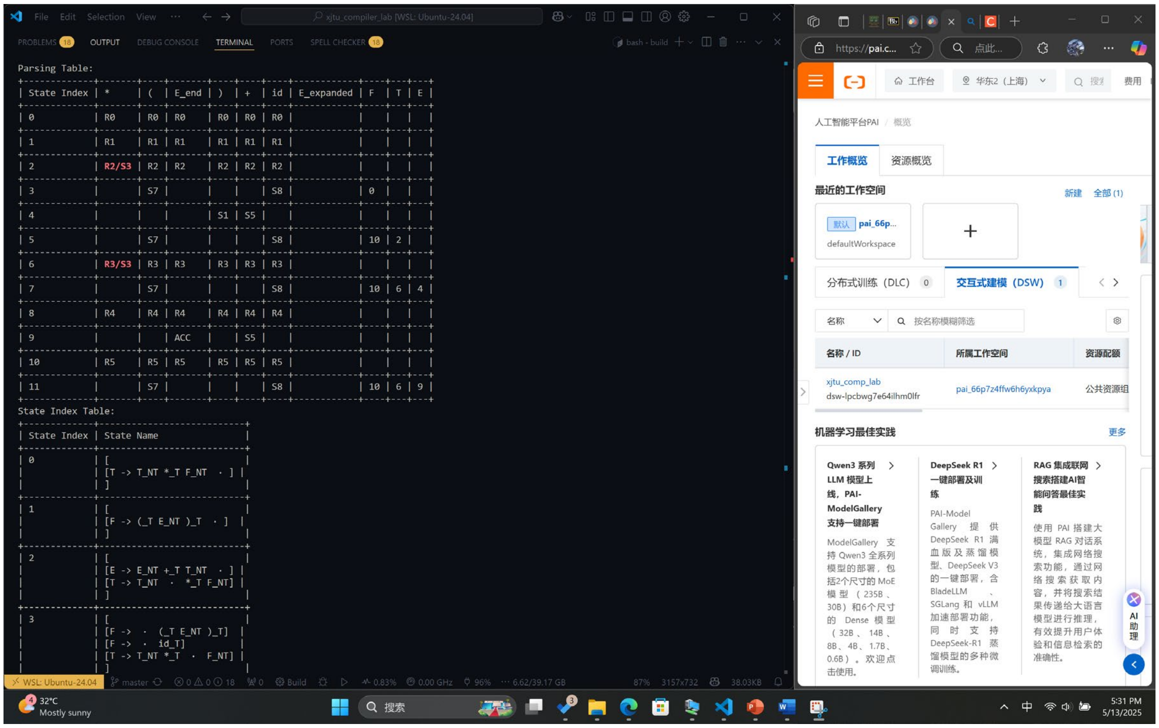
4 实验结果

本次的代码库基于实验三，因此大部分单元测试和实验三相同，此处直接给出可视化结果。

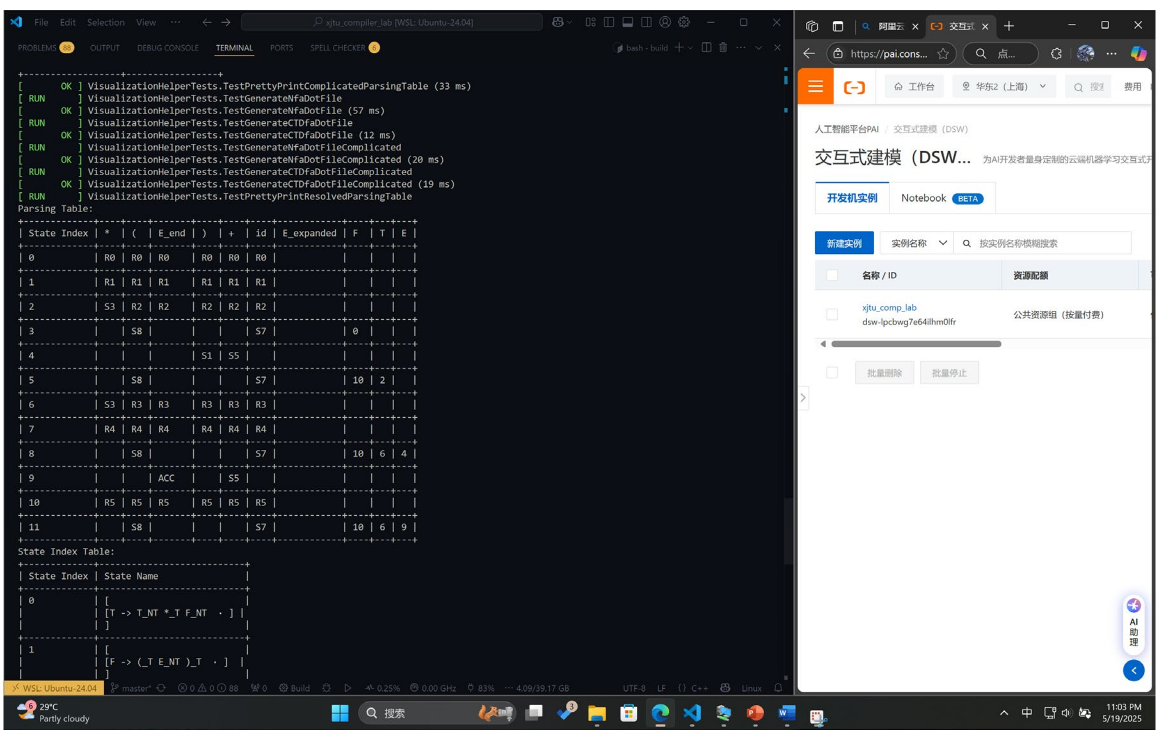
输入

$$S' \rightarrow E$$
