编译器设计专题实验三：
语法分析 LR(0)规范集设计

计算机 2101 田濡豪 2203113234

1 目录

2 环境配置.....1
3 实验内容（必做）.....2
3.1 实验要求（用户需求）.....2
3.2 实验设计.....2
3.2.1 需求分析.....2
3.2.2 整体架构.....3
3.2.3 关键数据模型设计.....4
3.3 实现细节.....12
3.3.1 输入模块.....12
3.3.2 项目集与规范族产生器.....13
3.3.3 NFA-DFA 转换器.....21
3.3.4 可视化部分.....26
4 实验内容（选做）.....26
5 实验结果.....27
5.1 项目集产生.....29
5.2 NFA 产生.....30
5.3 带有冲突的 DFA 产生.....32
5.4 ParsingTable 及冲突检测.....34
6 附录.....38

2 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。**

3 实验内容（必做）

3.1 实验要求（用户需求）

目标：实现一个经典语法分析器的前导组件，构建给定输入文法的 LR(0)项目集规范族（Canonical Collection of LR(0) Items），为后续生成 SLR(1)分析表提供基础。

要求：

- 输入文法规则，支持扩展巴科斯范式（如 E → E + T | T）。
- 需要手动增广文法：添加新的起始产生式 S' → S（S 为原文法起始符号）。
- 要求生成 LR(0)项目集规范族：计算每个状态的闭包（Closure）和状态转- 移（Goto）。
- 输出所有项目集及其内核项（Kernel Items）。
- 规范族的图形化表示（如状态编号和转移关系）。
- 需要考虑是否属于 LR(0)文法（无移进-归约或归约-归约冲突）

3.2 实验设计

3.2.1 需求分析

可以将用户的需求大致分为三个部分：

1. 输入文法规则

2. 计算 LR(0)项目集规范族

3. 输出项目规范族

下面尝试将这些需求拆分为单一职能的业务模块。

输入文法规则：

- 需要定义一种存储格式（计划使用 YAML 文件）对文法进行描述。

- 需要定义一个中间数据模型，在程序中对文法进行存储和操作。

- 需要实现一个解析器，将 YAML 文件解析为中间数据模型。

- 需要在输入后对文法进行合法性检查，确保文法符合 LR(0)的要求。

计算 LR(0)项目集规范族：

- 需要一个项目集合生成装置，增广 CFG 文法，并生成 LR(0)项目集 itemset。

- 需要定义一个 NFA 数据模型，并实现 NFA 向 DFA 的转换（闭包计算），以利用先前实验中实现的 DFA 模拟器。同时需要考虑到 NFA 和 DFA 之间的状态对应关系。

- 需要实现一个规范族生成器，该生成器接受文法定义和 itemset 作为输入，调用 NFA 模拟器生成 itemNFA 和对应的 itemDFA。然后根据原始文法和 DFA 的关系，参考 DFA 生成 LR(0)项目集规范族。

输出项目规范族：

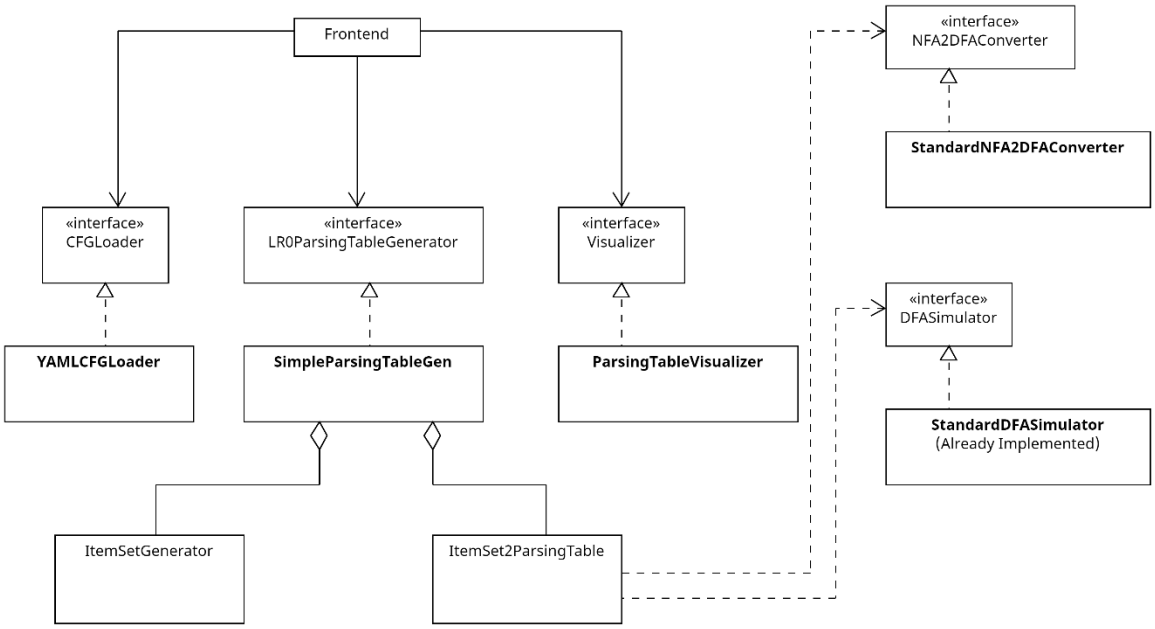
- 需要根据状态转移关系判断是否为 LR(0)文法。

- 需要输出生成的项目集，并标出其中的内核项。

- 需要将生成的项目集规范族进行图形化表示，如表示成一张状态转移图。

3.2.2 整体架构

在需求分析中确定的三个模块职能是单一的。因此可以设计一个总体控制程序作为前端，其保留有三个接口来倒置依赖于这三个模块。在执行时先调用输入模块接口获得文法定义，接着调用计算模块接口生成 LR(0)项目集规范族，最后调用输出模块接口输出项目集规范族。这样可以将三个模块的实现细节封装在各自的模块中，前端只需要关注接口的使用。



如图所示，CFGLoader 为输入模块接口，并由此派生实现 YAMLCFGLoader 类用于读取输入。

LR0ParsingTableGenerator 为计算模块接口，并由此派生实现 SimpleLR0ParsingTableGenerator 类用于生成 LR(0)项目集规范族。其由两个独立类聚合：ItemSetGenerator 和 ItemSet2ParsingTable。前者负责将输入文法转换成一个 ItemSet，后者负责从 ItemSet 生成 LR(0)项目集规范族。其中 ItemSet2ParsingTable 类的实现依赖于 NFA2DFAConverter 类和 DFASimulator 类。

3.2.3 关键数据模型设计

为了确定接口即后续程序设计的实现细节。首先对一些整个程序共有的关键数据模型进行设计。

其中包括：

- CFG 文法规则

- LR(0)项目集 ItemSet

- NFA

- LR(0)项目集规范族 ParsingTable

此外，还需要设计一个 YAML 文件格式，用于存储和加载 CFG 文法规则。

3.2.3.1 CFG 文法规则

一个上下文无关文法包含四个部分

- 终结符号（类别）集合，对本实验只要划分出类别即可，具体符号和类别的判断由词法分析器提供。

- 非终结符号集合。

- 产生式集合。

- 一个起始符号。

为了程序效率和可读性，作如下约定：

- 每个 CFG 的符号都有两个属性：符号名称和符号类别，符号名称使用字符串表示，符号类别使用 bool 表示是否为终结符号。

- 尽管有显式符号类别，依然规定终结符号的名称不能和非终结符号的名称相同。

因此一个符号定义如下：

```
namespace cfg_model
{
    struct symbol
    {
        std::string name = "";
        bool is_terminal = false;
        std::string special_property = "";
        // overload the equality operator
        bool operator==(const symbol &other) const
        {
            return name == other.name && is_terminal == other.is_terminal
&& special_property == other.special_property;
        }
        // string representation
        operator std::string() const
        {
            std::string result = name;
            result += std::string(is_terminal ? "_T" : "_NT");
            result += special_property.empty() ? "" : "_" + special_proper
ty;
            return result;
        }
```

```
    };
} // namespace cfg_model
```

请特别注意：因为终结符和非终结符都可以使用标准字符串表示，所以在打印 symbol 时会根据其类别和属性加上相应的后缀（见上文代码），比如一个非终结符 E，在后续状态打印中显示为 E_NT，即 non-terminal。

在 CFG 模型中，使用 unordered_set 来存储中介符号和非终结符号集合，并使用 unordered_map 来存储产生式集合。产生式集合的 key 为非终结符号，value 为一个 vector，vector 中存储该非终结符号的所有产生式。每个产生式使用 vector 表示，vector 中的元素为符号名称。

对于一个产生式（此处指没有经过化简，右侧只有单一候选式的产生式），其左边是一个非终结符号，右边是一个符号序列。符号序列使用 vector 表示，而产生式集合使用 unordered_map 存储，其中 key 为非终结符号，value 为一个 vector，vector 中存储该非终结符号的所有产生式。

此外还有一点需要注意，那就是 CFG 文法中的空产生式。为了清晰期间，我们将空产生式和非空产生式单独存储，即产生式集合分为一个 unordered_map 和一个 unordered_set。unordered_map 存储非空产生式，unordered_set 存储可以为空的产生式左侧符号。

CFG 文法定义如下：

```
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;
    };
} // namespace cfg_model
```

3.2.3.2 LR(0)项目集 ItemSet

ParsingTableGenerator 产生规范族的第一步是将输入的 CFG 文法进行扩展，产生其对应的项目集合。

项目集合中的每个项目形式和产生式相似，但是引入了一个点符号（.）来表示当前分析的进度。为了更清晰的表示其含义，我们将一个 Item 定义为一个包含以下部分的结构体：

- 产生式左侧符号

- 已经分析的符号序列，存储在 vector 中

- 尚未分析的符号序列，存储在 vector 中

因此 ItemSet 可以看作是一个 Item 的集合。为了解耦 ItemSet 和 Item 的关系，ItemSet 使用 unordered_set 来存储 Item 集合。

除了 Item 集合之外，ItemSet 还需要标出初始状态（此处指增广后文法的初始状态）对应的两条产生式：未分析的产生式和已经分析的产生式。这两条产生式将在后续的 LR0 分析下推自动机中标记初始状态和最终接受状态。

单个 Item 的定义如下：

```
    struct Item
    {
        cfg_model::symbol left_side_symbol;
        std::vector<cfg_model::symbol> sequence_already_parsed;
        std::vector<cfg_model::symbol> sequence_to_parse;
        // overload the equality operator
        bool operator==(const Item &other) const
        {
            return left_side_symbol == other.left_side_symbol &&
                   sequence_already_parsed == other.sequence_already_parse
d &&
                   sequence_to_parse == other.sequence_to_parse;
        }
        // string representation
        operator std::string() const;
    };
```

ItemSet 的定义如下：

```
    struct ItemSet
    {
        std::unordered_set<std::shared_ptr<Item>> items;
        std::shared_ptr<Item> start_item;
        std::shared_ptr<Item> end_item;
        std::unordered_set<cfg_model::symbol> symbol_set;
    };
```

3.2.3.3 NFA

考虑到单一职责原则，此处的 NFA 是一个通用 NFA 模型，并不包含 LR(0)项目集的相关信息。

不同于 DFA，NFA 的转移涉及到 ε -转移。为了清晰期间，我们将非空转移和空转移分别存储，因此***最初设计时***一个 NFA 包含以下部分：

- 状态集合: `unordered_set<string>`

- 字符集合：`unordered_set<char>`

- 起始状态: `string`

- 接受状态集合: `unordered_set<string>`

- 非空转移函数: `unordered_map<string, unordered_map<char, unordered_set<string>>>`

- 空转移函数: `unordered_map<string, unordered_set<string>>`

但是，考虑到实际上 LR 分析时实际上不计算*具体字符（计算具体字符的此法类别是词法分析器的任务）*，而是直接从词法分析获取输入的词法类别表进行判断，使用 char 来表示词法类别并不合适，因此采用了一种扩充定义的 NFA，允许使用 string 标识单个字符（它仍可以像标准 NFA 一样工作，只需要在字符和 string 间建立一个映射即可）

```
namespace nfa_model {
    struct NFA {
        std::unordered_set<std::string> states;
        std::unordered_set<std::string> character_set;
        std::string start_state;
        std::unordered_set<std::string> accepting_states;
        std::unordered_map<std::string, std::unordered_map<std::string, st
d::string>> non_epsilon_transitions;
        std::multimap<std::string, std::string>epsilon_transitions;
        int count_non_epsilon_transitions() const {
            int count = 0;
            for (const auto& ne_transition : non_epsilon_transitions) {
                count += ne_transition.second.size();
            }
            return count;
        }
    };
};
namespace nfa_model_helper {
    // helper function to check if the NFA is valid
    bool check_nfa_configuration(const nfa_model::NFA& nfa);
```

```
}
```

3.2.3.4 扩展 DFA 模型

对于一个理想状态下的 LR 语法分析，普通 DFA 就足以承担 NFA-DFA-ParsingTable 的转换任务。但是，正如上文中 NFA 遇到的问题一样：终结符使用 char 类型表示非常不方便，因此对 DFA 模型进行扩展，设置其为模板结构，在兼容前两个实验的同时允许今后的实验以 string 表示符号。

```
template<typename T>
struct DFA {
    std::unordered_set<T> character_set;                          // 字符集
    std::unordered_set<std::string> states_set;                   // 状态集
    std::string initial_state;                                    // 开始状态
    std::unordered_set<std::string> accepting_states;             // 接受状态集
    std::unordered_map<std::string, std::unordered_map<T, std::string>> transitions; // 状态转换表
};
```

3.2.3.5 LR(0)项目集规范族 ParsingTable

LR(0)项目集规范族 ParsingTable 是 LR(0)分析下推自动机的核心部分。其主要包含 ACTION 表和 GOTO 表。ACTION 表用于存储状态转移信息，GOTO 表用于存储状态转移的目标状态。事实上，ACTION 表和 GOTO 表只需要额外记录一份 CFG 的符号/状态表即可。

在 ACTION 表中，每个表格的值（假设没有冲突）可能拥有以下几种不同的涵义：

- 移进（Shift）：表示当前状态可以转移到下一个状态。

- 归约（Reduce）：表示当前状态可以归约为一个产生式。

- 接受（Accept）：表示当前状态可以接受输入串。

- 空（Empty）：表示当前状态没有任何转移。

因此我们可以将一个 ACTION 如下表示：

- 动作类型: `string`

- 目标状态: `string`

- 产生式: 左符号和右符号向量

```
    struct Action
    {
        std::string action_type = "";
        std::string target_state = "";                      // if the act
ion type is "shift"
        cfg_model::symbol reduce_rule_lhs;                  // if the act
ion type is "reduce"
        std::vector<cfg_model::symbol> reduce_rule_rhs = {}; // if the act
ion type is "reduce"
        // ==
        bool operator==(const Action &other) const
        {
            return action_type == other.action_type &&
                   target_state == other.target_state &&
                   reduce_rule_lhs == other.reduce_rule_lhs &&
                   reduce_rule_rhs == other.reduce_rule_rhs;
        }
    };
```

因此 ACTION 表可以表示为一个`unordered_map<string, unordered_map<symbol>, unordered_set<Action>>>`，其中 key 为当前状态，value 为一个 unordered_map，key 为输入符号。考虑到冲突的情况，value 为一个 unordered_set，存储所有可能的动作。

GOTO 表的表示方式和 ACTION 表类似，由当前状态和输入状态确定一个 unordered_set，存储所有可能的转移状态：`unordered_map<string, unordered_map<symbol, unordered_set<string>>>`。

因此一个 ParsingTable 可以表示为一个包含 ACTION 表、GOTO 表和 CFG 符号表的结构体。

```
namespace lr_parsing_model
{
    struct LRParsingTable
    {
        std::unordered_set<cfg_model::symbol> all_symbols;
        std::unordered_set<std::string> all_states;
                                                    // all states in the par
sing table
        std::unordered_map<std::string, std::unordered_map<cfg_model::symb
ol, std::unordered_set<Action>>> action_table;    // conflict tolerant act
ion table
        std::unordered_map<std::string, std::unordered_map<cfg_model::symb
ol, std::unordered_set<std::string>>> goto_table; // goto table, conflict
tolerant
```

```
        bool add_action(const std::string &state, const cfg_model::symbol
&symbol, const Action &action);
        bool add_goto(const std::string &state, const cfg_model::symbol &s
ymbol, const std::string &next_state);
        bool check_cell_empty(const std::string &state, const cfg_model::s
ymbol &symbol) const;
        bool check_action_table_cell_empty(const std::string &state, const
 cfg_model::symbol &symbol) const;
        bool check_goto_table_cell_empty(const std::string &state, const c
fg_model::symbol &symbol) const;
        std::unordered_set<Action> get_actions(const std::string &state, c
onst cfg_model::symbol &symbol) const;
        std::unordered_set<std::string> get_gotos(const std::string &state
, const cfg_model::symbol &symbol) const;
        bool filling_check() const;
    };
};
```

3.2.3.6 YAML 文件格式

YAML 文件格式的设计主要是为了方便用户输入文法规则。同样，其中应该涵盖 CFG 文法规则的各个部分。一个典型的 YAML 文件示例如下（这也是 PPT 中给出的文法）：

请注意这个文法中没有 S 符号，因为 YAML 记录的是增广前的原始文法，S 符号要在增广过程中产生。

```yaml
# S' → E
# E → E + T | T
# T → T * F | F
# F → (E) | id
cfg:
  terminals:
  - "id"
  - "+"
  - "*"
  - "("
  - ")"
  non_terminals:
  - "E"
  - "T"
  - "F"
  initial_symbol: "E"
  production_rules:
  - lhs: "E"
    rhs:
    - "E"
    - "+"
    - "T"
  - lhs: "E"
    rhs:
```

```
    - "T"
  - lhs: "T"
    rhs:
    - "T"
    - "*"
    - "F"
  - lhs: "T"
    rhs:
    - "F"
  - lhs: "F"
    rhs:
    - "("
    - "E"
    - ")"
  - lhs: "F"
    rhs:
    - "id"
```

特别的，如果在 rhs 中提供一个空串，则表示该产生式可以为空。此时需要在 YAML 文件中提供一个空串的标识符（如`""`）。

```
# E → ε
- lhs: "E"
  rhs:
  - ""
```

3.3 实现细节

3.3.1 输入模块

输入模块只提供了一个单一的功能接口：给出一个文件的路径，返回一个 CFG 文法规则对象。其工作过程大致分为四个步骤：

1. 检查 YAML 文件的合法性：文件是否存在、所定义的各个格式字段是否存在等。

2. 解析 YAML 文件：将 YAML 文件解析为一个 CFG 文法规则对象。

3. 检查 CFG 文法规则的合法性：检查终结符号和非终结符号是否有重复、产生式是否符合 LR(0)文法的要求等。

4. 返回 CFG 文法规则对象。

这部分与核心算法联系较少，此处只展示接口定义，具体实现请见源码。

```
class YAML_CFG_Loader : public CFGLoader
{
```

```
public:
    // constructor & destructor
    YAML_CFG_Loader();
    ~YAML_CFG_Loader() override;
    cfg_model::CFG LoadCFG(const std::string &filename) override;
};
// helper functions
namespace YAML_CFG_Loader_Helper
{
    // check the validity of the YAML file
    void CheckYAMLFile(const std::string &filename);
    // parse the YAML file and load the CFG
    cfg_model::CFG ParseYAMLFile(const std::string &filename);
    // check the validity of the CFG
    void CheckCFG(const cfg_model::CFG &cfg);
}
```

3.3.2 项目集与规范族产生器

3.3.2.1 2.1 项目集产生器

项目产生器的主要目标是给出一个 CFG 文法相对应的 ItemSet。其自然语言过程描述如下：

1. 读取 CFG 并确定初始状态。

2. 增广 CFG 文法，生成一个新状态名，这个状态名应该和所有的状态名不冲突。

3. 增广 CFG 文法，添加新的起始产生式 S' → S。

4. 对 CFG 中的每个产生式，构建其所有文法项目。

5. 生成项目集 ItemSet。

下面给出一些重要的核心代码。在增广 CFG 时，由于不知道原有 CFG 占用了哪些符号，因此我们会选取开始符号，不断增加_expand 后缀，直到其与所有 CFG 符号均不重复。同时，为了完成 ParsingTable，还需要在 CFG 中增加一个结束符号（即 PPT 中的#），其取名的方式同样，但是添加的后缀名为_end。**因此在实验结果部分，假设原有初始状态为 E 且文法中没有状态以_expand 结尾，则 E_expand 为增广后新的初始符号，否则是 E_expand_expand…直到没有冲突。同理，结束符号为 E_end…（一个或若干个_end）**

```
        // note this function creates a new CFG, it does not modify the or
iginal CFG
        cfg_model::CFG expanded_cfg = cfg;
        // 1. read the initial symbol
```

```
        auto original_initial_symbol = cfg.start_symbol;
        // 2. create a new initial symbol
        std::string new_initial_symbol_name = generate_unique_symbol_name(
original_initial_symbol.name, cfg, "_expanded");
        cfg_model::symbol new_initial_symbol;
        new_initial_symbol.name = new_initial_symbol_name;
        new_initial_symbol.is_terminal = false;
        // 3. add the new initial symbol to the CFG
        expanded_cfg.non_terminals.insert(new_initial_symbol);
        expanded_cfg.start_symbol = new_initial_symbol;
        // 4. add a new production rule start_symbol -> original_initial_s
ymbol
        std::vector<cfg_model::symbol> new_production_rule;
        new_production_rule.push_back(original_initial_symbol);
        expanded_cfg.production_rules[new_initial_symbol].insert(new_produ
ction_rule);
        // 5. copy the rest of the original CFG
        // copy non-terminals and terminals
        for (const auto &terminal : cfg.terminals)
        {
            expanded_cfg.terminals.insert(terminal);
        }
        for (const auto &non_terminal : cfg.non_terminals)
        {
            expanded_cfg.non_terminals.insert(non_terminal);
        }
        // copy production rules
        for (const auto &rule : cfg.production_rules)
        {
            expanded_cfg.production_rules[rule.first] = rule.second;
        }
        // copy epsilon production symbols
        for (const auto &epsilon_symbol : cfg.epsilon_production_symbols)
        {
            expanded_cfg.epsilon_production_symbols.insert(epsilon_symbol)
;
        }
        // 6. add an end symbol to the CFG
        std::string end_symbol_name = generate_unique_symbol_name(original
_initial_symbol.name, cfg, "_end");
        cfg_model::symbol end_symbol;
        end_symbol.name = end_symbol_name;
        end_symbol.is_terminal = true;
        end_symbol.special_property = "END";
        // add the end symbol to the CFG
        expanded_cfg.terminals.insert(end_symbol);
        return expanded_cfg;
```

接下来的主要工作是对每个产生式产生所有的 Item 并加入 ItemSet 中，此处注意如果空产生式单独存放在一个映射表中，因此需要单独处理。此处仅给出处理一个非空产生式的实现。

```
std::unordered_set<std::shared_ptr<lr_parsing_model::Item>> itemset_genera
tor_helper::gen_non_epsilon_production_rule_items(const cfg_model::symbol
&left_side_symbol, const std::vector<cfg_model::symbol> &right_side_sequen
ce)
{
    try
    {
        // generate the item set for a given production rule(not epsilon)
        std::unordered_set<std::shared_ptr<lr_parsing_model::Item>> items;
        // generate all items for the given production rule
        for (size_t i = 0; i <= right_side_sequence.size(); ++i)
        {
            // create a new item
            std::shared_ptr<lr_parsing_model::Item> item = std::make_share
d<lr_parsing_model::Item>();
            item->left_side_symbol = left_side_symbol;
            item->sequence_already_parsed = std::vector<cfg_model::symbol>
(right_side_sequence.begin(), right_side_sequence.begin() + i);
            item->sequence_to_parse = std::vector<cfg_model::symbol>(right
_side_sequence.begin() + i, right_side_sequence.end());
            // insert the item into the set
            items.insert(item);
            std::string parsed_sequence_str = "";
            std::string to_parse_sequence_str = "";
            for (const auto &symbol : item->sequence_already_parsed)
            {
                parsed_sequence_str += symbol.name + " ";
            }
            for (const auto &symbol : item->sequence_to_parse)
            {
                to_parse_sequence_str += symbol.name + " ";
            }
            spdlog::debug("Generated non-
epsilon production rule item: {} -> {} · {}", left_side_symbol.name, parse
d_sequence_str, to_parse_sequence_str);
        }
        return items;
    }
    catch (const std::exception &e)
    {
        std::string error_msg = "Error during non-
epsilon production rule item generation: " + std::string(e.what());
        spdlog::error(error_msg);
        throw std::runtime_error(error_msg);
    }
}
```