$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$





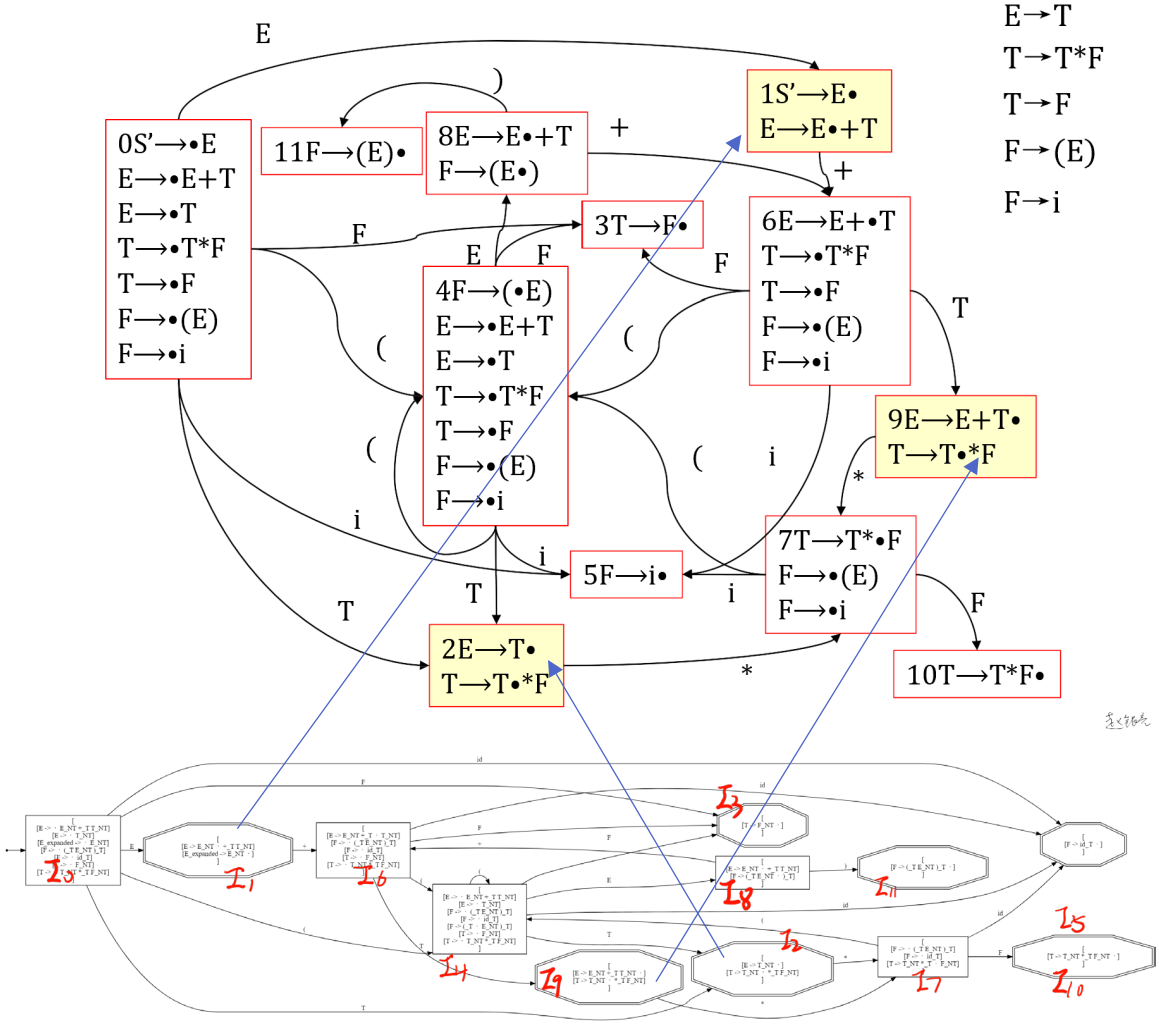
以上后两张截图取自实验三报告，在实验三中产生了带有冲突的分析表，请参考这两个冲突的发生位置以便比较冲突解决结果。接下来展示使用 SLR1 分析表产生工具得到无冲突的分析表。