3.3.2.2 规范族产生器

规范族产生器接收一个 ItemSet 对象，其主要功能如下：

1. 创建一个与 ItemSet 相对应的 NFA 对象，同时记录 ItemSet 与 NFA 之间的状态映射关系。

2. 调用 NFA-DFA 转换器，将 NFA 转换为 DFA，同时得到 NFA 和 DFA 之间的状态映射关系。

3. 计算得到 ItemSet 和 DFA 之间的状态映射关系。

4. 生成 LR(0)项目集规范族 ParsingTable 对象。

首先给出规范族产生器的接口定义：

```
class ItemSetToParsingTable
{
private:
    lr_parsing_model::ItemSet item_set;
    nfa_model::NFA nfa;
    lr_parsing_model::ItemSetNFAMapping item_set_nfa_mapping;
    dfa_model::ConflictTolerantDFA<std::string> dfa;
    lr_parsing_model::ItemSetDFAMapping item_set_dfa_mapping;
    lr_parsing_model::LRParsingTable parsing_table;
public:
    ItemSetToParsingTable(const lr_parsing_model::ItemSet &item_set);
    ~ItemSetToParsingTable() = default;
    // generate corresponding NFA for the item set
    lr_parsing_model::ItemSetNFAGenerationResult generate_nfa();
    // generate corresponding DFA for the item set
    lr_parsing_model::ItemSetDFAGenerationResult generate_dfa();
    lr_parsing_model::LRParsingTable build_parsing_table();
        // get nfa & mapping
    lr_parsing_model::ItemSetNFAGenerationResult get_nfa() const
    {
        return {nfa, item_set_nfa_mapping};
    }
    // get dfa & mapping
    lr_parsing_model::ItemSetDFAGenerationResult get_dfa() const
    {
        return {dfa, item_set_dfa_mapping};
    }
    // get parsing table
    lr_parsing_model::LRParsingTable get_parsing_table() const
    {
        return parsing_table;
    }
};
namespace itemset_to_parsing_table_helper
{
    // help generate a unique NFA character name for a given item
    std::string generate_nfa_character_name(const cfg_model::symbol &symbo
l);
    // help generate a unique NFA state name for a given item
```

```
    std::string generate_nfa_state_name(const lr_parsing_model::Item &item
);
}
```

可见其中的要点是：1. 完成 ItemSet-ItemNFA-ItemDFA 转换 2.记录并建立 ItemSet-ItemNFA-ItemDFA 中各个元素的保持关系。对于 NFA 和 DFA 转换由 NFA-DFA 转换器完成，因此要点 1 相对简单，下面给出如何通过各种映射关系从 ItemSet、ItemNFA 和 ItemDFA 中建立 ParsingTable：

1. 从 ItemSet 中获取 symbol 表，从 DFA 中获取 state 表
2. 考察 DFA 中的所有边（转移），这同时对应 Action 表中的 Shift 和 Goto 表中的下一个状态，所以要加以区分，同时还需要排除 ACC 状态。
3. 从 DFA 的所有接受状态中产生所有 Reduce 动作
4. 从 ItemSet 中找出 Accept 对应的 Item，在映射关系中找出其中对应的状态
5. 将 ParsingTable 中的空空格标记为空动作/空 Goto

对应的核心代码如下：

```
        // build the parsing table
        lr_parsing_model::LRParsingTable new_parsing_table;
        // 1. add all symbols from the ItemSet and all states from the DFA
 to the parsing table
        for (const auto &symbol : item_set.symbol_set)
        {
            new_parsing_table.all_symbols.insert(symbol);
        }
        for (const auto &state : dfa.states_set)
        {
            new_parsing_table.all_states.insert(state);
        }
        // 2. find all shift actions and goto states
        int added_shift_actions = 0;
        int added_goto_states = 0;
        for (const auto &state_transitions : dfa.transitions)
        {
            const std::string &state = state_transitions.first;
            const auto &transitions = state_transitions.second;
            // iterate over all transitions for the current state
            for (const auto &transition : transitions)
            {
                std::string input_char = transition.first;
                std::string next_state = transition.second;
                spdlog::debug("Processing transition: {} --{}-
-> {}", state, input_char, next_state);
                cfg_model::symbol corresponding_symbol = item_set_dfa_mapp
ing.dfa_character_to_item_set_symbol.at(input_char);
                // get all the items in the next state
                auto items_in_next_state = item_set_dfa_mapping.dfa_state_
to_item_set.at(next_state);
                spdlog::debug("Item number in next state for this transiti
on: {}", items_in_next_state.size());
```

```
                // iterate over all items in the next state
                for (const auto &item : items_in_next_state)
                {
                    // for terminal symbols, add a shift action as long as
 the item is not the end item
                    if (corresponding_symbol.is_terminal)
                    {
                        if (item == item_set.end_item)
                        {
                            // skip the end item and accepting states
                            continue;
                        }
                        lr_parsing_model::Action shift_action;
                        shift_action.action_type = "shift";
                        shift_action.target_state = next_state;
                        // add the action to the parsing table
                        bool added = new_parsing_table.add_action(state, c
orresponding_symbol, shift_action);
                        if (added) added_shift_actions++;
                    }
                    // for non-terminals, always add a goto state
                    else
                    {
                        // add a goto state
                        bool added = new_parsing_table.add_goto(state, cor
responding_symbol, next_state);
                        if (added) added_goto_states++;
                    }
                }
            }
        }
        spdlog::debug("Added {} shift actions and {} goto states to the pa
rsing table", added_shift_actions, added_goto_states);
        // 3. find all reduce actions
        // iterate over all accepting states in the DFA
        int added_reduce_actions = 0;
        for (const auto &accepting_state : dfa.accepting_states)
        {
            // get all items in the accepting state
            auto items_in_accepting_state = item_set_dfa_mapping.dfa_state
_to_item_set.at(accepting_state);
            // iterate over all items in the accepting state
            for (const auto &item : items_in_accepting_state)
            {
                // check if the item is an accepting state and not the end
 item
                if (item->sequence_to_parse.empty() && item != item_set.en
d_item)
                {
                    // create a reduce action
                    lr_parsing_model::Action reduce_action;
                    reduce_action.action_type = "reduce";
```

```
                    reduce_action.reduce_rule_lhs = item->left_side_symbol
;
                    reduce_action.reduce_rule_rhs = item->sequence_already
_parsed;
                    // add the action to the parsing table for all symbols
                    for (const auto &symbol : item_set.symbol_set)
                    {
                        // add the action to the parsing if the symbol is
a terminal
                        if (symbol.is_terminal)
                        {
                            // add the action to the parsing table
                            bool added = new_parsing_table.add_action(acce
pting_state, symbol, reduce_action);
                            if(added)
                            {
                                added_reduce_actions++;
                            }
                        }
                    }
                }
            }
        }
        spdlog::debug("Added {} reduce actions to the parsing table", adde
d_reduce_actions);
        // 4. find the accept action
        // get the end item from the item set and find the corresponding s
tate
        auto end_item = item_set.end_item;
        std::unordered_set<std::string> end_item_states;
        end_item_states = item_set_dfa_mapping.item_set_to_dfa_state.at(en
d_item);
        std::string end_item_state = "";
        // there should be only one end item state
        if (end_item_states.size() != 1)
        {
            std::string error_msg = "Error: End item state not found or mu
ltiple end item states found";
            spdlog::error(error_msg);
            throw std::runtime_error(error_msg);
        }
        else{
            end_item_state = *end_item_states.begin();
        }
        // get the end symbol
        cfg_model::symbol end_symbol;
        int end_symbol_count = 0;
        for (const auto &symbol: item_set.symbol_set)
        {
            if (symbol.special_property == "END")
            {
```

```
                spdlog::debug("Found end symbol: {}", std::string(symbol))
;
                end_symbol = symbol;
                end_symbol_count++;
            }
        }
        if (end_symbol_count != 1)
        {
            std::string error_msg = "Error: End symbol not found or multip
le end symbols found";
            spdlog::error(error_msg);
            throw std::runtime_error(error_msg);
        }
        // create an accept action
        lr_parsing_model::Action accept_action;
        accept_action.action_type = "accept";
        new_parsing_table.add_action(end_item_state, end_symbol, accept_ac
tion);
        spdlog::debug("Added accept action for state {} and symbol {}", en
d_item_state, std::string(end_symbol));
        // 5. fill all the empty cells in the parsing table with empty act
ions
        int patched_cells = 0;
        for (const auto &state : dfa.states_set)
        {
            for (const auto &symbol : item_set.symbol_set)
            {
                // check if the cell is empty
                if (new_parsing_table.check_cell_empty(state,symbol))
                {
                    // check if the symbol is a terminal or non-terminal
                    if (!symbol.is_terminal)
                    {
                        // fill in the goto state with an empty string
                        std::string next_state = "";
                        bool added = new_parsing_table.add_goto(state, sym
bol, next_state);
                        if (added) patched_cells++;
                    }
                    else
                    {
                        // fill in the action with an empty action
                        lr_parsing_model::Action empty_action;
                        empty_action.action_type = "empty";
                        bool added = new_parsing_table.add_action(state, s
ymbol, empty_action);
                        if (added) patched_cells++;
                    }
                }
            }
        }
```

```
        spdlog::debug("Patched {} empty cells in the parsing table", patch
ed_cells);
```

3.3.3 NFA-DFA 转换器

NFA-DFA 转换器的目前是将一个 NFA 转换为一个等价的 DFA。此处的 NFA 是一个通用的 NFA 模型，并不包含 LR(0)项目集的相关信息。其工作过程大致分为以下几个步骤：

1. 读取 NFA 的状态集合、字符集合、起始状态和接受状态集合。

2. 将 NFA 中的状态按照 ε -闭包进行分组，作为将来 DFA 状态的预备集合，同时进行重命名、判断是否为接受状态等。

3. 在 NFA 状态和 DFA 状态之间建立映射关系。

4. 根据 NFA 定义和其与 DFA 状态的映射关系，生成 DFA 的转移函数。

5. 根据闭包分组情况，确定 DFA 的起始和接受状态集合。

6. 构建并返回 DFA 对象。

其中最重要的确定闭包和生成转移函数。

确定闭包的经典算法描述如下：

- 初始化闭包集合为起始状态的 ε 闭包
- 对每个闭包集合中的闭包，寻找非 ε 转移所能达到的状态集和
- 计算可达状态集和的闭包并加入闭包集合
- 重复上述步骤，直到闭包集合不再扩大

其对应核心代码如下：

```
        spdlog::debug("Generating closure set:");
        std::unordered_set<NFAClosure> closure_set;
        int closure_set_increment = 1;
        // initialize the closure set with the closure of the start state
        NFAClosure start_state_closure = get_state_closure(nfa, nfa.start_
state);
        closure_set.insert(start_state_closure);
        spdlog::debug("Initial closure set: {}", start_state_closure.closu
re_name);
        // start growing the closure set
        int accepting_closure_count = 0;
        int initial_closure_count = 0;
        while (closure_set_increment > 0)
        {
            closure_set_increment = 0;
            std::unordered_set<NFAClosure> new_closures;
            // iterate through the closure set
            for (const auto &closure : closure_set)
```

```
            {
                std::unordered_map<std::string, std::unordered_set<std::st
ring>> transition_reachable_states;
                // iterate through the closure states
                for (const auto &state : closure.states)
                {
                    // check if the state has any non-epsilon transitions
                    if (nfa.non_epsilon_transitions.find(state) == nfa.non
_epsilon_transitions.end())
                    {
                        spdlog::debug("No non-
epsilon transitions for state {}", state);
                        continue;
                    }
                    // iterate through the non-epsilon transitions
                    for (const auto &transition : nfa.non_epsilon_transiti
ons.at(state))
                    {
                        // for now we don't build closure transitions but
just find the reachable states
                        std::string input_string = transition.first;
                        std::string next_state = transition.second;
                        // add the next state to the corresponding input s
tring in transition_reachable_states
                        transition_reachable_states[input_string].insert(n
ext_state);
                    }
                }
                // iterate through the transition reachable states
                for (const auto &reachable_states : transition_reachable_s
tates)
                {
                    std::string input_string = reachable_states.first;
                    std::unordered_set<std::string> reachable_state_set =
reachable_states.second;
                    spdlog::debug("Closure {} has reachable states for inp
ut {}: {}", closure.closure_name, input_string, fmt::join(reachable_state_
set, ", "));
                    // get the closure for the reachable states
                    NFAClosure new_closure = get_state_set_closure(nfa, re
achable_state_set);
                    // check if the new closure is already in the closure
set
                    if (closure_set.find(new_closure) == closure_set.end()
)
                    {
                        // add the new closure to the new closures set
                        new_closures.insert(new_closure);
                        spdlog::debug("New closure found: {}", new_closure
.closure_name);
                        // increment the closure set increment
                        closure_set_increment++;
```

```
                        if (new_closure.has_accepting_state)
                        {
                            spdlog::debug("This closure has an accepting s
tate");
                        }
                        if (new_closure.has_initial_state)
                        {
                            spdlog::debug("This closure has the initial st
ate");
                        }
                    }
                    else
                    {
                        spdlog::debug("Omitting one closure as it is alrea
dy in the closure set");
                    }
                }
            }
            // add the new closures to the closure set
            for (const auto &new_closure : new_closures)
            {
                closure_set.insert(new_closure);
                if (new_closure.has_accepting_state)
                {
                    accepting_closure_count++;
                }
                if (new_closure.has_initial_state)
                {
                    initial_closure_count++;
                    if (initial_closure_count > 1)
                    {
                        std::string error_message = "The closure set has m
ore than one initial state";
                        spdlog::error(error_message);
                        throw std::runtime_error(error_message);
                    }
                }
            }
            spdlog::debug("The closure set has grown by {} closures to {}
closures", closure_set_increment, closure_set.size());
        }
        // check if the closure set is empty
        if (closure_set.empty())
        {
            std::string error_message = "The closure set is empty";
            spdlog::error(error_message);
            throw std::runtime_error(error_message);
        }
        spdlog::debug("Closure set generation completed with {} closures,
{} of them are accepting closures, and {} of them are initial closures", c
losure_set.size(), accepting_closure_count, initial_closure_count);
        // return the closure set
```

```
        return closure_set;
```

计算转移的方法如下：（事实上，转移可以和闭包一同计算，此处为了解耦和代码可读性分开）

- 遍历 NFA 中的每一个状态，如果其是可达状态则一定在某个闭包中。
- 找到其闭包对应的 DFA 状态。
- 找到其在每个符号下能转移到的所有状态
- 找到每个符号对应可转移状态集所对应的闭包，这个闭包应该恰好对应一个 DFA 状态。
- 在 NFA 状态对应 DFA 状态和可转移状态集对应的 DFA 状态集间建立转移。

```
spdlog::debug("Generating DFA transitions:");
        std::unordered_map<std::string, std::unordered_map<std::string, st
d::string>> dfa_transitions;
        int generated_transitions_count = 0;
        // iterate through the DFA states
        for (const auto &dfa_state : dfa.states_set)
        {
            // iterate through the symbols in the DFA character set
            for (const auto &symbol : dfa.character_set)
            {
                // get the NFA states for the DFA state
                std::unordered_set<std::string> nfa_states = state_mapping
.dfa_to_nfa_mapping.at(dfa_state);
                // mark the nfa states that have a transition for this sym
bol
                std::unordered_set<std::string> nfa_states_has_transition;
                // the nfa states that are reachable from the current nfa
states at this symbol
                std::unordered_set<std::string> reachable_nfa_states;
                // iterate through the NFA states
                for (const auto &nfa_state : nfa_states)
                {
                    // check if the NFA state has a transition for the sym
bol
                    bool has_transition = false;
                    if (nfa.non_epsilon_transitions.find(nfa_state) != nfa
.non_epsilon_transitions.end())
                    {
                        if (nfa.non_epsilon_transitions.at(nfa_state).find
(symbol) != nfa.non_epsilon_transitions.at(nfa_state).end())
                        {
                            has_transition = true;
                        }
                    }
                    // if the NFA state has a transition for the symbol
                    if (has_transition)
                    {
                        // add it to the NFA states that have a transition
 for this symbol
                        nfa_states_has_transition.insert(nfa_state);
```

```
                        // get the next state for the symbol
                        std::string next_state = nfa.non_epsilon_transitio
ns.at(nfa_state).at(symbol);
                        // add the next state to the reachable NFA states
                        reachable_nfa_states.insert(next_state);
                        spdlog::debug("DFA state {} has NFA state {} with
transition for symbol {} to NFA state {}", dfa_state, nfa_state, symbol, n
ext_state);
                    }

                }
                // now we have the reachable NFA states for the symbol, th
is set should precisely correspond to a DFA state
                // check if the reachable NFA states are empty
                if (reachable_nfa_states.empty())
                {
                    spdlog::debug("DFA state {} has no reachable NFA state
s for symbol {}", dfa_state, symbol);
                    continue;
                }
                // get the closure for the reachable NFA states
                NFAClosure reachable_closure = get_state_set_closure(nfa,
reachable_nfa_states);
                // check if the reachable closure is already in the DFA st
ates
                if (dfa.states_set.find(reachable_closure.closure_name) !=
 dfa.states_set.end())
                {
                    // add the transition to the DFA transitions
                    dfa_transitions[dfa_state].insert({symbol, reachable_c
losure.closure_name});
                    spdlog::debug("DFA transition: {} --{}-
-> {}", dfa_state, symbol, reachable_closure.closure_name);
                    // increment the generated transitions count
                    generated_transitions_count++;
                }
                else
                {
                    // if the reachable closure is not in the DFA states,
we need to add it
                    std::string error_message = "DFA state " + dfa_state +
 " has a transition to a non-
existing DFA state " + reachable_closure.closure_name;
                    spdlog::error(error_message);
                    throw std::runtime_error(error_message);
                }
            }
        }
        spdlog::debug("Generated {} DFA transitions", generated_transition
s_count);
        return dfa_transitions;
```

3.3.4 可视化部分

此部分和核心算法关系不大。其最重要的功能是检查 ParsingTable 中的每个单元格是否存在多个动作，以此判断文法是否存在冲突（会将冲突标红显示）。其余 NFA 和 DFA 的可视化使用第三方库 Graphviz 完成。

声明：因为可视化部分和核心算法没有明显联系，Graphviz 绘图 API 调用方式由 Gemini Copliot 实现。实验的其余部分算法及模型均为本人独立完成。

4 实验内容（选做）

选做任务即完成实验三和实验一与实验二的串联。实验一与实验二的串联在实验二中已经完成，即对词法类别使用 DFA 判断。得益于统一使用 YAML 格式，实验的串联实际上相当容易。

注意到上文中 CFG 定义的每个非终结符允许使用 string 表示且不作大小写要求。二实验二中的每个词法类别也使用 string 表示，因此只要将其选相同名称即可完成连接。

下面给出一个示例：（同样，S 作为要增广的符号不在配置中出现）

```yaml
# S' -> X
# X -> (X)
# X -> ()
cfg:
  terminals:
  - "LPA"
  - "RPA"
  non_terminals:
  - "X"
  initial_symbol: "X"
  production_rules:
  - lhs: "X"
    rhs:
    - "LPA"
    - "X"
    - "RPA"
  - lhs: "X"
    rhs:
    - "LPA"
    - "RPA"
token_types:
- name: "LPA"
  priority: 11
  has_content: false
- name: "RPA"
  priority: 12
  has_content: false
dfas:
- name: "LPA"
```

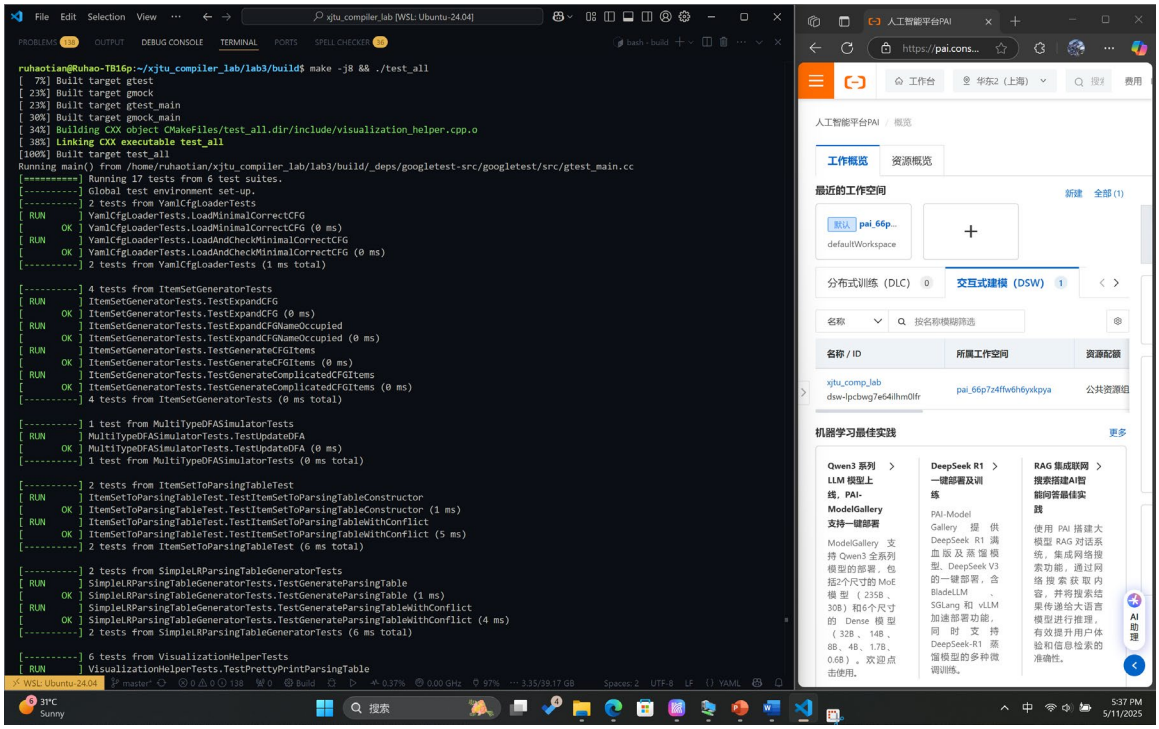
```
  character_set: "\\("
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: "("
- name: "RPA"
  character_set: "\\)"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: ")"
regexps:
- regexp: "^\\\\\\\($"
  token_type: "LPA"
- regexp: "^\\\\\\\)$"
  token_type: "RPA"
```