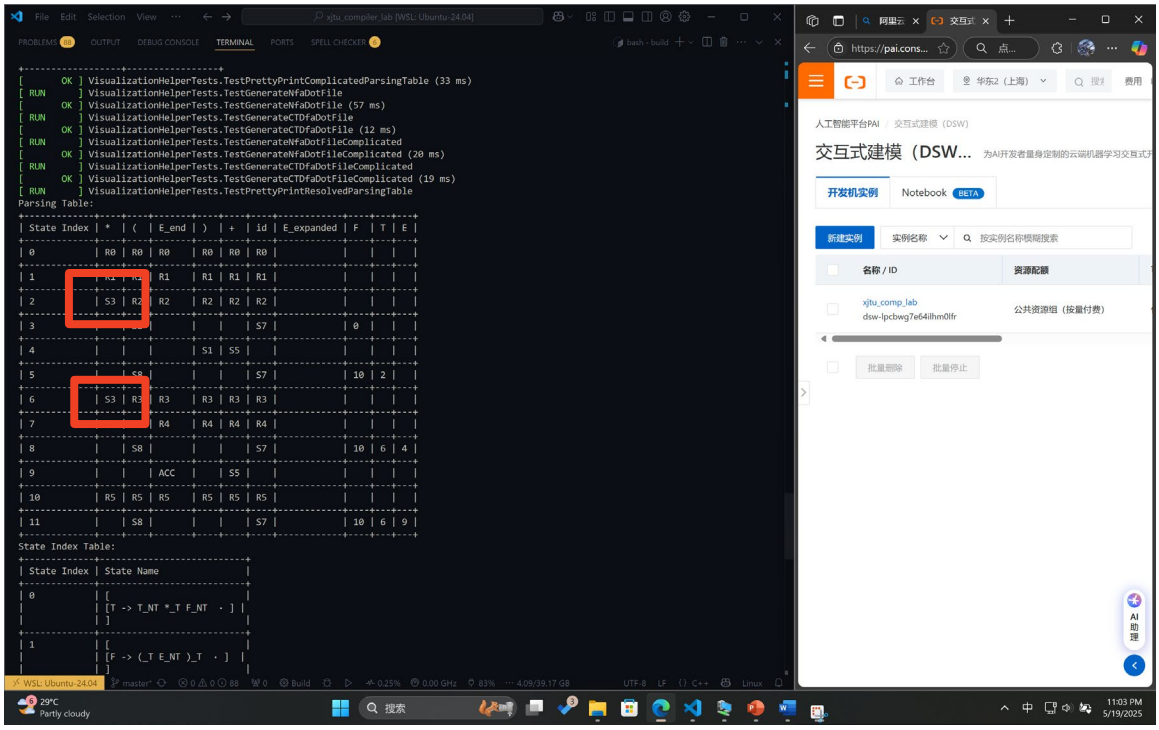
接下来按照 PPT 校验冲突解决：

例：构造SLR(1)分析表

S→E
E→E+T
E→T
T→T*F
T→F
F→(E)
F→i



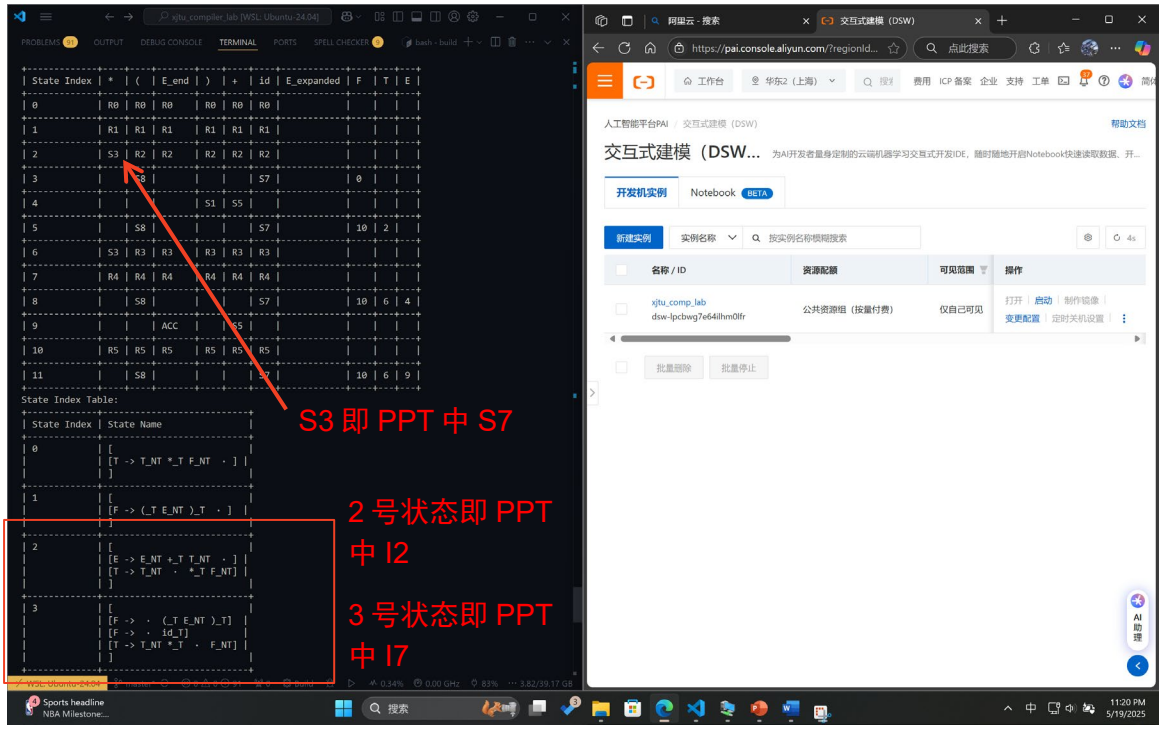
此处值得说明的是，I1 其实不存在冲突，因为 ACC ACTION 并不会添加到除了 END 符号之外的任何单元内，而 END 符号不可能出现 SHIFT ACTION。这就是为什么 PPT 中列写了三个冲突而上图分析表中只有两个红色单元格。PPT 中是为了展示清晰考量。