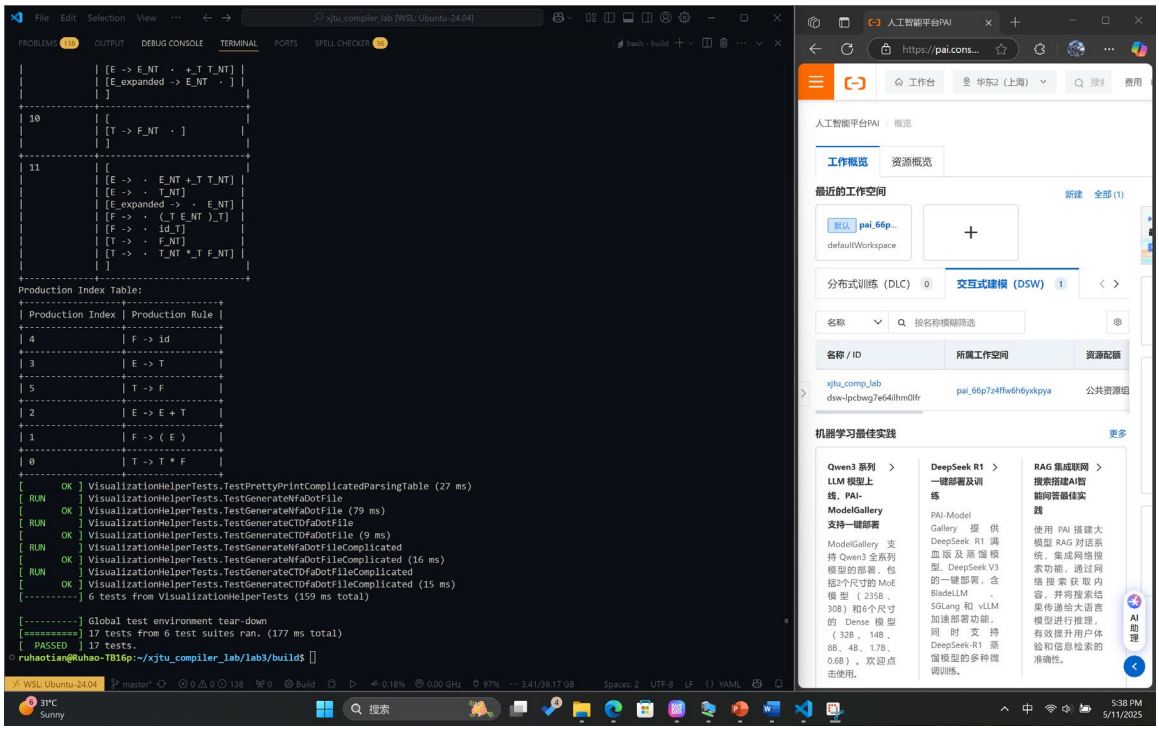
并且得益于 YAML 的字典性质，不同模块、不同后端的配置使用一个文件即可统一表示。

5 实验结果

在实验设计架构图中，本来准备实现一个简单的 CLI，但考虑到本次实验只是语法分析器的一部分，故采用单元测试展示结果。在 TDD 开发中共设计了 17 个单元测试。

本章节的测试结果表示和课件的符号表示有所不同，其具体差别在上文各章节中已经使用红字标出。





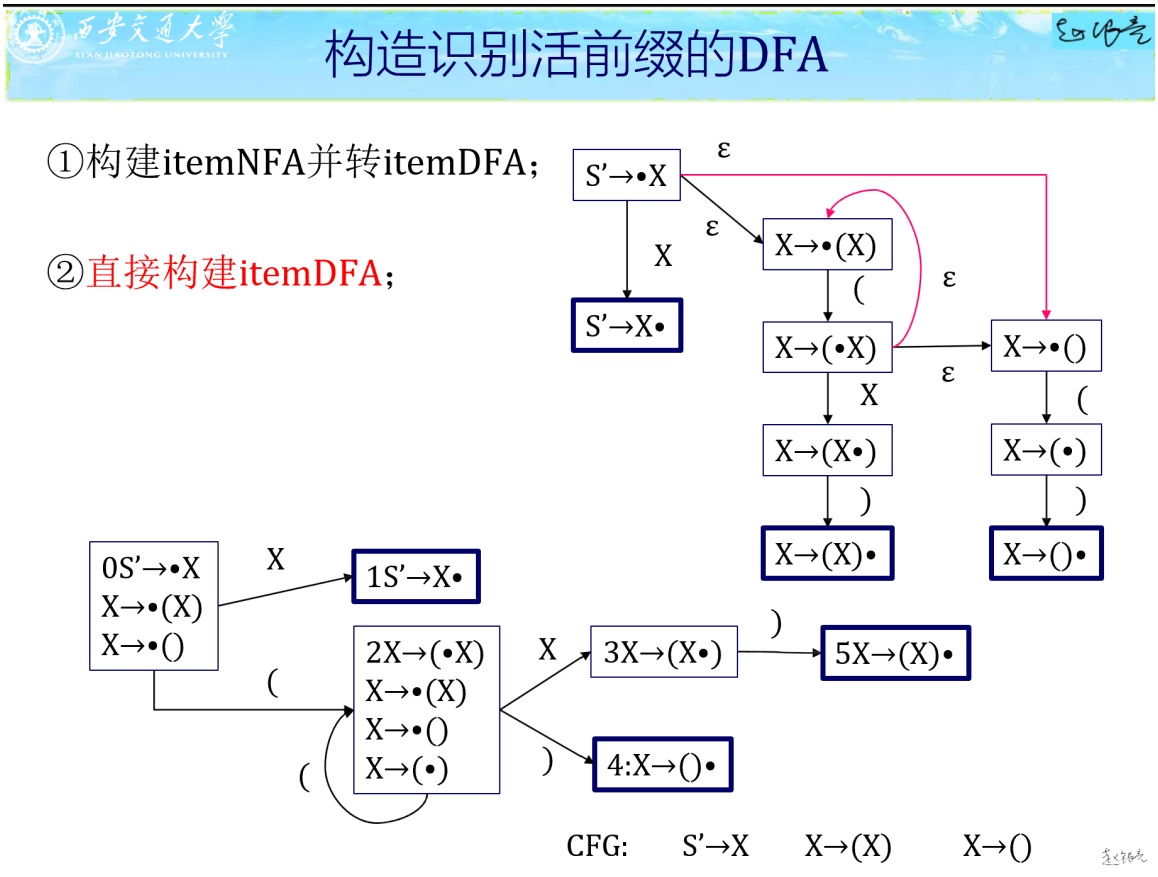
如图，这些单元测试均可通过。下面仅展示其中直接和实验要求相关的部分，其余测试请见源码附件。

5.1 项目集产生

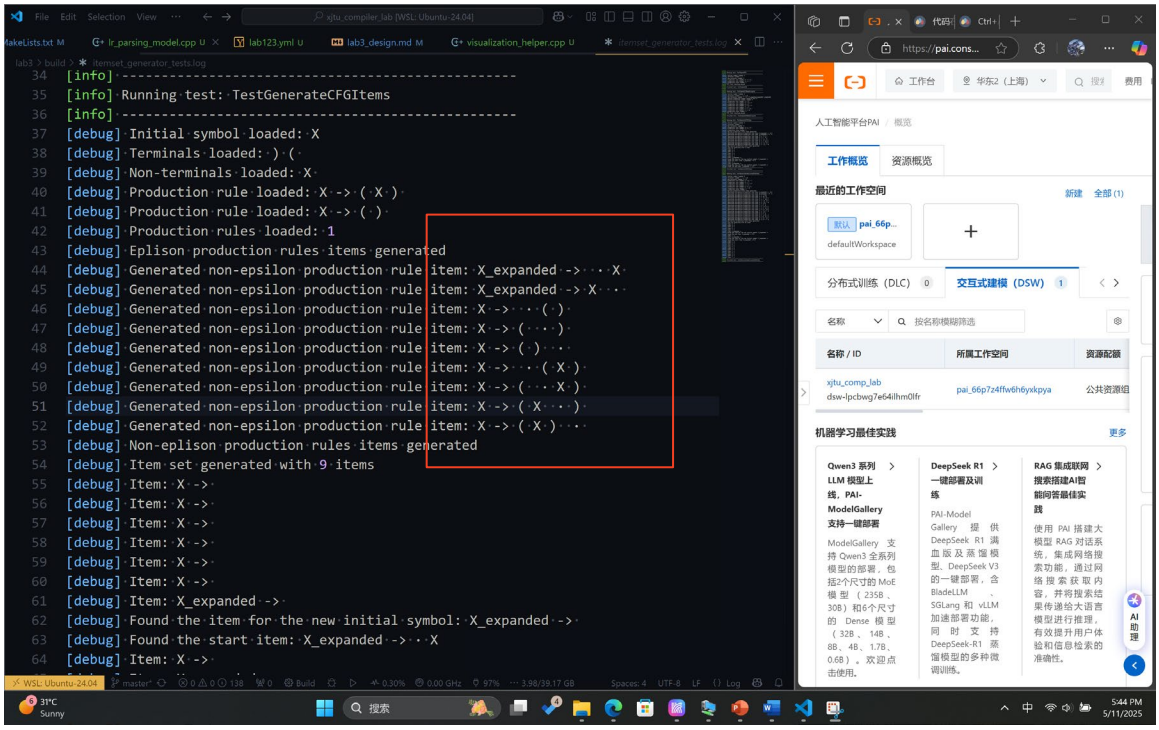
典型测试用例：

```
TEST_F(ItemSetGeneratorTests, TestGenerateCFGItems)
{
    // load the CFG from the YAML file
    std::string filename = "test/data/itemset_generator/minimal_correct_cfg.yml";
    cfg_model::CFG cfg = YAML_CFG_Loader_Helper::ParseYAMLFile(filename);

    ItemSetGenerator item_set_generator;
    lr_parsing_model::ItemSet item_set;
    ASSERT_NO_THROW(item_set = item_set_generator.generate_item_set(cfg));

    ASSERT_EQ(item_set.items.size(), 9);// 7 items + 2 initial and final items
    ASSERT_EQ(item_set.symbol_set.size(),(cfg.terminals.size() + cfg.non_terminals.size() + 2));
}
```

其使用的是第 8 章课件中的 CFG：



测试成功通过，且输出 log 如下：



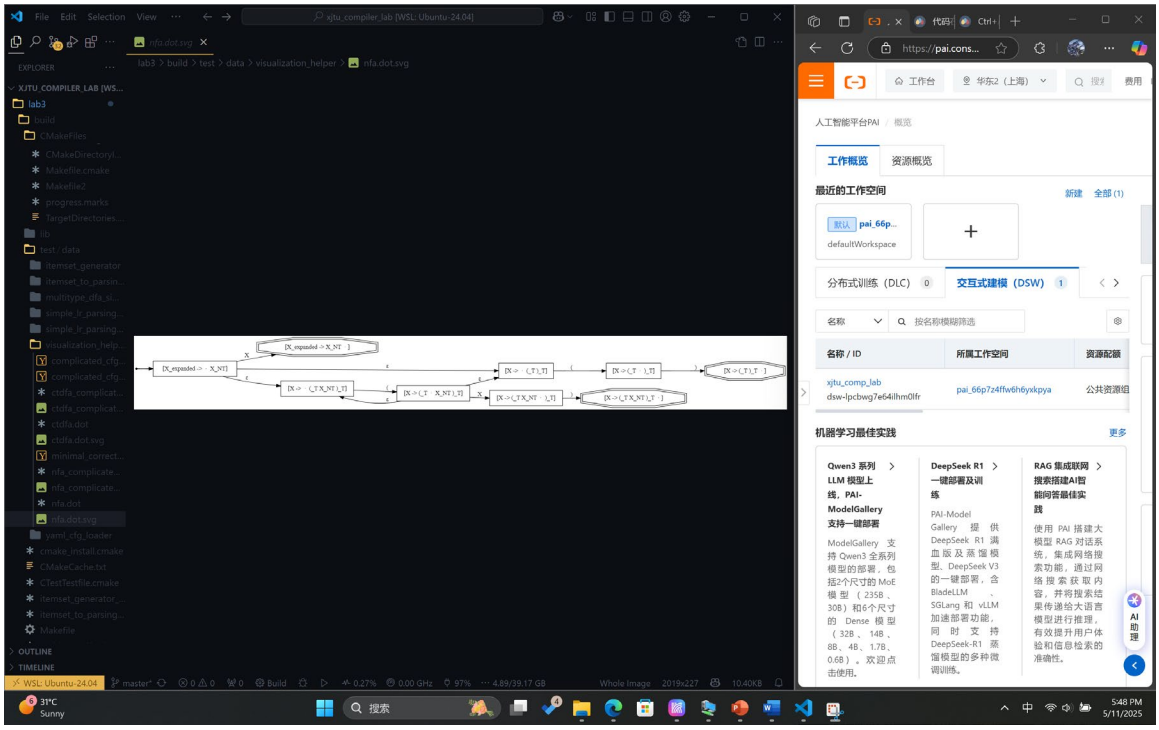
可见其中每个 Item 都被正确产生了。

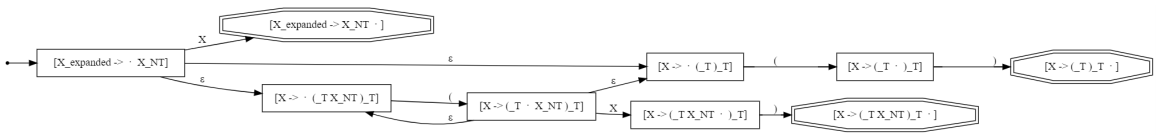
5.2 NFA 产生

典型测试用例：

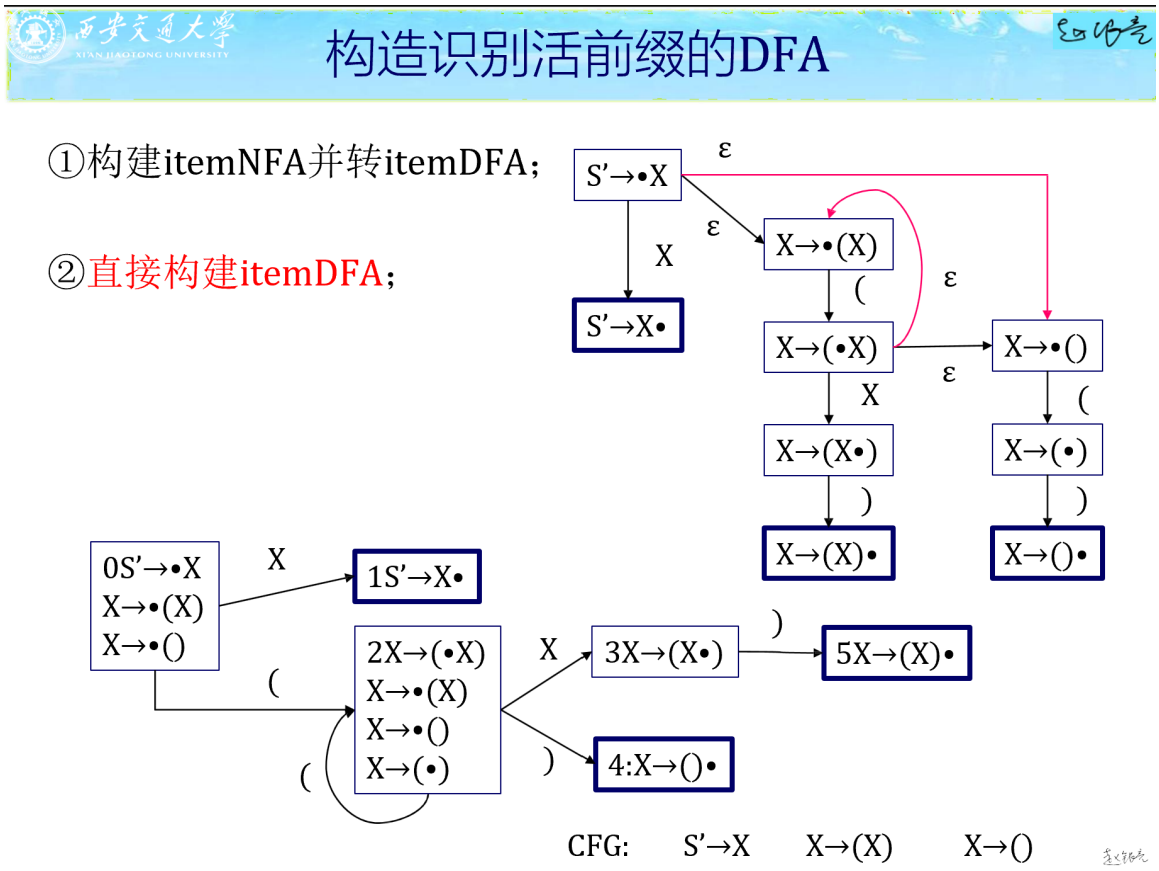
```
TEST_F(VisualizationHelperTests, TestGenerateNfaDotFile)
{
    // Load the CFG from the YAML file
    std::string cfg_file_path = "test/data/visualization_helper/minimal_correct_cfg.yml";
    cfg_model::CFG cfg = load_test_cfg(cfg_file_path);
    // Create an instance of SimpleLRParsingTableGenerator
    SimpleLRParsingTableGenerator generator;
    // Generate the parsing table
    lr_parsing_model::ItemSetNFAGenerationResult result = generator.generate_item_set_nfa(cfg);
    nfa_model::NFA nfa = result.nfa;
    // Generate the NFA dot file
    std::string dot_file_path = "test/data/visualization_helper/nfa.dot";
    ASSERT_NO_THROW(visualization_helper::generate_nfa_dot_file(nfa, dot_file_path, true)) << "generate_nfa_dot_file threw an exception";
}
```

同样使用上文的 CFG，输出并渲染的 DOT SVG 如下：





（以上是高清 SVG 可供放大查看）



对比可见其 NFA 是完全一致的。

5.3 带有冲突的 DFA 产生

```
TEST_F(VisualizationHelperTests, TestGenerateCTDfaDotFileComplicated)
{
    // Load the CFG from the YAML file
    std::string cfg_file_path = "test/data/visualization_helper/complicated_cfg_1_with_conflict.yml";
    cfg_model::CFG cfg = load_test_cfg(cfg_file_path);
    // Create an instance of SimpleLRParsingTableGenerator
    SimpleLRParsingTableGenerator generator;
    // Generate the parsing table
    lr_parsing_model::ItemSetDFAGenerationResult result = generator.generate_item_set_dfa(cfg);
    dfa_model::ConflictTolerantDFA<std::string> dfa = result.dfa;
    // Generate the CTDFA dot file
    std::string dot_file_path = "test/data/visualization_helper/ctdfa_complicated.dot";
    ASSERT_NO_THROW(visualization_helper::generate_ctdfa_dot_file(dfa, dot_file_path, true)) << "generate_ctdfa_dot_file threw an exception";
}
```