例：冲突消解与填表

0: S→E
1: E→E+T
2: E→T
3: T→T*F
4: T→F
5: F→(E)
6: F→i

- 规范集0没有冲突，直接填表
- ACTION[0,(]=s5；ACTION[0,i]=s4
- GOTO[0,E]=1；GOTO[0,T]=2；GOTO[0,F]=3
- 规范集1移进-归约冲突
- +∉FOLLOW(S)，ACTION[1,+]=s6，
- FOLLOW(S)={#}，ACTION[1,#]=acc
- 规范集2移进-归约冲突
- FOLLOW(E)={#,+,)}，
  ACITON[2,#]=ACTION[2,+]=ACTION[2,)]=r2；
- ACTION[2,*]=s7
- 规范集3无冲突
- FOLLOW(T)={+,*,),#}，
  ACTION[3,+]=ACTION[3,*]=ACTION[3,)]=ACTION[3,#]=r4

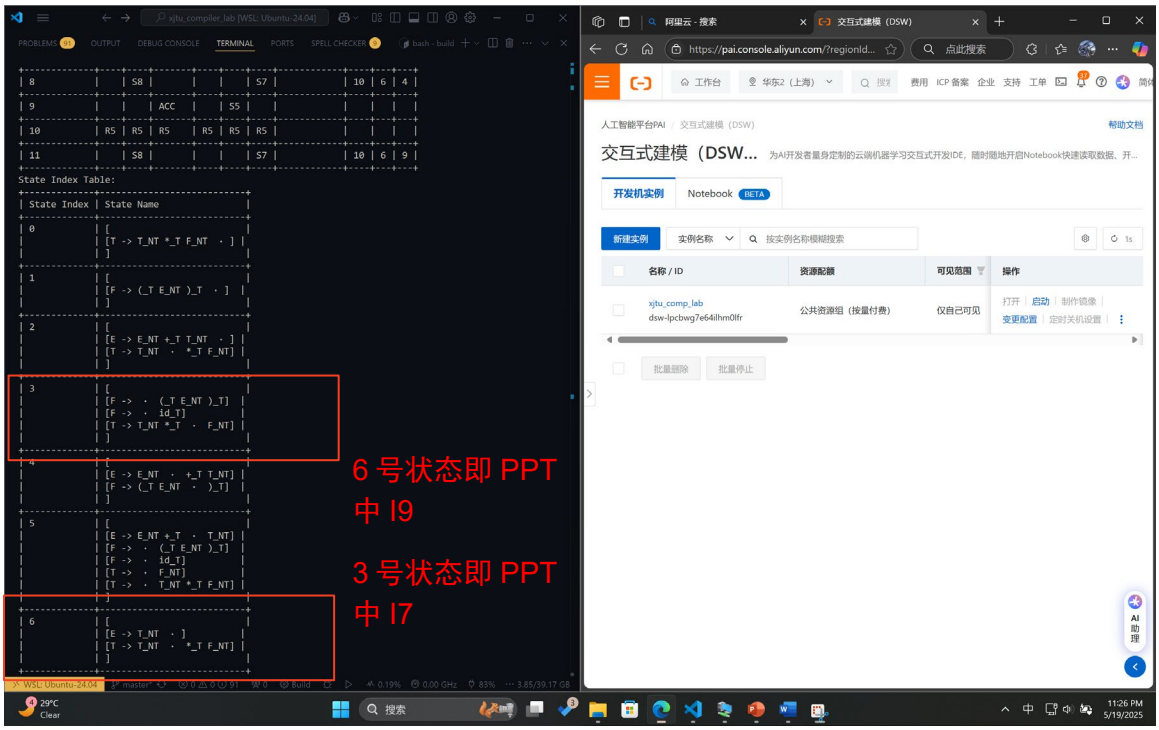


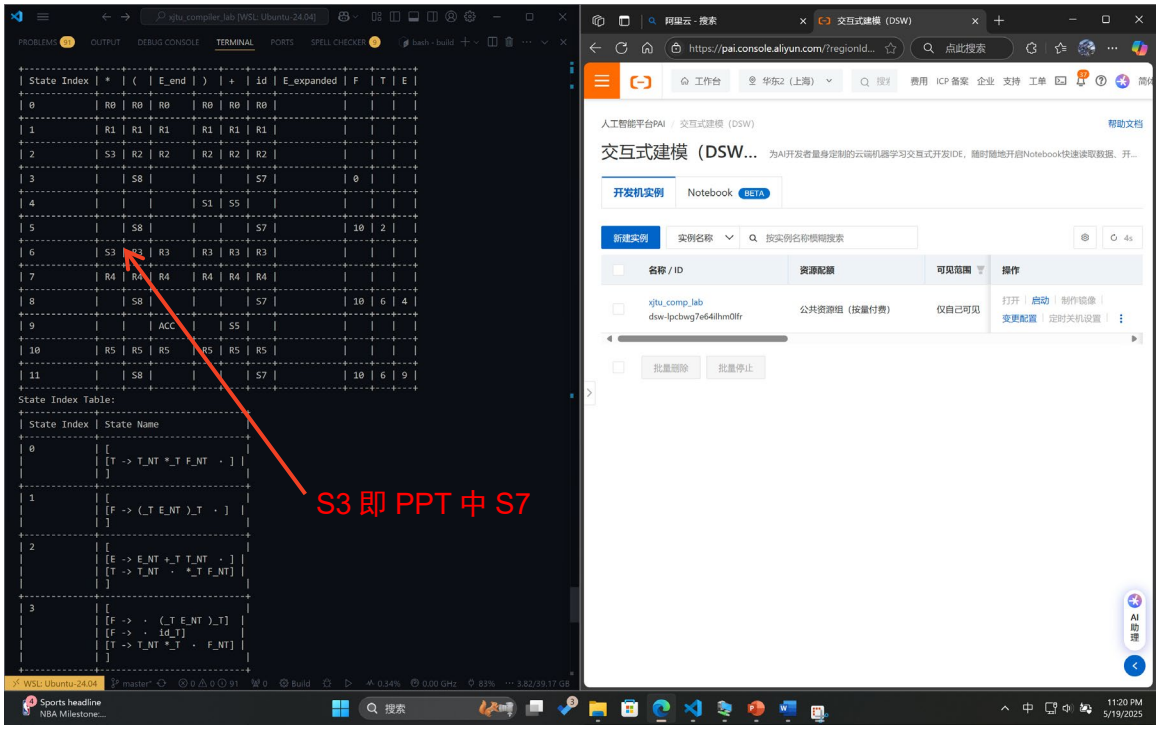
可见这个冲突解决是正确的。下面考虑 LR0 表格中的另一个冲突。

例：冲突消解与填表

0: S→E
1: E→E+T
2: E→T
3: T→T*F
4: T→F
5: F→(E)
6: F→i

- 项目集8没有冲突，直接填表
- ACTION[8,)]=s11；ACTION[4,+]=s6
- GOTO[4,E]=8；GOTO[4,T]=2；GOTO[4,F]=3
- 项目集9移进-归约冲突
- FOLLOW(E)={+,),#}， ACTION[9,*]=s7
- ACTION[9,+]=ACTION[9,)]=ACTION[9,#]=r1
- 项目集10无冲突
- FOLLOW(T)={+,*,),#}，ACTION[10,+]=ACTION[10,*]=ACTION[10,)]=ACTION[10,#]=r3
- 项目集11无冲突
- FOLLOW(T)={+,*,),#}，ACTION[11,+]=ACTION[11,*]=ACTION[11,)]=ACTION[11,#]=r5





可见两个冲突都被成功解决，前文已经说过 ACC 的冲突解决是自然的。