使用的是实验要求中的语法：

输入

S' → E

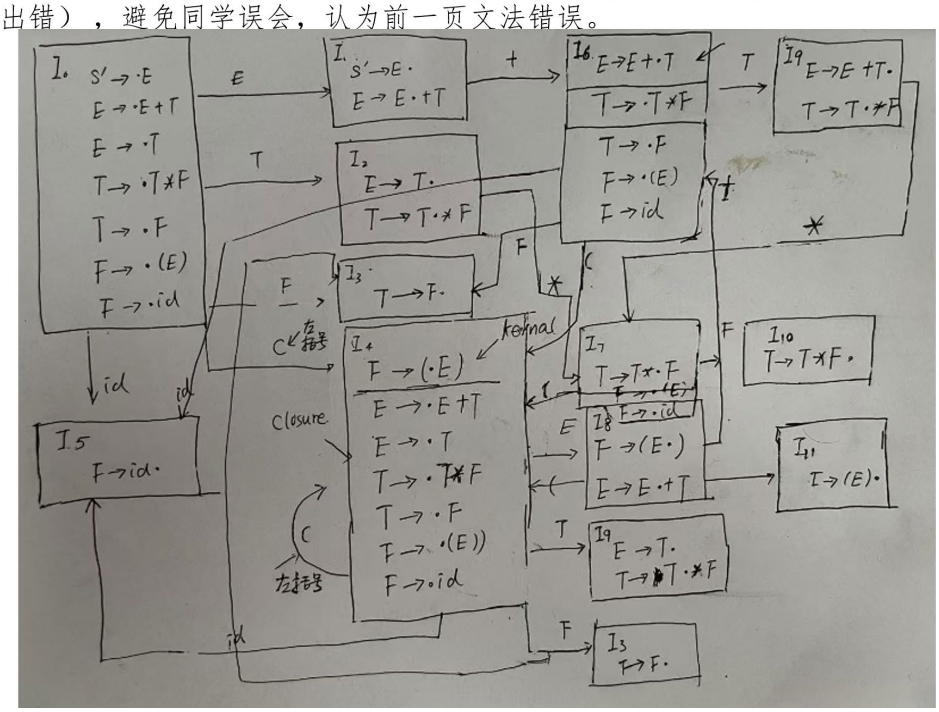
E → E + T | T

T → T * F | F

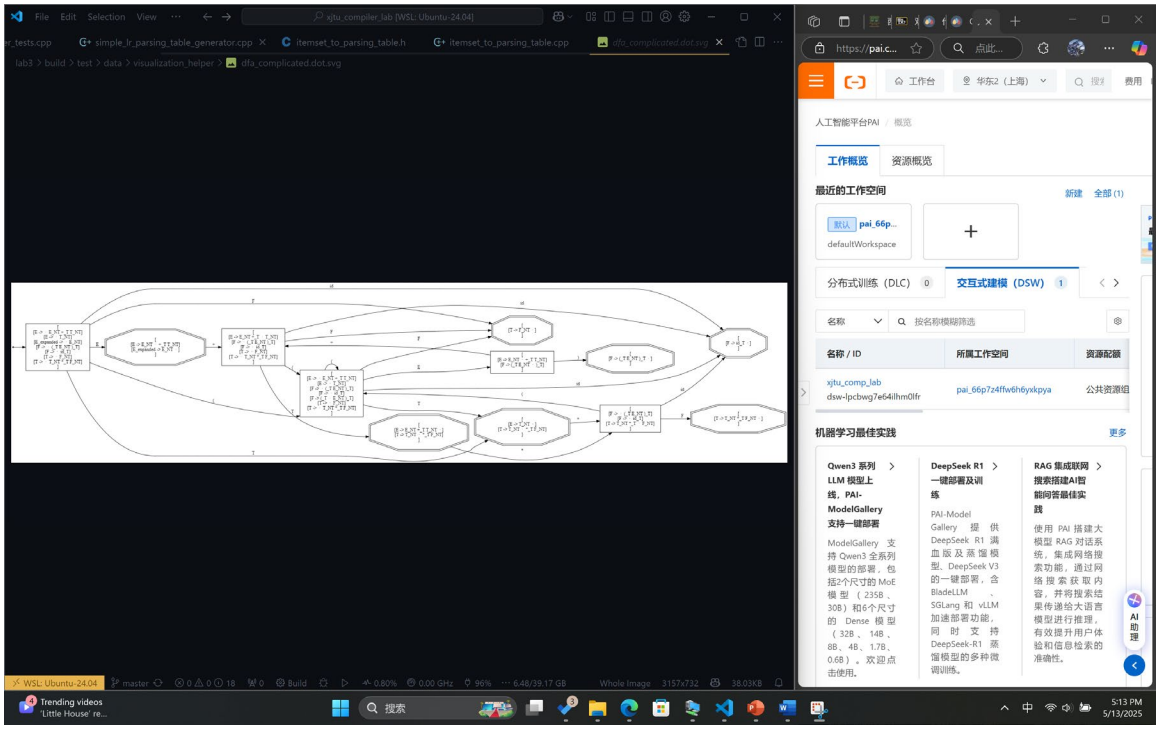
F → (E) | id

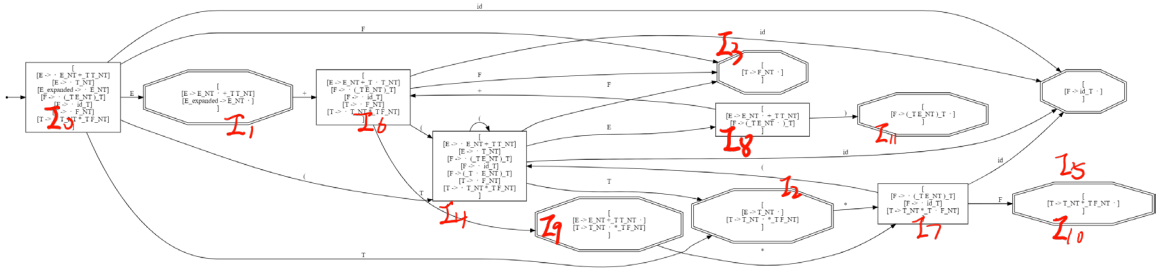
一、实验三回顾

- 重画了一下，前一页的文法和示例文法有点像，本页根据示例文法画出（手绘容易出错），避免同学误会，认为前一页文法错误。



模拟结果：





该测试用例同时展示了闭包和 DFA，其中内核项及起始项加上每个 DFA 状态（闭包）内・不在产生式右侧最左边的项。

5.4 ParsingTable 及冲突检测

典型测试语句如下：

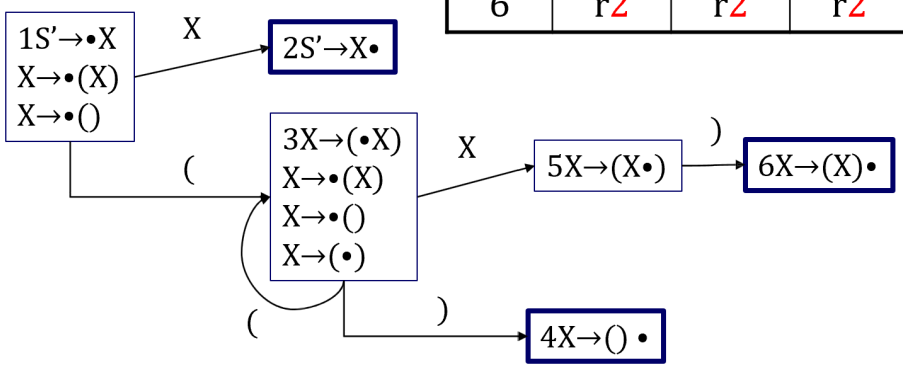
```
// Test pretty print complicated parsing table with conflicts
TEST_F(VisualizationHelperTests, TestPrettyPrintComplicatedParsingTable)
{
    // Load the CFG from the YAML file
    std::string cfg_file_path = "test/data/visualization_helper/complicated_cfg_1_with_conflict.yml";
    cfg_model::CFG cfg = load_test_cfg(cfg_file_path);
    // Create an instance of SimpleLRParsingTableGenerator
    SimpleLRParsingTableGenerator generator;
    // Generate the parsing table
    lr_parsing_model::LRParsingTable parsing_table = generator.generate_parsing_table(cfg);
```

```
    ASSERT_NO_THROW(parsing_table.filling_check()) << "Parsing table filli
ng check failed";
    // Pretty print the parsing table
    ASSERT_NO_THROW(visualization_helper::pretty_print_parsing_table(parsi
ng_table)) << "pretty_print_parsing_table threw an exception";
}
```

下面分别展示前文两个测试例子的 ParsingTable

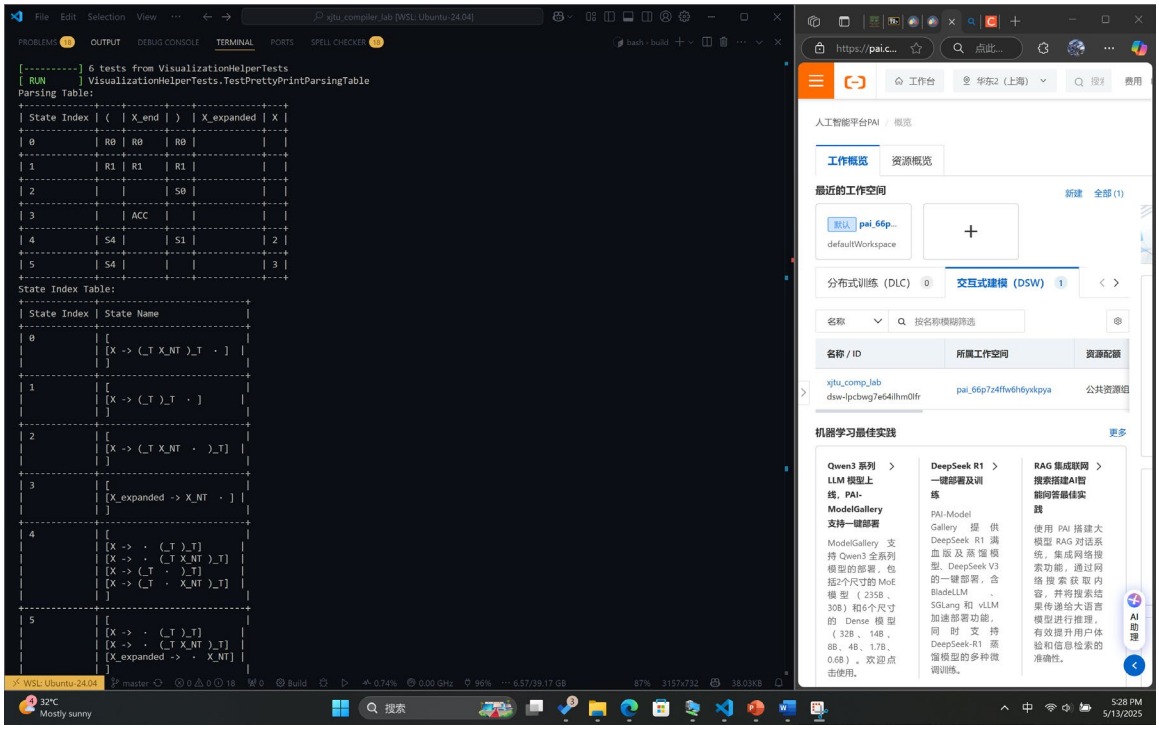
- 给定CFG；
- 先求出DFA；
- 再填ACTION表；
- 填GOTO表；
- 完成。

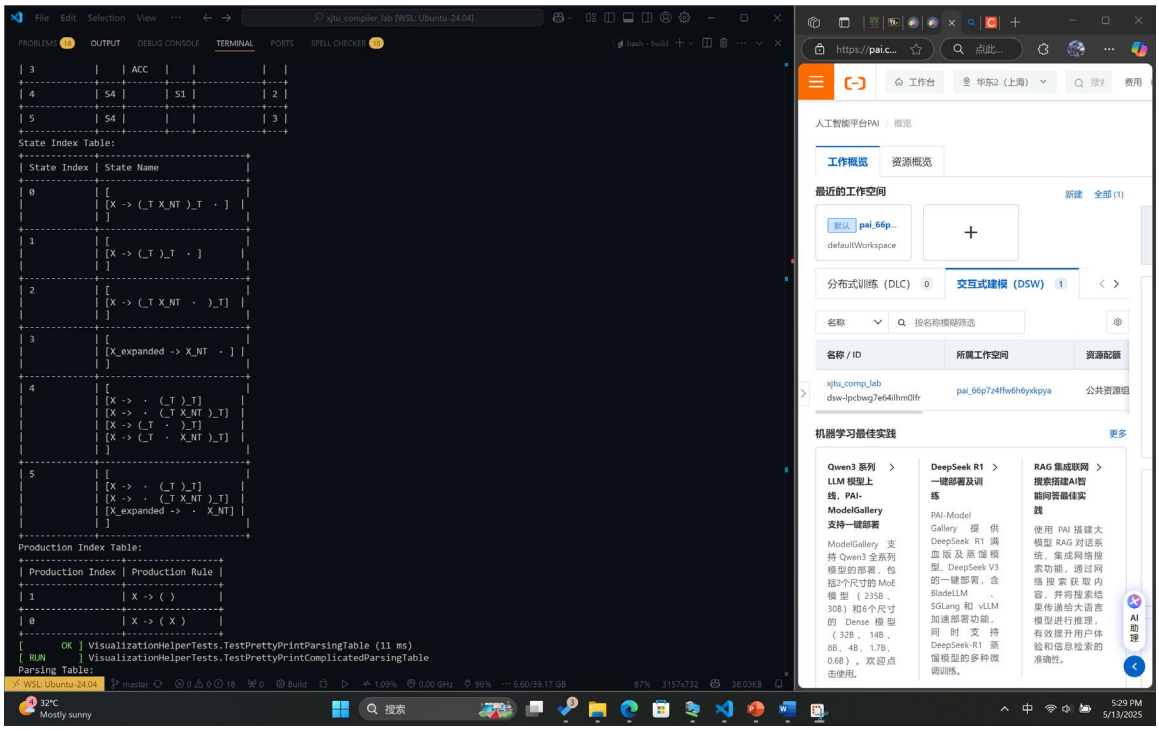
	ACTION			GOTO
状态	(	)	#	X
1	s3			2
2			acc	
3	s3	s4		5
4	r3	r3	r3	
5		s6		
6	r2	r2	r2	



1S→X
2X→(X)
3X→()

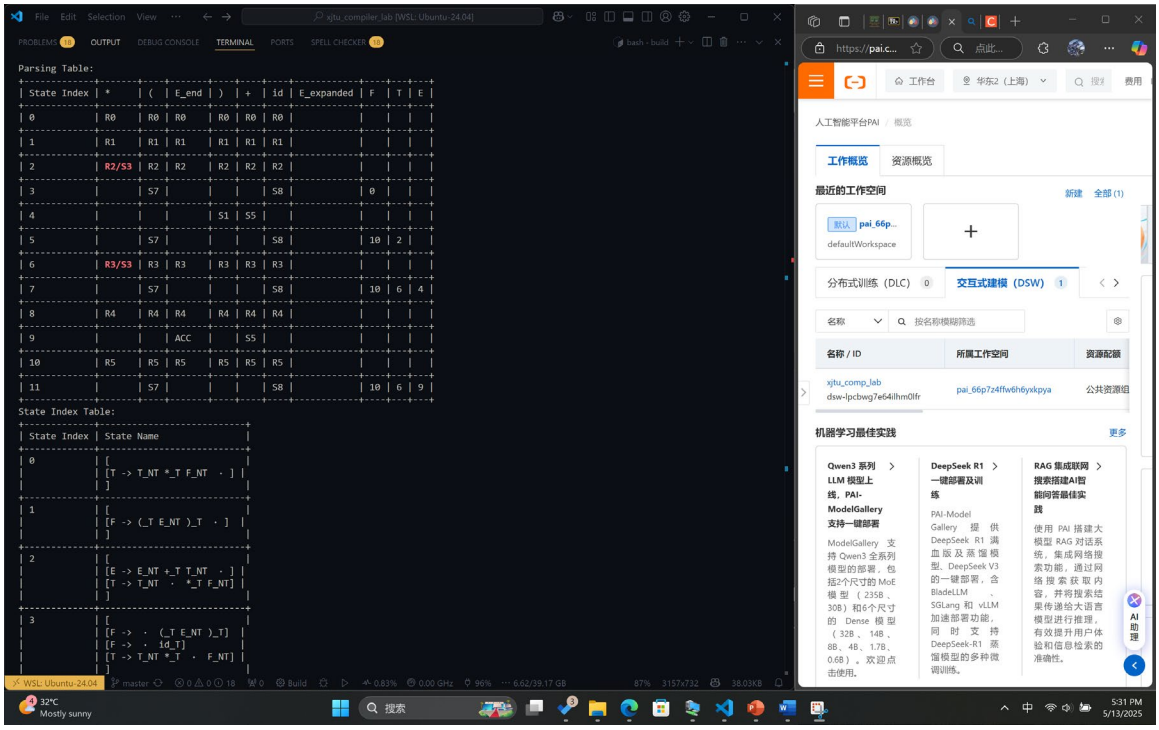
对应的 ParsingTable 如下：

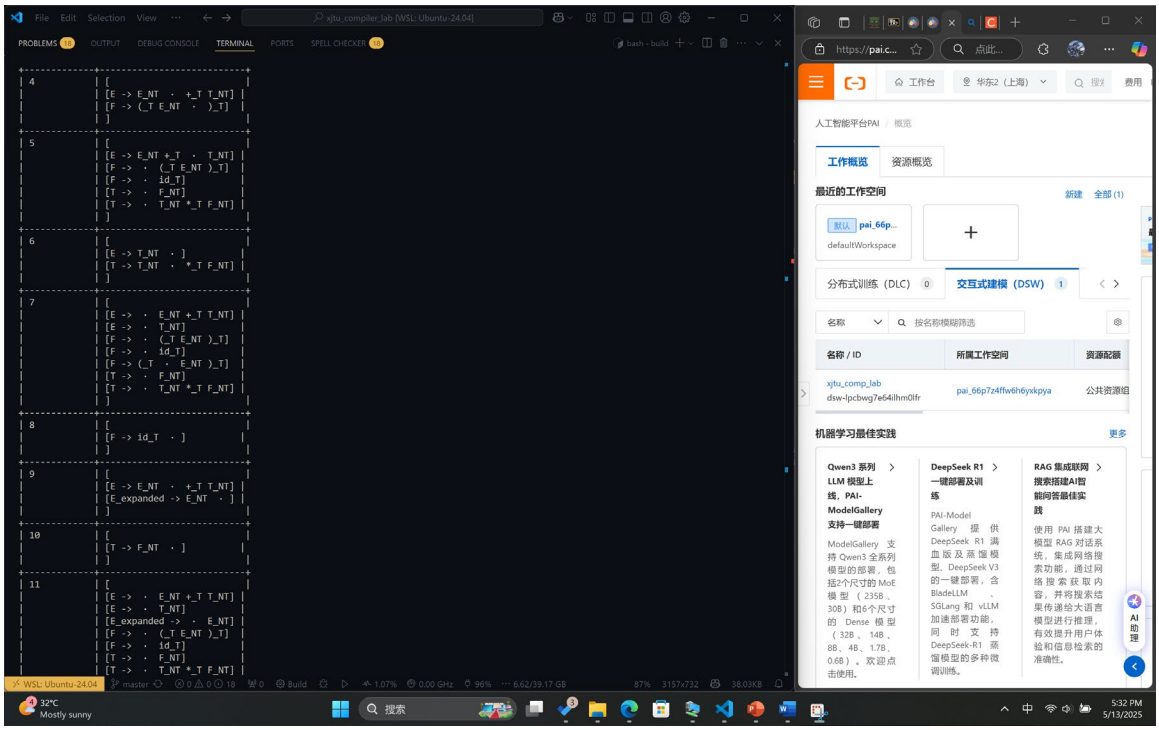


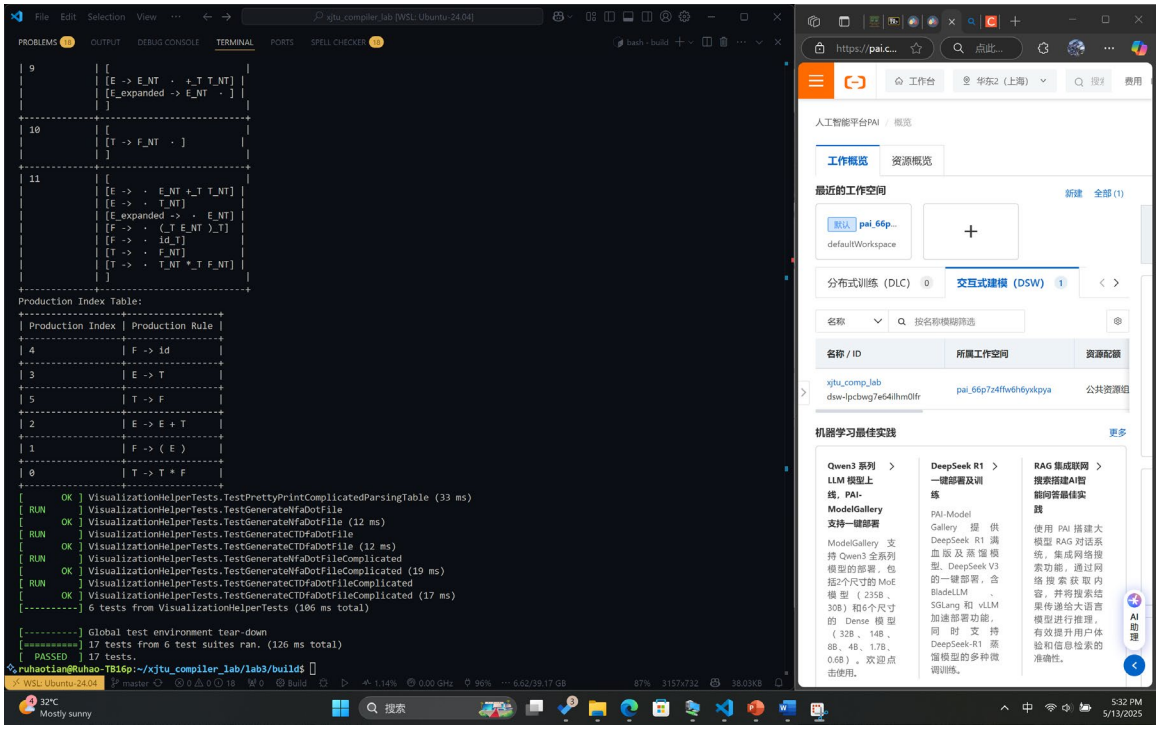


此处 Production Rule 只列出了会用到的 Production Rule，故 ACC 状态对应的产生式没有列出（收到 ACC Action 就直接接受了）

对带有冲突的复杂 CFG（第二个例子），ParsingTable 中用红色标出：


File Edit Selection View
xjtu_compiler_lab [WSL: Ubuntu-24.04]
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 10
bash - build
Parsing Table:
State Index | * | (| E_end |) | + | id | E_expanded | F | T | E
0 | R0 | R0 | R0 | R0 | R0 | R0 | | | | |
1 | R1 | R1 | R1 | R1 | R1 | R1 | | | | |
2 | R2/S3 | R2 | R2 | R2 | R2 | R2 | | | | |
3 | | S7 | | | | S8 | | 0 | | |
4 | | | | S1 | S5 | | | | | |
5 | | S7 | | | | S8 | | 10 | 2 | |
6 | R3/S3 | R3 | R3 | R3 | R3 | R3 | | | | |
7 | | S7 | | | | S8 | | 10 | 6 | 4 |
8 | R4 | R4 | R4 | R4 | R4 | R4 | | | | |
9 | | | ACC | | S5 | | | | | |
10 | R5 | R5 | R5 | R5 | R5 | R5 | | | | |
11 | | S7 | | | | S8 | | 10 | 6 | 9 |
State Index Table:
State Index | State Name
0 | [[T -> T_NT *_T F_NT ·]]
1 | [[F -> (_T E_NT)_T ·]]
2 | [[E -> E_NT +_T T_NT ·] [T -> T_NT · *_T F_NT]]
3 | [[F -> · (_T E_NT)_T] [F -> · id_T] [T -> T_NT *_T · F_NT]]
WSL: Ubuntu-24.04 master Build 0.83% 0.00 GHz 96% 6.62/39.17 GB 87% 3157x732 38.03KB
https://pai.c... 点此...
工作台 华东2（上海） 搜索 费用
人工智能平台PAI / 概览
工作概览 资源概览
最近的工作空间 新建 全部 (1)
默认 pai_66p... defaultWorkspace
分布式训练（DLC） 0 交互式建模（DSW） 1
名称 按名称模糊筛选
名称 / ID 所属工作空间 资源配额
xjtu_comp_lab dsw-lpcbwg7e64ilhm0lfr pai_66p7z4ffw6h6yxkpya 公共资源组
机器学习最佳实践 更多
Qwen3 系列 LLM 模型上线，PAI-ModelGallery 支持一键部署
ModelGallery 支持 Qwen3 全系列模型的部署，包括2个尺寸的 MoE 模型（235B、30B）和6个尺寸的 Dense 模型（32B、14B、8B、4B、1.7B、0.6B）。欢迎点击使用。
DeepSeek R1 一键部署及训练
PAI-ModelGallery 提供 DeepSeek-R1 满血版及蒸馏模型。DeepSeek-V3 的一键部署，含 BladeLLM、SGLang 和 vLLM 加速部署功能，同时支持 DeepSeek-R1 蒸馏模型的多种微调训练。
RAG 集成联网搜索搭建AI智能问答最佳实践
使用 PAI 搭建大模型 RAG 对话系统，集成网络搜索功能，通过网络搜索获取内容，并将搜索结果传递给大语言模型进行推理，有效提升用户体验和信息检索的准确性。
AI 助理
32°C Mostly sunny
搜索
5:31 PM 5/13/2025


File Edit Selection View
xjtu_compiler_lab [WSL: Ubuntu-24.04]
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 10
bash - build
4 | [E -> E_NT · +_T T_NT] [F -> (_T E_NT ·)_T]
5 | [E -> E_NT +_T · T_NT] [F -> · (_T E_NT)_T] [F -> · id_T] [T -> · F_NT] [T -> · T_NT *_T F_NT]
6 | [E -> T_NT ·] [T -> T_NT · *_T F_NT]
7 | [E -> · E_NT +_T T_NT] [E -> · T_NT] [F -> · (_T E_NT)_T] [F -> · id_T] [F -> (_T · E_NT)_T] [T -> · F_NT] [T -> · T_NT *_T F_NT]
8 | [F -> id_T ·]
9 | [E -> E_NT · +_T T_NT] [E_expanded -> E_NT ·]
10 | [T -> F_NT ·]
11 | [E -> · E_NT +_T T_NT] [E -> · T_NT] [E_expanded -> · E_NT] [F -> · (_T E_NT)_T] [F -> · id_T] [T -> · F_NT] [T -> · T_NT *_T F_NT]
WSL: Ubuntu-24.04 master Build 1.07% 0.00 GHz 96% 6.62/39.17 GB 87% 3157x732 38.03KB
https://pai.c... 点此...
工作台 华东2（上海） 搜索 费用
人工智能平台PAI / 概览
工作概览 资源概览
最近的工作空间 新建 全部 (1)
默认 pai_66p... defaultWorkspace
分布式训练（DLC） 0 交互式建模（DSW） 1
名称 按名称模糊筛选
名称 / ID 所属工作空间 资源配额
xjtu_comp_lab dsw-lpcbwg7e64ilhm0lfr pai_66p7z4ffw6h6yxkpya 公共资源组
机器学习最佳实践 更多
Qwen3 系列 LLM 模型上线，PAI-ModelGallery 支持一键部署
ModelGallery 支持 Qwen3 全系列模型的部署，包括2个尺寸的 MoE 模型（235B、30B）和6个尺寸的 Dense 模型（32B、14B、8B、4B、1.7B、0.6B）。欢迎点击使用。
DeepSeek R1 一键部署及训练
PAI-ModelGallery 提供 DeepSeek-R1 满血版及蒸馏模型。DeepSeek-V3 的一键部署，含 BladeLLM、SGLang 和 vLLM 加速部署功能，同时支持 DeepSeek-R1 蒸馏模型的多种微调训练。
RAG 集成联网搜索搭建AI智能问答最佳实践
使用 PAI 搭建大模型 RAG 对话系统，集成网络搜索功能，通过网络搜索获取内容，并将搜索结果传递给大语言模型进行推理，有效提升用户体验和信息检索的准确性。
AI 助理
32°C Mostly sunny
搜索
5:32 PM 5/13/2025



6 附录

关于复现：安装 lab3 cmakelist 中要求的 lib 后 build&make 即可。其中 tabulate 等库在不同 linux 发行版上可能需要手动编译安装。在 build 文件夹中运行 test_all 即可执行所有单元测试。输出 log 将在同一目录。输出的图片将在 build/test/data/对应测试集名目录下。